

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS**

**Разработка умной системы реагирования на оповещения системы мониторинга с
использованием Telegram Bot API и Go**

Обучающийся / Student Лешков Роман Сергеевич

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет

инфокоммуникационных технологий

Группа/Group K34212

Направление подготовки/ Subject area 11.03.02 Инфокоммуникационные технологии и
системы связи

Образовательная программа / Educational program Программирование в
инфокоммуникационных системах 2019

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Бакалавр

Руководитель ВКР/ Thesis supervisor Самохин Никита Юрьевич, Университет ИТМО,
факультет инфокоммуникационных технологий, ассистент (квалификационная категория
"ассистент")

Обучающийся/Student


Документ подписан	
Лешков Роман Сергеевич	
01.04.2023	

(эл. подпись/ signature)

Лешков Роман
Сергеевич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Самохин Никита Юрьевич	
31.03.2023	

(эл. подпись/ signature)

Самохин
Никита
Юрьевич

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS**

Обучающийся / Student Лешков Роман Сергеевич

Факультет/институт/кластер/

Faculty/Institute/Cluster

факультет

инфокоммуникационных технологий

Группа/Group K34212

Направление подготовки/ Subject area 11.03.02 Инфокоммуникационные технологии и системы связи

Образовательная программа / Educational program Программирование в инфокоммуникационных системах 2019

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Бакалавр

Тема ВКР/ Thesis topic Разработка умной системы реагирования на оповещения системы мониторинга с использованием Telegram Bot API и Go

Руководитель ВКР/ Thesis supervisor Самохин Никита Юрьевич, Университет ИТМО, факультет инфокоммуникационных технологий, ассистент (квалификационная категория "ассистент")

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Целью ВКР является разработка умной системы реагирования на оповещения системы мониторинга с использованием Telegram Bot API и Go

Для успешного достижения цели работы были определены следующие задачи: составление технического задания на разработку, проектирование структуры баз данных, разработка серверной части, разработка моделей пользовательского меню в Telegram, тестирование и отладка программного модуля.

Техническое задание на разработку:

Необходимо разработать информационный модуль, позволяющий получать оповещения от системы мониторинга, в зависимости от пользователя в системе мониторинга, и давать пользователю выбор действий для реагирования.

Данный модуль должен позволять пользователям изменить статус оповещения на "В процессе решения", "Решено", предлагать список скриптов пользователя для решения проблемы в оповещении, давать возможность оставлять комментарий о проделанной работе, и также переадресовать оповещение для решения другому сотруднику.

Список действий пользователя должен быть определен уровнем доступа к узлу, от которого пришло оповещение. Все действия по решению проблемы оповещения должны быть сохранены в базу данных.

Модуль должен быть разработан с помощью стека технологий Telegram Bot API и Go, в качестве системы мониторинга использована система мониторинга Zabbix.

Форма представления материалов ВКР / Format(s) of thesis materials:

В результате выполнения ВКР будут представлены следующие материалы: блок-схема спроектированной структуры базы данных, основные фрагменты компьютерного кода, необходимые для понимания функционирования информационного модуля, снимки экрана, демонстрирующие пользовательский интерфейс Telegram-бота, а также сам модуль с тестовыми данными в таблицах базы данных для демонстрации реализованного функционала.

Дата выдачи задания / Assignment issued on: 30.01.2023

Срок представления готовой ВКР / Deadline for final edition of the thesis 15.05.2023

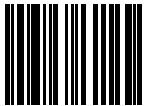
Характеристика темы ВКР / Description of thesis subject (topic)

Тема в области фундаментальных исследований / Subject of fundamental research: нет / not

Тема в области прикладных исследований / Subject of applied research: да / yes

СОГЛАСОВАНО / AGREED:

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Самохин Никита Юрьевич	
31.03.2023	

(эл. подпись)

Самохин
Никита
Юрьевич

Задание принял к
исполнению/ Objectives
assumed BY

Документ подписан	
Лешков Роман Сергеевич	
31.03.2023	

(эл. подпись)

Лешков Роман
Сергеевич

Руководитель ОП/ Head
of educational program

Документ подписан	
Зудилова Татьяна Викторовна	
22.05.2023	

(эл. подпись)

Зудилова
Татьяна
Викторовна

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Лешков Роман Сергеевич

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет
инфокоммуникационных технологий

Группа/Group K34212

Направление подготовки/ Subject area 11.03.02 Инфокоммуникационные технологии и
системы связи

Образовательная программа / Educational program Программирование в
инфокоммуникационных системах 2019

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Бакалавр

Тема ВКР/ Thesis topic Разработка умной системы реагирования на оповещения системы
мониторинга с использованием Telegram Bot API и Go

Руководитель ВКР/ Thesis supervisor Самохин Никита Юрьевич, Университет ИТМО,
факультет инфокоммуникационных технологий, ассистент (квалификационная категория
"ассистент")

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Разработка информационного модуля умной системы реагирования на оповещения
системы мониторинга с использованием Telegram Bot API.

Задачи, решаемые в ВКР / Research tasks

1) Моделирование пользовательского интерфейса в Telegram-чате, 2) Проектирование базы
данных информационного модуля, 3) Разработка системы получения оповещений от
системы мониторинга, 4) Разработка системы отправки оповещений пользователям,
ответственным за узел, 5) Разработка системы реагирования, согласно действиям
пользователя в Telegram-боте.

Краткая характеристика полученных результатов / Short summary of results/findings

Разработан информационный модуль, который получает оповещения от системы
мониторинга, и отправляет оповещения через Telegram Bot пользователям, ответственным за
узел от которого пришло оповещение, и предлагает список действий для реагирования.

Обучающийся/Student

Документ подписан	
----------------------	--

	
Лешков Роман Сергеевич	
01.04.2023	

(эл. подпись/ signature)

Лешков Роман
Сергеевич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Самохин Никита Юрьевич	
31.03.2023	

(эл. подпись/ signature)

Самохин
Никита
Юрьевич

(Фамилия И.О./ name
and surname)

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	6
ВВЕДЕНИЕ.....	7
1 АНАЛИЗ ДОКУМЕНТАЦИИ	10
1.1 ZABBIX	10
1.2 ANSIBLE	12
1.3 TELEGRAM BOT API.....	12
2 РАЗВЕРТЫВАНИЕ СИСТЕМЫ МОНИТОРИНГА.....	15
2.1 РАЗВЕРТЫВАНИЕ ZABBIX-СЕРВЕРА	15
2.2 РАЗВЕРТЫВАНИЕ ANSIBLE.....	18
2.3 НАПИСАНИЕ ПЛЕЙБУКА ANSIBLE	19
2.4 НАСТРОЙКА АВТОРЕГИСТРАЦИИ ХОСТОВ В ZABBIX.....	24
2.5 ПОЛЬЗОВАТЕЛЬСКИЕ ПАРАМЕТРЫ ZABBIX.....	27
3 РАЗРАБОТКА TELEGRAM-БОТА.....	31
3.1 ОПОВЕЩЕНИЯ ZABBIX ЧЕРЕЗ TELEGRAM-БОТ	31
3.2 РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА.....	33
3.3 СИСТЕМА ДЕЙСТВИЙ ПРИ НАЖАТИИ КНОПОК	35
3.4 НАПИСАНИЕ ЗАПРОСОВ ДЛЯ TELEGRAM BOT API	36
3.5 НАПИСАНИЕ ЗАПРОСОВ ДЛЯ ZABBIX API	38
3.6 ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО TELEGRAM-БОТА ...	40
ЗАКЛЮЧЕНИЕ	42
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	43

ВВЕДЕНИЕ

В любой современной IT-инфраструктуре используется множество серверов, виртуальных машин, контейнеров, сетевых устройств и других хостов. Для поддержания работоспособности инфраструктуры необходимо вести мониторинг хостов для отслеживания нагрузки устройств, обнаружения проблем и уведомления ответственных лиц для скорейшего решения проблем. При этом скорость реагирования на обнаружение проблемы также важна, поэтому кроме системы мониторинга должны быть настроены пути доставки оповещений.

Zabbix – система мониторинга с открытым исходным кодом, имеющая множество возможностей для решения задач корпоративного класса. Система Zabbix имеет структуру сервер-агент: для связи с сервера с хостом, на хосте должен быть установлен Zabbix-агент. Zabbix-сервер отправляет запросы хостам, агент собирает запрошенные данные и отдает их на сервер, сервер записывает полученные данные в свою базу данных.

Система мониторинга Zabbix позволяет использовать шаблоны для указания параметров, какие параметры должны собираться с хостов, а также система позволяет настроить авторегистрацию хостов, что позволяет автоматизировать настройку мониторинга при первоначальной настройке сети или при её масштабировании.

Основные функции Zabbix заключаются в сборе данных, настраиваемые триггеры обнаружения проблем, настраиваемые оповещения, представление графиков по полученным от хостов данным, логирование проблем и их решений, что отвечает всем запросам к системе мониторинга.

Так как Zabbix имеет структуру сервер-агент, то должен быть решен вопрос с доставкой приложения агента на хост и его настройкой. Одним из возможных решений является использование Ansible.

Ansible – система управления конфигурациями с открытым исходным кодом, использующая для подключения SSH, поэтому для его использования

не нужны приложения-агенты на хостах. Ansible используется для решения задач развертывания сервисов и их настройки.

Telegram – кроссплатформенная система с функциями мгновенного обмена текстовыми, голосовыми и видеосообщениями. В Telegram поддерживается создание небольших приложений, которые способны выполнять разнообразные задачи – Telegram-боты.

Telegram-боты подключены к серверу владельца, который обрабатывает входящие запросы от пользователей. Telegram выступает в роли клиентской части приложения, размещенного на сервере. При этом соединение Telegram-сервера с сервером разработчика происходит через https соединение, а для управления Telegram-ботом используется Telegram Bot API.

Golang – язык программирования с открытым исходным кодом, который поддерживает Google. Имеет широкое применение в облачных и сетевых сервисах, для веб-разработки, консольных приложений, в сфере DevOps и SRE.

В данной работе описывается процесс развертывания и настройки системы мониторинга Zabbix и разработки Telegram-бота, который по предоставленным пользователем данным осуществляет функционал для реагирования на оповещения от системы мониторинга.

Данная работа является актуальной, так как мониторинг состояния и проблем позволяет уменьшить время реагирования на проблему, что повысит эффективность инфраструктуры, или оптимизировать затраты на обновление оборудования: обновить комплектующие хоста, которому не хватает ресурсов для выполнения поставленных задач. А в современных реалиях количество хостов может исчисляться десятками тысяч, что исключает вариант ручной настройки, следовательно лучшим вариантом является автоматическое развертывание системы мониторинга с автонастройкой агентов и авторегистрацией хостов.

Целью практической работы является развертывание системы мониторинга Zabbix с использованием Ansible и реализация умной системы

реагирования на оповещения системы мониторинга с использованием Telegram Bot API и языка программирования Golang.

Для достижения поставленной цели необходимо решить следующие задачи:

- Анализ документации Telegram Bot API
- Развертывание и настройка системы мониторинга Zabbix
- Настройка пути оповещений Zabbix через Telegram-бот
- Разработка пользовательского интерфейса для Telegram-чата
- Разработка системы действий на основе нажатия кнопок в Telegram-чате
- Написание запросов для Telegram Bot API
- Написание запросов для Zabbix API
- Тестирование разработанного Telegram-бота

1 АНАЛИЗ ДОКУМЕНТАЦИИ

В данном разделе описаны основные шаги для развертывания и использования Zabbix и Ansible, а также структуры и принципы их работы необходимые для автоматизированного развертывания системы мониторинга.

1.1 ZABBIX

Вся документация для системы мониторинга Zabbix располагается на официальном сайте Zabbix[1]. Начало работы с ботом начинается с загрузки пакетов. На странице загрузки продукта Zabbix есть удобная система выбора пакетов для Zabbix с выводом необходимых для развертывания и конфигурирования команд: в первом пункте выбирается версия Zabbix, дистрибутив системы, на которой будет развернут Zabbix, версия ОС, компонент для установки, база данных, которая будет использована для хранения данных Zabbix и движок веб-сервера.

Во втором шаге описаны команды для установки репозитория по выбранным в первом шаге параметрам, установки выбранных пакетов Zabbix из установленного репозитория. Также указаны команды для создания базы данных на ранее указанном сервере базы данных и команда для импортирования данных для созданной базы данных. Затем указаны конфигурационные файлы для настройки Zabbix-сервера и команды для автозапуска Zabbix-сервера и -агента. После этого веб-сервис Zabbix будет доступен по ранее указанному в конфигурационном файле порту.

У Zabbix существует две версии Zabbix-агента. Вторая версия имеет расширенный функционал по сравнению с первой, поэтому имеет смысл на хостах устанавливать сразу вторую версию, чтобы при необходимости добавления функционала к агенту не пришлось удалять первую версию, устанавливать вторую, снова изменять конфигурационные файлы агента, переносить пользовательские параметры в папку агента второй версии.

У Zabbix есть свой API, который позволяет интегрировать функции Zabbix в стороннее приложение. В Zabbix все API запросы являются POST-запросами, и имеют одинаковую структуру: заголовок для авторизации,

заголовок, который говорит о том, что прикреплен JSON-файл, и сам JSON-файл (Рисунок 1).

```
curl --request POST \
--url 'https://example.com/zabbix/api_jsonrpc.php' \
--header 'Authorization: Bearer ${AUTHORIZATION_TOKEN}' \
--header 'Content-Type: application/json-rpc' \
--data '{"jsonrpc":"2.0","method":"item.create","params":{"name":"Free disk space on /home/joe/","key_":'
```

Рисунок 1 – Структура запроса Zabbix API

В JSON-файле обязаны содержаться параметры "jsonrpc", "method", "params": {} и "id". В первом параметре всегда содержится значение "2.0" – версия протокола JSON-RPC, которым пользуется Zabbix, во втором параметре указывается метод, который будет исполнять в системе Zabbix, в структуре "params" указываются аргументы для метода, указанного во втором параметре, последний обязательный параметр – это ID-пользователя, под которым выполнена авторизация.

Структуру ответа от Zabbix API можно разделить на два вида: один возвращается в случае успешного срабатывания метода, второй в случае ошибки. Оба ответа содержат параметры "jsonrpc" и "id", но отличаются тем, что первый возвращает ответ в структуре "result", а второй возвращает ошибку в структуре "error".

Для реагирования на события в системе мониторинга Zabbix нужно использовать методы "event.get" и "event.acknowledge". Первый метод, соответствуя своему имени, возвращает информацию о событиях, или как еще называют – проблемы. А второй дает возможность обновлять состояние события. В рамках этой работы у метода "event.get" будет указываться только один аргумент – "eventids", который указывает ID-события, информацию о котором ожидается получить. Для "event.acknowledge" будут использованы аргументы "eventids", "action", "message" и "severity": первый указывает ID-события, которое будет обновлено, второй аргумент передает битовую маску, по которой указываются действия для обновления, в рамках работы будут использованы значения: "1" – для закрытия события, "4" – для написания сообщения и "8" – для изменения важности события.

1.2 ANSIBLE

Документация Ansible расположена на официальном сайте Ansible[2]. Установка Ansible может пройти и через стандартные репозитории, и через утилиту Python3 pip, в документации рекомендуется использовать второй вариант.

После развертывания Ansible первым делом настраивается инвентарный файл `/etc/ansible/hosts`, в котором указываются хосты, которые будут настраиваться через Ansible. Далее в документации рекомендуется установить связь по SSH через ключи, чтобы при соединении не требовался ввод пароля, и чтобы пароль не хранился в открытом виде в инвентарном файле.

При использовании Ansible вызывается плейбук – список плеев в заданном порядке, плей в свой очередь – список задач, применяемых к хостам, задачи – списки модулей, которые определяют действия, которые будут совершены на хостах, а модуль – команды, которые будут запущены на хостах. Все модули собраны в коллекции с FQCN.

Стоит добавить, что задачи могут быть включены в роли, которые используются для улучшения структуры и читаемости плейбука.

1.3 TELEGRAM BOT API

Вся документация для Telegram Bot API располагается на официальном сайте Telegram[3]. Начало работы с ботом начинается с регистрации бота. Регистрация происходит через бота @BotFather: запрашиваешь создание нового бота командой `/newBot`, после разработчиком вводится имя бота для управления им, в случае этой работы: `MyZabbixBot`, после этого имя бота в чате будет отображаться как `MyZabbixBot`. После имени запрашивается тэг бота, в данной работе `RLeshZabbixBot`, то есть, чтобы найти этого бота в Telegram нужно искать @RLeshZabbixBot.

После ввода тэга @BotFather присылает сообщение содержащие токен созданного бота. Именно благодаря токену осуществляется управления ботом: все запросы к боту осуществляются по следующему шаблону: https://api.telegram.org/bot<token>/METHOD_NAME.

Для получения обновлений от Telegram-бота на сервер используется метод “getUpdates”, который возвращает массив объектов типа “Update” (Рисунок 2). Для дальнейшей работы выбраны следующие поля “Update”: “update_id” – id обновления, “message” – новое входящее сообщение, “callback_query” – оповещение об использовании интерактивной клавиатуры. В свою очередь поля “message”, “callback_query” имеют свои поля. У типа “Message”: “message_id” – id сообщения, “from” – id и имя пользователя, “chat” – id чата, “text” – текст сообщения, “reply_markup” – прикрепленная интерактивная клавиатура. У типа “CallbackQuery”: id – id оповещения, “from” – id и имя пользователя, “Data” – текст, привязанный к кнопке, и поле “message” – сообщение, к которому прикреплена кнопка.

```
type Update struct {
    UpdateId int           `json:"update_id"`
    Message    Message     `json:"message"`
    CallbackQuery CallbackQuery `json:"callback_query"`
}

type Message struct {
    MessageId int           `json:"message_id"`
    From User           `json:"from"`
    Chat struct {
        Id int           `json:"id"`
    } `json:"chat"`
    Text string           `json:"text"`
    ReplyMarkup InlineKeyboardMarkup `json:"reply_markup"`
}

type CallbackQuery struct {
    Id string           `json:"id"`
    From User           `json:"from"`
    Data string         `json:"data"`
    Message Message     `json:"message"`
}

type User struct {
    Id int           `json:"id"`
    Username string  `json:"username"`
}
```

Рисунок 2 – Структуры для получения обновлений

Модуль клавиатуры представляет из себя таблицу кнопок, которая реализована как массив массивов кнопок, в свою очередь имеет поля: “text” – текст, отображающийся на кнопки и “callback_data” – данные, которые придут на сервер после нажатия на кнопку (Рисунок 3).

```
type InlineKeyboardMarkup struct {
    InlineKeyboard [][]InlineKeyboardButton `json:"inline_keyboard"`
}

type InlineKeyboardButton struct {
    Text string           `json:"text"`
    CallbackData string `json:"callback_data"`
}
```

Рисунок 3 – Структуры для реализации клавиатуры

Поле “message” присутствует в ответе только в случае, если пришло новое сообщение, а поле “callback_query” только в случае нажатия на кнопку.

Для ответа пользователю выбраны три метода “sendMessage”, “editMessageText” и “answerCallbackQuery”. Первый используется для отправки сообщения в чат, для этого нужно передать параметры: id чата, текстовое сообщение и клавиатуру, второй метод для редактирования текста сообщения бота: требует ту же параметры, что и “sendMessage”, и id сообщения для редактирования. Третий метод нужен для того, чтобы дать обратную связь на нажатие кнопки, и для него указываются id оповещения и всплывающий текст.

2 РАЗВЕРТЫВАНИЕ СИСТЕМЫ МОНИТОРИНГА

2.1 РАЗВЕРТЫВАНИЕ ZABBIX-СЕРВЕРА

В качестве дистрибутива ОС выбран Debian 11. На нем будут развернуты система мониторинга Zabbix, его база данных на PostgreSQL, веб-сервер на Nginx и система управления конфигурациями Ansible.

PostgreSQL[4] – это объективно-реляционная система управления базами данных с открытым исходным кодом, которая имеет множество функций стандарта SQL и дает возможность создавать свои типы данных, функции, операторы, методы индексирования и процедурные языки. Эта СУБД имеет большое комьюнити и продолжает улучшаться, также имеет высокую производительность и масштабируемость. Из-за своей производительности и гибкости включена во многие docker-образы, что говорит ее надежности.

Nginx[5] – веб-сервер и прокси-сервер, основной задачей которого является обслуживание статических запросов. Для обработки динамических запросов используется вместе с php7.4-fpm – интерпретатором языка программирования PHP.

После установки ОС в терминале выполняются команды:

Для установки репозитория Zabbix:

```
# wget
https://repo.zabbix.com/zabbix/6.4/debian/pool/main/z/z
abbix-release/zabbix-release_6.4-1+debian11_all.deb
# dpkg -i zabbix-release_6.4-1+debian11_all.deb
# apt update
```

Для установки Zabbix-сервера, веб-сервера и агента:

```
# sudo apt install zabbix-server-pgsql zabbix-frontend-
php php7.4-pgsql zabbix-nginx-conf zabbix-sql-scripts
zabbix-agent
```

Для установки PostgreSQL и создания базы данных:

```
# sudo apt -y install postgresql
# sudo -u postgres createuser --pwprompt zabbix
# вводится пароль
# sudo -u postgres createdb -O zabbix zabbix
```

Для импорта схемы и данных Zabbix в созданную базу данных:

```
# zcat /usr/share/zabbix-sql-  
scripts/postgresql/server.sql.gz | sudo -u zabbix psql  
zabbix
```

Далее в файле `/etc/zabbix/zabbix_server.conf` редактируется строка с ключом `DBPassword` – в качестве параметра указывается пароль пользователя `zabbix` в формате `DBPassword=<password>`.

А в файле `/etc/zabbix/nginx.conf` задается порт и имя сервера:

```
listen 8080;  
server_name zabbix.lan;
```

Далее вводятся команды для рестарта сервисов и автозапуска Zabbix с ОС:

```
# systemctl restart zabbix-server zabbix-agent nginx  
php7.4-fpm  
# systemctl enable zabbix-server zabbix-agent nginx  
php7.4-fpm
```

После этого на порту 8080 стал доступен веб-интерфейс системы мониторинга Zabbix. При первом посещении будет запрошен доступ к базе данных. И после успешного подключения к ней, при перезагрузке страницы будет показано окно авторизации (Рисунок 4)

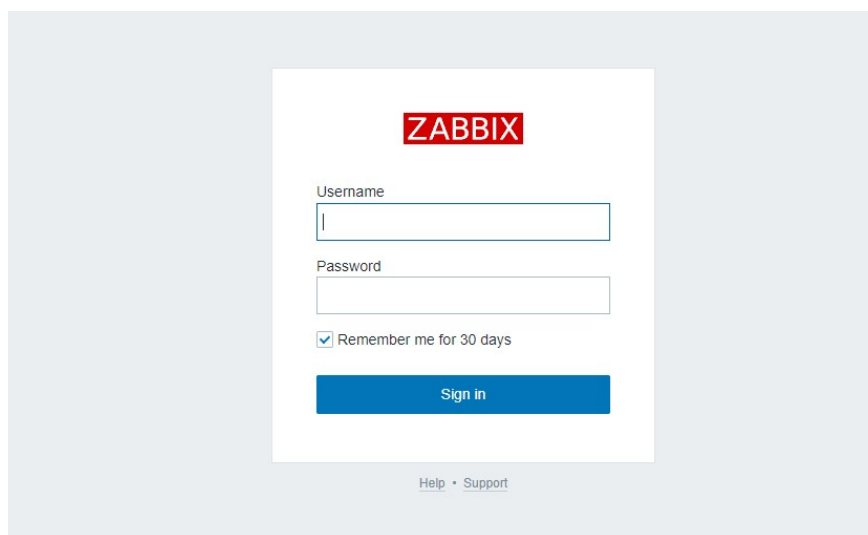


Рисунок 4 – Меню авторизации Zabbix

После авторизации под стандартным пользователем `“Admin:zabbix”` открывается стандартный дашборд Zabbix (Рисунок 5).

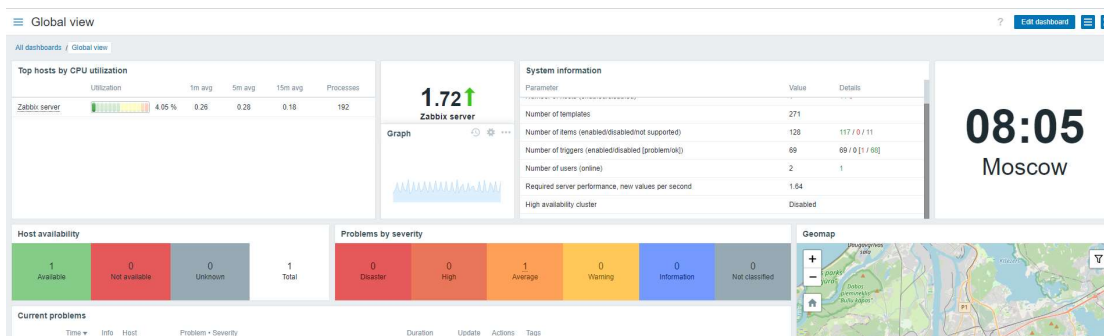


Рисунок 5 – Базовый дашборд Zabbix

В числе хостов указан только сам сервер (Рисунок 6). К хосту применены шаблоны “Linux by Zabbix agent” и “Zabbix server health”, которые показывают какие параметры сервер должен спрашивать с хоста. Такие параметры в Zabbix называются итемами. Zabbix-сервер включен в группу хостов “Zabbix servers” для удобства управления.

Рисунок 6 – Базовая настройка Zabbix-сервера

Базовая настройка Zabbix-сервера снимает 128 итемов и отслеживает 69 триггеров – условий, при которых срабатывает определенное действие, чаще всего – уведомления пользователя.

Среди отслеживаемых параметров есть такие, как количество свободной оперативной памяти в МБ и в процентах, загрузка ЦПУ, дисковое пространство каждого раздела и многие другие (Рисунок 7). Также на множество параметров отслеживаются триггерами, которые вызывают оповещение с разным уровнем строгости.

<input type="checkbox"/>	Linux by Zabbix agent: Available memory	Triggers 1	vm.memory.size[available]	1m	7d	365d	Zabbix agent	Enabled
Average	Linux by Zabbix agent: Lack of available memory		max(Zabbix server/vm.memory.size[available],5m)<({MEMORY_AVAILABLE_MIN}) and last(Zabbix server/vm.memory.size[total])>0					Enabled
<input type="checkbox"/>	Linux by Zabbix agent: CPU guest nice time		system.cpu.util[,guest_nice]	1m	7d	365d	Zabbix agent	Enabled
<input type="checkbox"/>	Linux by Zabbix agent: CPU guest time		system.cpu.util[,guest]	1m	7d	365d	Zabbix agent	Enabled
<input type="checkbox"/>	Linux by Zabbix agent: CPU idle time		system.cpu.util[,idle]	1m	7d	365d	Zabbix agent	Enabled
<input type="checkbox"/>	Linux by Zabbix agent: CPU interrupt time		system.cpu.util[,interrupt]	1m	7d	365d	Zabbix agent	Enabled
<input type="checkbox"/>	Linux by Zabbix agent: CPU iowait time		system.cpu.util[,iowait]	1m	7d	365d	Zabbix agent	Enabled
<input type="checkbox"/>	Linux by Zabbix agent: CPU nice time		system.cpu.util[,nice]	1m	7d	365d	Zabbix agent	Enabled
<input type="checkbox"/>	Linux by Zabbix agent: CPU softirq time		system.cpu.util[,softirq]	1m	7d	365d	Zabbix agent	Enabled
<input type="checkbox"/>	Linux by Zabbix agent: CPU steal time		system.cpu.util[,steal]	1m	7d	365d	Zabbix agent	Enabled
<input type="checkbox"/>	Linux by Zabbix agent: CPU system time		system.cpu.util[,system]	1m	7d	365d	Zabbix agent	Enabled
<input type="checkbox"/>	Linux by Zabbix agent: CPU user time		system.cpu.util[,user]	1m	7d	365d	Zabbix agent	Enabled
<input type="checkbox"/>	Linux by Zabbix agent: CPU idle time: CPU utilization	Triggers 1	system.cpu.util		7d	365d	Dependent	Enabled
<input type="checkbox"/>	Linux by Zabbix agent: Free sw							
Warning	Linux by Zabbix agent: High CPU utilization		min(Zabbix server/system.cpu.util,5m)>({CPU_UTIL_CRIT})					Enabled

Рисунок 7 – Стандартные итемы и триггеры

2.2 РАЗВЕРТЫВАНИЕ ANSIBLE

Установка Ansible производится из стандартных репозиториев:

```
# sudo apt update
# sudo apt install ansible
```

Для демонстрации развернуты три виртуальные машины на Ubuntu 22.04. Единственное, что нужно для начала работы Ansible с хостами – создание пользователя для доступа по SSH с достаточными привилегиями для выполнения команд. На тестовых хостах создан пользователь sshUser:sshUser командами:

```
# sudo useradd -m sshUser
# echo -e "sshUser\nsshUser" | sudo passwd sshUser
```

Кроме этого, настроено ssh-соединение через ключи, чтобы при запуске плейбука не было необходимости вводить пароль. Для этого генерируются ключи на Ansible-сервере командой:

```
# sudo -u sshUser ssh-keygen
```

Для копирования публичного ключа на хосты использованы утилита sshpass и ssh-copy-id. Создается скрипт deliverSshKey.sh, в котором берутся хосты из инвентарного файла Ansible группы ubuntuServers для пересылки ssh-ключа (Рисунок 8).

```
GNU nano 5.4 deliverSshkey.sh
#!/bin/bash

servers=$(ansible ubuntuServers --list-hosts | tail +2)
echo $servers
for line in $servers
do
    echo "running $line"
    sshpass -p sshUser ssh-copy-id sshUser@$line
done
```

Рисунок 8 – Скрипт для отправки публичного ssh-ключа

2.3 НАПИСАНИЕ ПЛЕЙБУКА ANSIBLE

Далее идет настройка файла /etc/ansible/hosts. В нем указываются хосты для Ansible, объединение их в группы и, при необходимости, дополнительные параметры. Созданы две группы хостов zabbixServers и ubuntuServers, в них перечислены хосты с использованием их имен, хотя есть возможность добавить их и через IP-адрес (Рисунок 9).

```
GNU nano 5.4 hosts *
[zabbixServers]
zabbix-server[]
[ubuntuServers]
ubuntuServer101
ubuntuServer102
ubuntuServer103
```

Рисунок 9 – Файл инвентаря Ansible

Далее будет создана директория для роли zabbix-agent, для этого используется утилита ansible-galaxy, а вызывается она командой:

```
# sudo ansible-galaxy init zabbix-agent
```

После этого в текущей директории создается директория zabbix-agent с структурой, содержащей папки files, tasks, templates, vars (Рисунок 10).

```
rlesh@zabbix-server:/etc/ansible$ ls -la zabbix-agent
total 44
drwxr-xr-x 10 root root 4096 May 22 09:35 .
drwxr-xr-x  3 root root 4096 May 24 19:44 ..
drwxr-xr-x  2 root root 4096 May 19 17:25 defaults
drwxr-xr-x  3 root root 4096 May 22 09:45 files
drwxr-xr-x  2 root root 4096 May 19 17:25 handlers
drwxr-xr-x  2 root root 4096 May 19 17:25 meta
-rw-r--r--  1 root root 1328 May 19 17:25 README.md
drwxr-xr-x  2 root root 4096 May 22 09:49 tasks
drwxr-xr-x  2 root root 4096 May 22 09:03 templates
drwxr-xr-x  2 root root 4096 May 19 17:25 tests
drwxr-xr-x  2 root root 4096 May 21 21:15 vars
```

Рисунок 10 – Структура директории роли zabbix-agent

В папке tasks в файл main.yml указываются задачи, которые выполнит роль. Первым делом нужно установить репозиторий Zabbix и установить пакет zabbix-agent2. Для этого используется встроенный модуль ansible.builtin.apt. В первой задаче устанавливается официальный репозиторий Zabbix (Рисунок 11).

```
- name: Install zabbix 6.4 repo .deb
  ansible.builtin.apt:
    deb: https://repo.zabbix.com/zabbix/6.4/ubuntu/pool/main/z/zabbix-release/zabbix-release_6.4-1+ubuntu22.04_all.deb
```

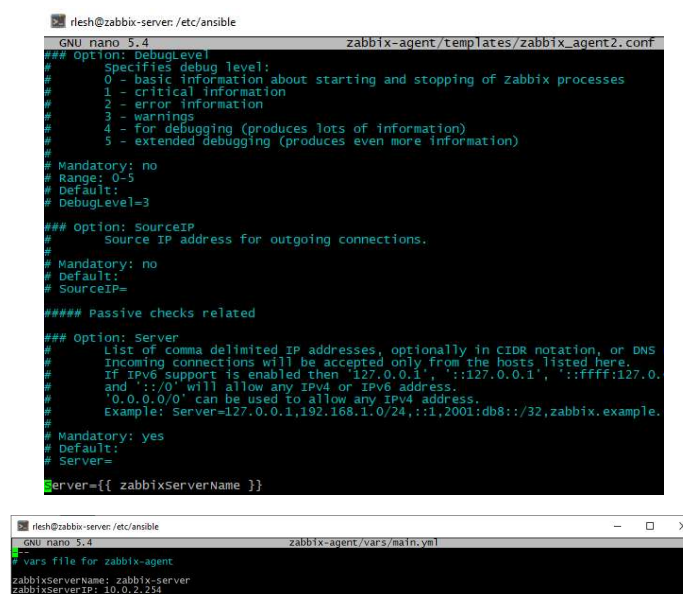
Рисунок 11 – Задача для установки репозитория Zabbix

Во второй задаче устанавливается пакет zabbix-agent2, с указанием версии пакета и с обновлением списка пакетов (Рисунок 12).

```
- name: Install zabbix-agent2 6.4
  ansible.builtin.apt:
    name: zabbix-agent2
    state: present
    update_cache: yes
```

Рисунок 12 – Задача для установки zabbix-agent2

Далее в папке templates копируется файл конфигурации zabbix_agent2.conf. В нем необходимо добавить адрес Zabbix-сервера в белый список. Для этого в параметре “Server” указывается IP-адрес или имя сервера. Для более гибкой настройки лучше указать переменную, например так: Server={{ zabbixServerName }}, значение в двойных фигурных скобках берется из файла vars/main.yml (Рисунок 13).



```
GNU nano 5.4 zabbix-agent/templates/zabbix_agent2.conf
### Option: DebugLevel
# Specifies debug level:
# 0 - basic information about starting and stopping of Zabbix processes
# 1 - critical information
# 2 - error information
# 3 - warnings
# 4 - for debugging (produces lots of information)
# 5 - extended debugging (produces even more information)
# Mandatory: no
# Range: 0-5
# Default:
# DebugLevel=3

### Option: SourceIP
# Source IP address for outgoing connections.
# Mandatory: no
# Default:
# SourceIP=

#### Passive checks related

### Option: Server
# List of comma delimited IP addresses, optionally in CIDR notation, or DNS
# Incoming connections will be accepted only from the hosts listed here.
# If IPv6 support is enabled then '127.0.0.1', '::127.0.0.1', '::ffff:127.0.
# and ':::0' will allow any IPv4 or IPv6 address.
# '0.0.0.0/0' can be used to allow any IPv4 address.
# Example: Server=127.0.0.1,192.168.1.0/24,::1,2001:db8::/32,zabbix.example.
# Mandatory: yes
# Default:
# Server=
Server={{ zabbixServerName }}
```

```
GNU nano 5.4 zabbix-agent/vars/main.yml
vars file for zabbix-agent
zabbixServerName: zabbix-server
zabbixServerIP: 10.0.2.254
```

Рисунок 132 – Использование шаблона в Ansible

Для доставки конфигурационного файла на хосты пишется еще одна задача, использующая встроенный модуль `ansible.builtin.template`, в качестве параметров указываются источник – `zabbix_agent2.conf`, в виде шаблона, место назначения на хостах – `/etc/zabbix/zabbix_agent2.conf`, а также параметры доступа к перенесенному файлу (Рисунок 14).

```
- name: Deliver zabbix_agent2.conf
  ansible.builtin.template:
    src: zabbix_agent2.conf
    dest: /etc/zabbix/zabbix_agent2.conf
    owner: root
    group: root
    mode: '0644'
```

Рисунок 3 – Задача для доставки конфигурационного файла на хосты

Так как в качестве адреса сервера указано его имя, а DNS в тестовой среде не поднят, то для того, чтобы хосты могли получить адрес `zabbix-сервера` по его имени, нужно добавить строку в файл `/etc/hosts` на хостах, содержащую адрес сервера и его имя. Для этого использован модуль `ansible.builtin.lineinfile`, в параметрах указывается путь к файлу на хостах – `/etc/hosts`, строка, которая будет добавлена – `{{ zabbixServerIP }} {{ zabbixServerName }}`, и параметры доступа к файлу (Рисунок 15). Переменные в фигурных скобках берутся из файла `vars/main.yml`.

```
- name: Add zabbix-server name to /etc/hosts
  ansible.builtin.lineinfile:
    path: /etc/hosts
    line: '{{ zabbixServerIP }} {{ zabbixServerName }}'
    owner: root
    group: root
    mode: '0644'
```

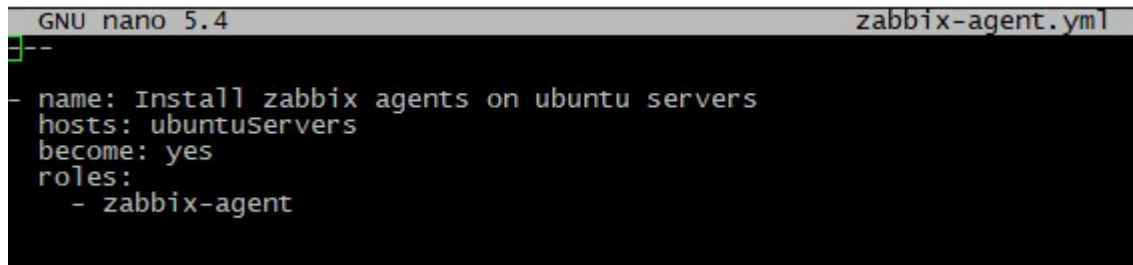
Рисунок 4 – Задача для добавления строки в файл `/etc/hosts`

После установки доставки конфигурационного файла следует перезапустить `Zabbix-агента`, для этого у `Ansible` используется модуль `ansible.builtin.systemd`. В качестве параметров указываются состояние – `restarted`, имя утилиты – `zabbix-agent2`, и автозапуск при старте ОС (Рисунок 16).

```
- name: Restarting zabbix-agent2
  ansible.builtin.systemd:
    state: restarted
    name: zabbix-agent2
    enabled: yes
```

Рисунок 5 – Задача для перезапуска `zabbix-agent2`

Далее создан плейбук `zabbix-agent.yml`, в котором содержится один плей, в котором указана группа хостов, на которых будут выполнены задачи – `ubuntuServers`, параметр для повышения полномочий “`become: yes`”, и роль для исполнения – `zabbix-agent` (Рисунок 17).



```

GNU nano 5.4 zabbix-agent.yml
--
- name: Install zabbix agents on ubuntu servers
  hosts: ubuntuServers
  become: yes
  roles:
    - zabbix-agent

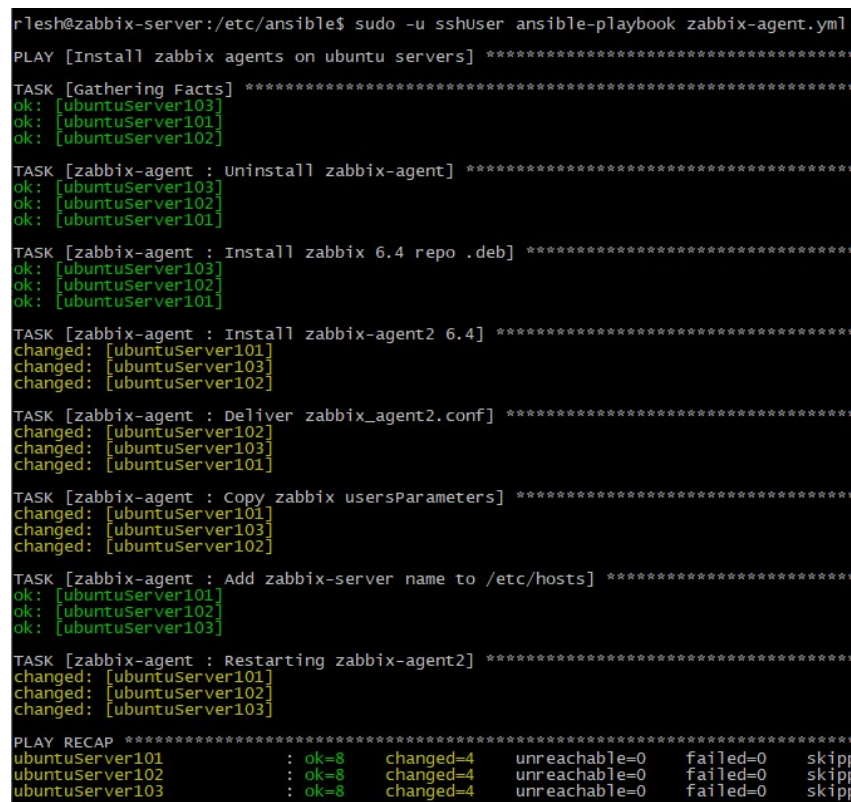
```

Рисунок 6 – Плейбук для установки `zabbix-agent2` на хосты

Для запуска плейбука используется команда:

```
# sudo -u sshUser ansible-playbook zabbix-agent
```

В терминале будет показан статус выполнения задач, в случае успешной отработки, статусы будут находиться в состоянии “`ok`”, либо в состоянии “`changed`”: первое для задач, которые выполнены и не принесли изменений в конфигурацию хоста, а второе для изменивших конфигурацию (Рисунок 18).



```

rlesh@zabbix-server:/etc/ansible$ sudo -u sshuser ansible-playbook zabbix-agent.yml
PLAY [Install zabbix agents on ubuntu servers] *****
TASK [Gathering Facts] *****
ok: [ubuntuServer103]
ok: [ubuntuServer101]
ok: [ubuntuServer102]
TASK [zabbix-agent : uninstall zabbix-agent] *****
ok: [ubuntuServer103]
ok: [ubuntuServer102]
ok: [ubuntuServer101]
TASK [zabbix-agent : Install zabbix 6.4 repo .deb] *****
ok: [ubuntuServer103]
ok: [ubuntuServer102]
ok: [ubuntuServer101]
TASK [zabbix-agent : Install zabbix-agent2 6.4] *****
changed: [ubuntuServer101]
changed: [ubuntuServer103]
changed: [ubuntuServer102]
TASK [zabbix-agent : Deliver zabbix_agent2.conf] *****
changed: [ubuntuServer102]
changed: [ubuntuServer103]
changed: [ubuntuServer101]
TASK [zabbix-agent : Copy zabbix usersParameters] *****
changed: [ubuntuServer101]
changed: [ubuntuServer103]
changed: [ubuntuServer102]
TASK [zabbix-agent : Add zabbix-server name to /etc/hosts] *****
ok: [ubuntuServer101]
ok: [ubuntuServer102]
ok: [ubuntuServer103]
TASK [zabbix-agent : Restarting zabbix-agent2] *****
changed: [ubuntuServer101]
changed: [ubuntuServer102]
changed: [ubuntuServer103]
PLAY RECAP *****
ubuntuServer101 : ok=8 changed=4 unreachable=0 failed=0 skip
ubuntuServer102 : ok=8 changed=4 unreachable=0 failed=0 skip
ubuntuServer103 : ok=8 changed=4 unreachable=0 failed=0 skip

```

Рисунок 7 – Успешно выполненный плейбук

После этого появилась возможность добавлять хосты вручную, через веб-интерфейс Zabbix. Для добавления хоста вручную нужно перейти в пункт меню “Data collection/Hosts” и нажать на кнопку “Create host” (Рисунок 19). После ввода имени хоста, его группы и интерфейса, есть возможность применить шаблон для опроса итемов. Zabbix имеет свою большую библиотеку шаблонов, которые охватывают большую часть параметров. Для машин на Linux для простого мониторинга подходит шаблон “Linux by Zabbix agent”.

В качестве имени указано “ubuntuServer101”, шаблон – “Linux by Zabbix agent” и создана новая группа хостов – “ubuntuServers”, интерфейс zabbix-агент с указанием IP.

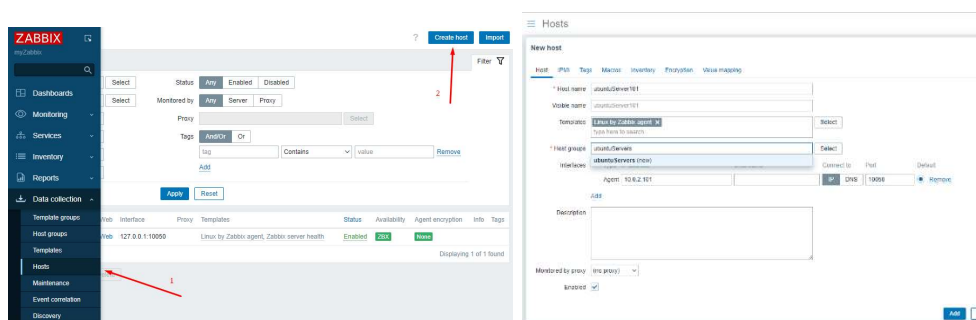


Рисунок 8 – Интерфейс добавления хоста

После нажатия кнопки “add”, новый хост появится в списке, и после первого опроса параметров от сервера поле “Availability” загорится зеленым (Рисунок 20).

<input type="checkbox"/>	Name ▲	Items	Triggers	Graphs	Discovery	Web	Interface	Proxy	Templates	Status	Availability	Agent encryption
<input type="checkbox"/>	ubuntuServer101	Items 55	Triggers 25	Graphs 10	Discovery 3	Web	10.0.2.101:10050		Linux by Zabbix agent	Enabled	ZBX	None

Рисунок 20 – Новый хост в списке

Полученные параметры можно проверить в меню “Monitoring/Hosts”, в виде значений или дашбордов, например, по умолчанию доступен дашборд “System performance” (Рисунок 21), отрисовывающий графики использования системных ресурсов.

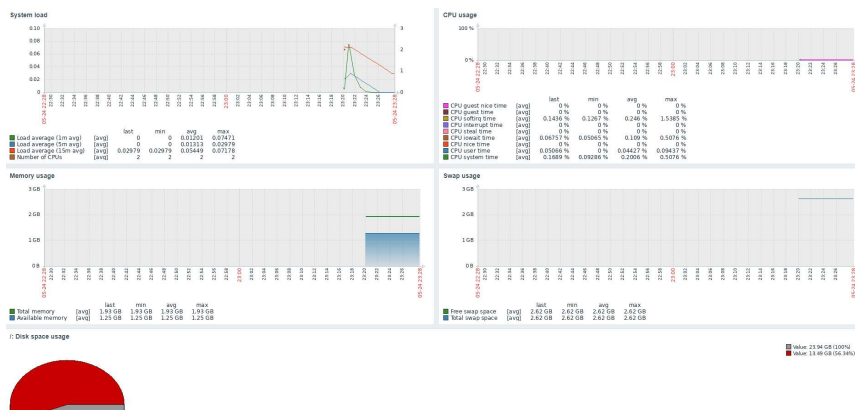


Рисунок 21 – Дашборд для нового хоста

2.4 НАСТРОЙКА АВТОРЕГИСТРАЦИИ ХОСТОВ В ZABBIX

Так как количество хостов в современной инфраструктуре может насчитывать десятки тысяч устройств, то добавлять вручную хосты в систему мониторинга неэффективно. Для решения этого вопроса в zabbix предусмотрены “Discovery rules” и “Autoregistration actions”.

В “Discovery rules” задаются параметры для исследования сети, такие как IP-диапазон, проверку на наличие сервиса или определенного ключа Zabbix, а также какой параметр будет использовать в качестве имени хоста (Рисунок 22).

Discovery rules

Name: Local network

Discovery by proxy: No proxy

IP range: 10.0.2.100-150

Update interval: 1h

Checks:

Type	Actions
Zabbix agent "system.uname"	Edit Remove

Device uniqueness criteria:

☒ IP address

☐ Zabbix agent "system.uname"

Host name:

☒ DNS name

☐ IP address

☐ Zabbix agent "system.uname"

Visible name:

☒ Host name

☐ DNS name

☐ IP address

☐ Zabbix agent "system.uname"

Enabled: ☒

[Update](#) [Clone](#) [Delete](#) [Cancel](#)

Рисунок 22 – Параметры Discovery rule

В “Autoregistration actions” указываются условия срабатывания этого действия этого действия и операции, которые выполняются при выполнении условий. В качестве условия указано содержание в метаданных хоста строки “ubuntuServer”, а в качестве операций происходит добавление хоста в группу “ubuntuServers” и присоединение шаблона “ubuntuServerTemplate” (Рисунок 23).

The image shows two screenshots of a web interface for configuring 'Autoregistration actions'.

The top screenshot shows the 'Action' tab. It has a text input field for 'Name' containing 'Autoregistration ubuntuServers in local network'. Below it is a 'Conditions' section with a table:

Label	Name	Action
A	Host metadata contains ubuntuServer	Remove

Below the table is an 'Add' link. At the bottom, there is an 'Enabled' checkbox which is checked, and a note: '* At least one operation must exist.' At the very bottom are buttons: 'Update', 'Clone', 'Delete', and a refresh icon.

The bottom screenshot shows the 'Operations 2' tab. It has a 'Details' section with two rows:

Add to host groups: ubuntuServers	Edit Remove
Link to templates: ubuntuServerTemplate	Edit Remove

Below the table is an 'Add' link. At the bottom, there is a note: '* At least one operation must exist.' At the very bottom are buttons: 'Update', 'Clone', 'Delete', and a refresh icon.

Рисунок 23 – Параметры “Autoregistration action”

Для завершения настройки необходимо в конфигурационном файле zabbix-агента необходимо указать сервер для активных проверок: указать параметр “ServerActive={{ zabbixServerName }}" и указать “HostMetadata=ubuntuServer” (Рисунок 24). После этого еще раз запускается плейбук zabbix-agent.yml.

```

ServerActive={{ zabbixServerName }}

### option: Hostname
# List of comma delimited unique, case sensitive strings
# Required for active checks and must match the host's actual name
# Value is acquired from HostnameItem if used
# Mandatory: no
# Default:
# Hostname=

### option: HostnameItem
# Item used for generating Hostname if it is used
# Does not support UserParameters or aliases
# Mandatory: no
# Default:
# HostnameItem=system.hostname

### option: HostMetadata
# Optional parameter that defines host metadata
# Host metadata is used at host auto-registration
# An agent will issue an error and not start if not defined, value will be acquired from the template
# Mandatory: no
# Range: 0-2034 bytes
# Default:
# HostMetadata=

HostMetadata=ubuntuServer

```

Рисунок 24 – Обновления конфигурационного файла zabbix-агента

После обновления конфигурационного файла в списке хостов добавляются все хосты из указанного диапазона, имеющее в параметре HostMetadata строку ubuntuServer. Полученные хосты включены в группы “ubuntuServers” и “Discovered hosts”, и подключены к шаблону “ubuntuServerTemplate” (Рисунок 25).

The image shows two parts of the Zabbix web interface. The top part is the 'Hosts' list, which displays a table of hosts with columns for Name, Item, Template, Groups, IP, and Status. The bottom part is the 'Host' configuration page for 'ubuntuServer101', showing fields for Host name, Visible name, Templates, Host groups, Interfaces, and Description.

Name	Item	Template	Groups	IP	Status
ubuntuServer101	Item 1	Template 1	Groups 1	10.0.2.101	Enabled
ubuntuServer102	Item 2	Template 2	Groups 2	10.0.2.102	Enabled
ubuntuServer103	Item 3	Template 3	Groups 3	10.0.2.103	Enabled
ubuntuServer104	Item 4	Template 4	Groups 4	10.0.2.104	Enabled

Host configuration details:

- Host name: ubuntuServer101
- Visible name: ubuntuServer101
- Templates: ubuntuServerTemplate
- Host groups: Discovered hosts, ubuntuServers
- Interfaces: Agent (10.0.2.101), DNS (ubuntuServer101), Connect to (IP), Port (10050)
- Description: (empty text area)
- Monitored by proxy: (no proxy)
- Enabled: ☒

Рисунок 25 – Хосты, добавленные с использованием авторегистрации

2.5 ПОЛЬЗОВАТЕЛЬСКИЕ ПАРАМЕТРЫ ZABBIX

Хотя в Zabbix имеется множество различных шаблонов с большим количеством итемов, пользователям может понадобиться ввести свои собственные итемы. Для этого в Zabbix добавлена возможность создавать `usersParameters`.

Для этого в папке zabbix-агента создается файл с расширением “.conf”. В этом файле в формате “`UserParameter=<key>[*], command <arguments>`”. Максимальное количество аргументов, которое может быть передано через Zabbix, равно девяти. Для демонстрации созданы три параметра: `executeBashCommand`, `pythonTestScript`, `createZabbixItems` (Рисунок 26).

```
rlesh@zabbix-server:/etc/ansible/zabbix-agent/files$ cat zabbix_agent2.d/testUserParametr.conf
UserParameter=executeBashCommand[*], /etc/zabbix/zabbix_agent2.d/testUserParametr.conf.d/executeBashCommand.sh "$1" "$2" "$3" "$4" "$5" "$6" "$7" "$8" "$9"
UserParameter=pythonTestScript[*], python3 /etc/zabbix/zabbix_agent2.d/testUserParametr.conf.d/pythonTestScript.py "$1" "$2" "$3" "$4" "$5" "$6" "$7" "$8" "$9"
UserParameter=createZabbixItems[*], python3 /etc/zabbix/zabbix_agent2.d/testUserParametr.conf.d/createZabbixItems.py "$1" "$2" "$3" "$4" "$5" "$6" "$7" "$8" "$9"
```

Рисунок 26 – Файл с указанием пользовательских параметров

Все ключи записаны в файле `testUserParametr.conf`, а исполняемые файлы в папке `testUserParametr.conf.d` и имеют такое же имя, как и имя параметра. Итоговая структура пользовательских параметров строится в папке `files` в роли zabbix-агента на zabbix-сервере (Рисунок 27).

```
rlesh@zabbix-server:/etc/ansible/zabbix-agent/files$ tree
├── zabbix_agent2.d
│   ├── testUserParametr.conf
│   └── testUserParametr.conf.d
│       ├── createZabbixItems.py
│       ├── executeBashCommand.sh
│       └── pythonTestScript.py
```

Рисунок 27 – Структура пользовательских параметров

Параметр `executeBashCommand`, если первый аргумент равен “`bash`”, то второй аргумент будет исполнен как `bash`-команда, а если первый аргумент равен “`print`”, то в вывод просто печатаются все указанные аргументы (Рисунок 28).

```
#!/bin/bash
if [ $1 == "bash" ]
then
    echo "$($2)"
elif [ $1 == "print" ]
then
    for parameter in "$@"
    do
        echo "$parameter"
    done
fi
exit 0
```

Рисунок 28 – Исполняемый код параметра `executeBashCommand`

Параметр `pythonTestScript`, имеет такой же функционал, как и предыдущий параметр, с добавлением функционала случайной суммы чисел: если первый аргумент равен `"randomSum"`, то находится сумма n -случайных чисел от 0 до n , где n – третий аргумент. Вторым отличием является то, что исполняемый файл написан на python (Рисунок 29).

```
import sys
import os
import random

match sys.argv[1]:
    case "bash":
        os.system("echo \"$((" + sys.argv[2] + "))\"")
    case "print":
        for i in range(1, len(sys.argv)):
            print(sys.argv[i])
    case "randomSum":
        sum = 0
        for i in range(int(sys.argv[2])):
            sum += random.randint(0, int(sys.argv[2]))
        print(sum)
    case _:
        print("unknown request")
```

Рисунок 29 – Исполняемый код параметра `pythonTestScript`

Параметр `createZabbixItems`, используется для тестирования обнаружения итемов: создается список, в который добавляются словари, в которых ключи являются шаблонами, а значения – передаваемыми аргументами. Количество строк в списке равно аргументу, а значения – порядковому номеру строки. Затем список конвертируется в формат `json`, и он выводится в ответ (Рисунок 30).

```
import sys
import json

n=int(sys.argv[1])
arrayItems=[]
for i in range(n):
    arrayItems.append({"#ITEMNAME":i,"#ITEMTYPE":i})
print(json.dumps(arrayItems))
```

Рисунок 30 – Исполняемый код параметра `createZabbixItems`

Далее должен быть решен вопрос с доставкой файлов пользовательских параметров на хосты. Для этого в Ansible добавляется еще одна задача для копирования файлов с использование модуля `ansible.builtin.copy`: копируется вся папка пользовательских параметров в папку `zabbix-агента` на хостах (Рисунок 31).

```
- name: Copy zabbix usersParameters
  ansible.builtin.copy:
    src: zabbix_agent2.d/
    dest: /etc/zabbix/zabbix_agent2.d/
```

Рисунок 31 – Задача для копирования файлов пользовательских параметров

Для добавления итема в шаблон, достаточно нажать в “Data collection/Templates/items” create item, в нем указывается имя итема, его тип, ключ параметра и его аргументы, тип возвращаемой информации и расписание вызова.

Добавлены три итема: Testing bashScript, Testing MacrosValue и Testing pythonScript (Рисунок 32). В первом ожидается получить сетевые интерфейсы, во втором ожидается получить случайную сумму, аргументом которой является число, записанное в макрос, третье ожидает получить в ответ отправленные аргументы.

	Name	Triggers	Key	Interval	History
<input type="checkbox"/>	Testing bashScript		executeBashCommand["bash","ip -br a"]	1m	90d
<input type="checkbox"/>	Testing MacrosValue		pythonTestScript["randomSum",{\$VALUE1}]	30s	90d
<input type="checkbox"/>	Testing pythonScript		pythonTestScript["print",1,2,3,4,5,6,7,8]	1m	90d

Item

Tags

Preprocessing

* Name

Testing MacrosValue

Type

Zabbix agent

* Key

pythonTestScript["randomSum",{\$VALUE1}]

Select

Type of information

Numeric (unsigned)

Units

* Update interval

30s

Custom intervals

Type	Interval	Period	Action
Flexible	Scheduling	50s	1-7,00:00-24:00

Add

* History storage period

Do not keep history

Storage period

90d

* Trend storage period

Do not keep trends

Storage period

365d

Value mapping

type here to search

Select

Populates host inventory field

-None-

Description

Enabled

☒

Update

Clone

Test

Delete

Cancel

All templates / ubuntuServerTemplate

Items 3

Triggers

Graphs 1

Dashboards

Discovery rules 1

Web scenarios

Templates

Tags

Macros 1

Value mapping

Template macros

Inherited and template macros

Macro

Value

{\$VALUE1}

100

T

Add

Update

Clone

Full clone

Delete

Delete and clear

Cancel

Рисунок 32 – Добавление собственных итемов в Zabbix

Также создано “Discovery rule”, которое отправляет запрос к хосту, а ответ идет на анализ в “item prototypes”, в котором указывается шаблон итемов для автоматизированного создания.

Создано “Discovery rule” с именем generate_ints, которое запрашивает у хоста параметр “createZabbixItems[10]”, далее в “item prototypes”(Рисунок 33) указывается шаблон: сгенерированные итемы будут иметь имена в виде “item {#ITEMNAME}”, где значение в скобках берется из полученного json-ответа из “Discovery rule”, а значения этих итемов будут браться от параметра “pythonTestscript”, в аргументы которого передано значение “{#ITEMTYPE}” из того же json-ответа.

The screenshot displays two panels in the Zabbix web interface. The left panel, titled "Discovery rules", shows a rule named "generate_ints" of type "Zabbix agent". Its key is "createZabbixItems[10]" and its host interface is "10.0.2.101:10050". The update interval is set to "1m". The right panel, titled "Item prototypes", shows a prototype named "item {#ITEMNAME}" of type "Zabbix agent". Its key is "pythonTestScript('randomSum',{#ITEMTYPE})". The update interval is "1m". Both panels have buttons for "Update", "Clone", "Execute now", "Test", "Delete", and "Cancel".

Рисунок 33 – Генерация Zabbix-итемов через пользовательские параметры

Так как “Discovery rule” прикреплено к шаблону, то все хосты, прикрепленные к этому шаблону, получили новые сгенерированные итемы (Рисунок 34).

The screenshot shows the "Hosts" page in Zabbix. On the left, a list of hosts is shown, including "ubuntuServer101", "ubuntuServer102", "ubuntuServer103", "ubuntuServer104", "ubuntuServer105", "ubuntuServer106", "ubuntuServer107", "ubuntuServer108", "ubuntuServer109", "ubuntuServer110", "ubuntuServer111", "ubuntuServer112", "ubuntuServer113", "ubuntuServer114", "ubuntuServer115", "ubuntuServer116", "ubuntuServer117", "ubuntuServer118", "ubuntuServer119", "ubuntuServer120", "ubuntuServer121", "ubuntuServer122", "ubuntuServer123", "ubuntuServer124", "ubuntuServer125", "ubuntuServer126", "ubuntuServer127", "ubuntuServer128", "ubuntuServer129", "ubuntuServer130", "ubuntuServer131", "ubuntuServer132", "ubuntuServer133", "ubuntuServer134", "ubuntuServer135", "ubuntuServer136", "ubuntuServer137", "ubuntuServer138", "ubuntuServer139", "ubuntuServer140", "ubuntuServer141", "ubuntuServer142", "ubuntuServer143", "ubuntuServer144", "ubuntuServer145", "ubuntuServer146", "ubuntuServer147", "ubuntuServer148", "ubuntuServer149", "ubuntuServer150", "ubuntuServer151", "ubuntuServer152", "ubuntuServer153", "ubuntuServer154", "ubuntuServer155", "ubuntuServer156", "ubuntuServer157", "ubuntuServer158", "ubuntuServer159", "ubuntuServer160", "ubuntuServer161", "ubuntuServer162", "ubuntuServer163", "ubuntuServer164", "ubuntuServer165", "ubuntuServer166", "ubuntuServer167", "ubuntuServer168", "ubuntuServer169", "ubuntuServer170", "ubuntuServer171", "ubuntuServer172", "ubuntuServer173", "ubuntuServer174", "ubuntuServer175", "ubuntuServer176", "ubuntuServer177", "ubuntuServer178", "ubuntuServer179", "ubuntuServer180", "ubuntuServer181", "ubuntuServer182", "ubuntuServer183", "ubuntuServer184", "ubuntuServer185", "ubuntuServer186", "ubuntuServer187", "ubuntuServer188", "ubuntuServer189", "ubuntuServer190", "ubuntuServer191", "ubuntuServer192", "ubuntuServer193", "ubuntuServer194", "ubuntuServer195", "ubuntuServer196", "ubuntuServer197", "ubuntuServer198", "ubuntuServer199", "ubuntuServer200". On the right, a table shows the generated item prototypes for each host. The table has columns: "Name", "Type", "Key", "Value", "Update interval", "Storage period", "Trend storage period", "Value mapping", "Description".

Name	Type	Key	Value	Update interval	Storage period	Trend storage period	Value mapping	Description
generate_ints_item 0	Zabbix agent	pythonTestScript('randomSum',{#ITEMTYPE})		1m	90d	365d		
generate_ints_item 1	Zabbix agent	pythonTestScript('randomSum',{#ITEMTYPE})		1m	90d	365d		
generate_ints_item 2	Zabbix agent	pythonTestScript('randomSum',{#ITEMTYPE})		1m	90d	365d		
generate_ints_item 3	Zabbix agent	pythonTestScript('randomSum',{#ITEMTYPE})		1m	90d	365d		
generate_ints_item 4	Zabbix agent	pythonTestScript('randomSum',{#ITEMTYPE})		1m	90d	365d		
generate_ints_item 5	Zabbix agent	pythonTestScript('randomSum',{#ITEMTYPE})		1m	90d	365d		
generate_ints_item 6	Zabbix agent	pythonTestScript('randomSum',{#ITEMTYPE})		1m	90d	365d		
generate_ints_item 7	Zabbix agent	pythonTestScript('randomSum',{#ITEMTYPE})		1m	90d	365d		
generate_ints_item 8	Zabbix agent	pythonTestScript('randomSum',{#ITEMTYPE})		1m	90d	365d		
generate_ints_item 9	Zabbix agent	pythonTestScript('randomSum',{#ITEMTYPE})		1m	90d	365d		

Рисунок 34 – Работа сгенерированных итемов

3 РАЗРАБОТКА TELEGRAM-БОТА

3.1 ОПОВЕЩЕНИЯ ZABBIX ЧЕРЕЗ TELEGRAM-БОТ

Для того, чтобы оповещения приходили через Telegram-бота необходимо на Zabbix-сервере в меню “Alerts/Media types” настроить и активировать уже предустановленный media type – Telegram. Вся настройка заключается в копировании токена своего Telegram-бота в поле “Token” в параметрах media type/Telegram (Рисунок 35).

The screenshot shows the 'Media types' configuration page in Zabbix. The 'Name' field is set to 'Telegram'. The 'Type' is set to 'Webhook'. The 'Parameters' table is as follows:

Name	Value	Action
Message	{ALERT.MESSAGE}	Remove
ParseMode		Remove
Subject	{ALERT.SUBJECT}	Remove
To	{ALERT.SENDTO}	Remove
Token	6290618300:AAF0Owm-wiLwPgXs	Remove

Below the table is an 'Add' button. At the bottom, the 'Script' field contains 'var Telegram = (...)' and the 'Timeout' is set to '10s'.

Рисунок 35 – Настройка оповещений через Telegram

Далее необходимо указать у пользователей их идентификатор в Telegram. Он указывается в меню “Users/Users” у каждого пользователя отдельно во вкладке “Media”. Вместе с методом оповещения также можно указать оповещения о событиях какого уровня важности будут приходить уведомления, и в какой промежуток времени (Рисунок 36).

The screenshot shows the 'Users' configuration page in Zabbix, specifically the 'Media' tab for a user. The table lists the media type configuration:

Media	Type	Send to	When active	Use if severity	Status	Action
	Telegram	550150925	1-7,00:00-24:00	N I W A I H D	Enabled	Edit Remove

Below the table are buttons for 'Update', 'Delete', and 'Cancel'.

Рисунок 36 – Указание ID-пользователя Telegram

Так как оповещения происходят из-за срабатывания триггеров, то основные настройки отправки происходят в действиях на триггеры. Для добавления действий по триггеру необходимо во вкладке “Alerts/Actions/Trigger actions” создать новое действие нажатием “Create action” или клонированием уже готового. В действии по триггерам

указывается имя действия, условия срабатывания и операции при срабатывании триггера и изменении его состояния.

Созданы три действия по триггерам: первое срабатывает только на триггеры с важностью равной нулю и единице и отправляет оповещения пользователям в группе “Admins level 1”, второе срабатывает на важность равную двум и трем и отправляет оповещения пользователям в группе “Admins level 2”, третья срабатывает на важность четыре и пять, а отправляет сообщения пользователям группы “Admins level 3” (Рисунок 37).

<input type="checkbox"/> Name ▲	Conditions	Operations
<input type="checkbox"/> Report problems to admins level 1	Trigger severity is less than or equals <i>Information</i>	Send message to user groups: Admins level 1 via Telegram
<input type="checkbox"/> Report problems to admins level 2	Trigger severity equals <i>Average</i> Trigger severity equals <i>Warning</i>	Send message to user groups: Admins level 2 via Telegram
<input type="checkbox"/> Report problems to admins level 3	Trigger severity is greater than or equals <i>High</i>	Send message to user groups: Admins level 3 via Telegram

Рисунок 37 – Настроенные действия для отправки оповещений

Для тестирования были написаны три триггера с указанием разной важности. Первый триггер проверяет наличие файла на хосте, и в случае его отсутствия срабатывает триггер с важностью равной одному, второй триггер срабатывает в случае, если у хоста закрыт восьмидесятый порт – срабатывает триггер с важностью равной двум, третий триггер проверяет доступ к хосту по SSH, если доступ отсутствует, то вызывается событие с важностью равной четырем (Рисунок 38).

<input type="checkbox"/> Severity	Name ▲	Operational data	Expression
<input type="checkbox"/> Information	File /etc/zabbix/secret.txt is not exist		<code>last(/Checks/vfs.file.exists{/etc/zabbix/secret.txt})=0</code>
<input type="checkbox"/> Warning	Nginx is stopped		<code>last(/Checks/net.tcp.service{http,"localhost","80"})=0</code>
<input type="checkbox"/> High	SSH is not available		<code>last(/Checks/net.tcp.service{tcp,"localhost","22"})=0</code>

Рисунок 38 – Триггеры, созданные для проверки оповещений

Для проверки был остановлен сервис sshd, который отвечает за ssh, и от Telegram-бота пришло уведомление об обнаруженной проблеме (Рисунок 39).

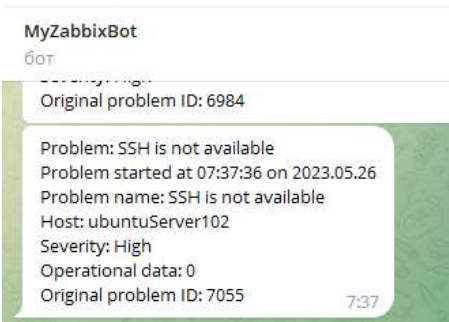


Рисунок 39 – Оповещение, пришедшее от Telegram-бота

3.2 РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

В качестве пользовательского меню в Telegram-чате выступает текстовое сообщение с прикрепленным модулем клавиатуры.

Использование telegram-бота начинается с нажатия кнопки “Старт” в чате бота, при нажатии которой в чат боту отправляется сообщение с текстом “/start”. После этого должно появиться стартовое меню.

В стартовом меню нужно будет указать данные пользователя: id, логин, пароль и сервер Zabbix, это меню получило название “userDataMenu”. После ввода данных или нажатия кнопки “Главное меню”, должен произойти переход в главное меню – “mainMenu”. Это связующее меню, от которого идут переходы в другие части интерфейса. В этом меню предлагается ввести команду с указанием ID события, чтобы получить информацию о событии. После ввода команды или выбора события из списка в problemGetMenu откроется меню с информацией о событии “eventGetMenu”, и также указаны переходы к меню применения скриптов “scriptGetscriptsbyevents” и обновлений (действий) “eventAcknowledgeMenu”. В действиях можно выбрать какие действия будут применены к событию и, следовательно, какие параметры будут запрошены. После выбора действий идет меню, в котором выбираются параметры для действия: если было выбрано действие с обновлением важности – будет предложено выбрать новую важность, если выбрано действие с отправкой сообщения, то будет предложено ввести его в строку. При выборе пункта “Скрипты” под событием откроется меню выбора скриптов, после нажатия на имя скрипта он исполнится, а его вывод будет выведен в следующем меню “scriptExecuteMenu”(Рисунок 40).

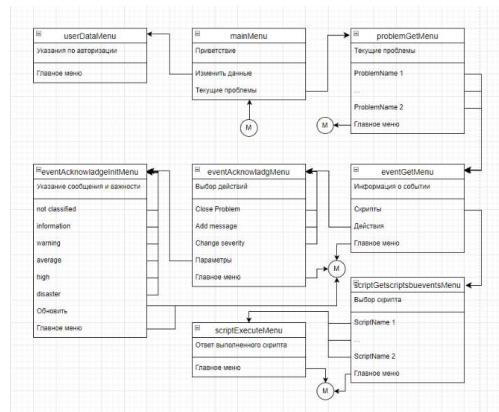


Рисунок 40 – Макеты реализованных меню

Для реализации меню на языке программирования Go создана структура “menu” с двумя полями: “text” – текст и “keyboard” – клавиатура. В поле текста указывается дополнительная информация, в поле клавиатуры указывается массив массивов кнопок. В поле “CallbackData” кнопок указывается метод и параметры, которые необходимо передать в Zabbix API.

Меню userDataMenu и mainMenu не зависят от полученных данных, поэтому реализованы в виде переменных типа “menu”, а остальные меню реализованы в виде функций, который получают на вход параметры, необходимые для построения меню, строят меню и возвращают в виде структуры типа menu (Рисунок 41).

```

type menu struct {
    keyboard [][]InlineKeyboardButton
    text string
}

func eventInfoMenu(events EventGetResponse) menu {
    event := events.Result[0]
    tmStr := strUnixTimeToStrDate(event.Clock)
    menu := menu{
        text: "Информация о событии " + event.Eventid + "\n" +
            "Имя события: " + event.Name + "\n" +
            "Время обнаружения: " + tmStr + "\n" +
            "Серьезность: " + event.Severity + "\n",
    }
    if len(event.Acknowledges) > 0 {
        menu.text += "Обновления события:\n"
        for _, acknowledge := range event.Acknowledges {
            menu.text += "---:\n" +
                "Время: " + strUnixTimeToStrDate(acknowledge.Clock) + "\n" +
                "От: " + acknowledge.Username + "\n" +
                "Сообщение:\n" + acknowledge.Message + "\n"
        }
        menu.keyboard = append(
            menu.keyboard,
            []InlineKeyboardButton{
                InlineKeyboardButton{Text: "Скрипты",
                    CallbackData: "/scripts " + event.Eventid},
                InlineKeyboardButton{Text: "Действия",
                    CallbackData: "/event.acknowledge " + event.Eventid},
                InlineKeyboardButton{Text: "Главное меню",
                    CallbackData: "/mainMenu"},
            },
        )
    }
    return menu
}

```

Рисунок 41 – Реализация пользовательского интерфейса

3.3 СИСТЕМА ДЕЙСТВИЙ ПРИ НАЖАТИИ КНОПОК

При нажатии кнопок сервер получает “Update”, в котором через поле “callback_query” передан текст, указанный за кнопкой. В этом тексте прописаны команды, на которые сервер должен реагировать.

Команды в написанных меню делятся на два типа: вызов меню, не зависящего от пользовательских данных, и вызов меню, зависящих от пользовательских данных.

Для первой группы меню достаточно сравнить команду и имена этих меню, и передать меню через post-запрос.

У второй группы меню в команде содержатся название методом и пользовательские параметры, например ID события, или текст пользовательского сообщения, знать которые серверная часть не может. Так что есть два варианта – это передавать кнопкой структуру, содержащую пары ключа и значения, или придерживаться шаблона команды и указывать все параметры в определенном порядке. От первого варианта пришлось отказаться, так как поле “callback_data” кнопки может содержать только 64 байта, чего не хватает для некоторых ресурсов. Поэтому принято решение придерживаться шаблона команд, указывая все параметры через пробел, где единственным идентификатором передающегося параметра будет являться его порядковый номер в запросе.

Также в меню с выбором действий для обновления реализован выбор нескольких опций. Реализован множественный выбор через битовые маски: каждой кнопке присвоено число равное степени двойки, при этом все числа разные, и в меню передается число, изначально равное нулю. И когда пользователь нажимает на кнопку происходит побитовая операция «исключающая или» между передаваемым числом и числом, закрепленным за кнопкой, а после в это же меню передается уже полученное число. В итоге при повторном нажатии на ту же кнопку снова происходит побитовое «исключающая или» и бит, отвечающий за эту кнопку снова стирается. Так же

в макет меню добавлены проверки получаемого числа, чтобы пользователь мог отслеживать уже выбранные пункты (Рисунок 42).

```
func eventAcknowledgeMenu(eventId string, actions int) menu{
    menu := menu{
        text: "Выберите действие для " + eventId + "\n" +
            "После выбора нажмите \"Параметры\"",
    }
    var button1 InlineKeyboardButton = InlineKeyboardButton{Text: "Close problem: No",
        CallbackData: "/event.acknowledge " + eventId + " " + strconv.Itoa(1 ^ actions)}
    var button2 InlineKeyboardButton = InlineKeyboardButton{Text: "Add message: No",
        CallbackData: "/event.acknowledge " + eventId + " " + strconv.Itoa(4 ^ actions)}
    var button3 InlineKeyboardButton = InlineKeyboardButton{Text: "Change severity: No",
        CallbackData: "/event.acknowledge " + eventId + " " + strconv.Itoa(8 ^ actions)}
    if (1 ^ actions) < actions {
        button1.Text = "Close problem: Yes"
    }
    if (4 ^ actions) < actions {
        button2.Text = "Add message: Yes"
    }
    if (8 ^ actions) < actions {
        button3.Text = "Change severity: Yes"
    }
    menu.Keyboard = append(
        menu.Keyboard,
        []InlineKeyboardButton{
            button1,
            button2,
            button3,
        },
    )
}
```

Рисунок 42 – Реализация множественного выбора в меню Telegram-бота

3.4 НАПИСАНИЕ ЗАПРОСОВ ДЛЯ TELEGRAM BOT API

Для получения обновления от Telegram-бота написана функция “getUpdates” (Рисунок 43), в аргументах которой указываются url бота и параметр “offset”. В методе отправляется get-запрос по url, с открытым параметром “offset”. Параметр “offset”, позволяет получать не все обновления на сервере, а только те, чей “update_id” выше параметра значения “offset”.

```
func getUpdates(botUrl string, offset int) ([]Update, error){
    resp, err := http.Get(botUrl + "/getUpdates" + "?offset=" + strconv.Itoa(offset))
    if err != nil: nil, err }
    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)
    if err != nil: nil, err }
    var response Response
    if err := json.Unmarshal(body, &response); err != nil: nil, err }
    return response.Result, nil
}
```

Рисунок 43 – Функция для получения обновлений от Telegram-бота

Далее для каждого обновления вызываются функции “respondChat”, “editMessage”. Первый отвечает на новые сообщения, используя метод “sendMessage” (Рисунок 44).

```

func respondChat(botUrl string, update Update) error {
    if update.Message.Chat.Id == 0: nil
    var botMessage BotMessage
    botMessage.ChatId = update.Message.Chat.Id
    command := strings.Split(update.Message.Text, " ")
    botMessage.Text = "Главное меню"
    botMessage.ReplyMarkup.InlineKeyboard = mainMenu.keyboard
    switch command[0]{...}
    buf, err := json.Marshal(botMessage)
    if err != nil: err
    _, err = http.Post(botUrl + "/sendMessage", "application/json", bytes.NewBuffer(buf))
    if err != nil: err
    return nil
}

func editMessage(botUrl string, update Update) error {
    if update.CallbackQuery.Message.Chat.Id == 0: nil
    var answerQuery AnswerCallbackQuery
    answerQuery.CallbackQueryId = update.CallbackQuery.Id
    answerQuery.Text = ""
    var editMessage EditMessageText
    editMessage.ChatId = update.CallbackQuery.Message.Chat.Id
    editMessage.MessageId = update.CallbackQuery.Message.MessageId
    editMessage.Text = mainMenu.text
    editMessage.ReplyMarkup.InlineKeyboard = mainMenu.keyboard
    command := strings.Split(update.CallbackQuery.Data, " ")
    switch command[0]{...}
    buf, err := json.Marshal(editMessage)
    if err != nil: err
    _, err = http.Post(botUrl + "/editMessageText", "application/json", bytes.NewBuffer(buf))
    if err != nil: err
    buf2, err := json.Marshal(answerQuery)
    if err != nil: err
    _, err = http.Post(botUrl + "/answerCallbackQuery", "application/json", bytes.NewBuffer(buf2))
    return nil
}

```

Рисунок 44 – Функции для отправки и изменения сообщений Telegram-бота

В качестве хэндлера выступает первый элемент полученного сообщения, элементы — это массив, полученный из текста, полученного сообщения, разделенный по пробелам. Первый элемент проходит проверку на соответствие команде (Рисунок 45), и если соответствие есть, то начинается дальнейшая проверка параметров и попытка выполнить код.

```

command := strings.Split(update.CallbackQuery.Data, " ")
switch command[0] {
case "/mainMenu":
    editMessage.Text = mainMenu.text
    editMessage.ReplyMarkup.InlineKeyboard = mainMenu.keyboard
case "/requestDataMenu":
    editMessage.Text = userDataMenu.text
    if fileExist, err := checkUserData(update.CallbackQuery.FromId); err != nil {
        log.Println("Error in func checkUserData(): ", err)
    } else if fileExist {
        userData, err := getUserData(update.CallbackQuery.FromId)
        if err != nil {
            log.Println("Error in func getUserData(): ", err)
        }
        editMessage.Text = editMessage.Text + userData.String()
    }
    editMessage.ReplyMarkup.InlineKeyboard = userDataMenu.keyboard
case "/eventAcknowledge":
    if len(command) == 2 {
        out := eventAcknowledgeMenu(command[1], actions[0])
        editMessage.Text = out.text
        editMessage.ReplyMarkup.InlineKeyboard = out.keyboard
    }
    if len(command) == 3 {
        actions := strconv.Atoi(command[2])
        out := eventAcknowledgeMenu(command[1], actions)
        editMessage.Text = out.text
        editMessage.ReplyMarkup.InlineKeyboard = out.keyboard
    }
case "/eventAcknowledgeInit":
    if len(command) == 3 {

```

Рисунок 45 – Проверка первого параметра на соответствие команде Telegram-бота

Для отправки запроса используются методы из встроенной библиотеки “net/http”: `http.Post()` для получения токена авторизации, и `http.NewRequest()` для запросов, в которых необходимо указать дополнительный заголовок для

авторизации. Для авторизации в Zabbix ID, логин, пароль и адрес сервера берутся из ранее введенных данных. Для хранения пользовательских данных реализованы функции: проверки введенных данных, проверки существования директории для хранения, создания директории для хранения, создания записи о новом пользователе, изменения записи пользователя (Рисунок 46).

```
type UserData struct {
    Id int           json:"id"
    Login string     json:"login"
    Password string  json:"password"
    Server string    json:"server"
}

func checkCacheDir() error{...}

func checkUserData(userId int) (bool,error){...}

func getUserData(userId int) (UserData,error) {...}

func createUserData(userId int, data UserData) error{...}

func strToUserData(str string) (UserData,error){...}

func (UserData UserData) String() string{...}
```

Рисунок 46 – Функции для хранения введенных значений

3.5 НАПИСАНИЕ ЗАПРОСОВ ДЛЯ ZABBIX API

В Zabbix API есть авторизация через “AuthToken” – токен авторизация, для его получения необходимо отправить запрос с методом авторизации на Zabbix-сервер. Метод авторизации называется “user.login” (Рисунок 47), а в качестве параметров в него передаются логин и пароль. В ответ на этот запрос Zabbix-сервер, в случае успешной авторизации возвращает “AuthToken” в поле “result”.

```
func getAuthToken(id int, username string, password string, server string) (string,error) {
    userLoginRequest := newUserLoginRequest(id, username, password)
    jsonBody, _ := json.Marshal(userLoginRequest)
    buf, err := json.Marshal(userLoginRequest)
    if err != nil {
        log.Println("Error in func json.Marshal()", err)
        return userLoginResponse(), err
    }
    resp, err := http.Post(zabbixURL, "application/json", bytes.NewReader(buf))
    if err != nil {
        log.Println("Error in func http.Post()", err)
        return userLoginResponse(), err
    }
    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        log.Println("Error in func ioutil.ReadAll()", err)
        return userLoginResponse(), err
    }
    var response userLoginResponse
    if err := json.Unmarshal(body, &response); err != nil {
        log.Println("Error in func json.Unmarshal()", err)
        return userLoginResponse(), err
    }
    if response == userLoginResponse{} {
        return userLoginResponse(), errors.New("wrong userData")
    }
    return response, nil
}

type UserLoginRequest struct {
    Id int           json:"id"
    Username string  json:"username"
    Password string  json:"password"
}

func newUserLoginRequest(id int, username string, password string) UserLoginRequest {
    ans := UserLoginRequest{}
    ans.Id = id
    ans.Username = username
    ans.Password = password
    return ans
}

type UserLoginResponse struct {
    Id int           json:"id"
    Result string    json:"result"
}
```

Рисунок 47 – Реализация получения токена авторизации от Zabbix-сервера

В работе использованы такие методы Zabbix, как “problem.get”, “event.get”, “event.acknowledge”, “script.getscriptsbyevents” и “script.execute”. Для каждого из этих методов написана функция с соответственным именем, кроме функций также созданы структуры для отправки запросов и получения

ответов. Во всех функциях указан http-метод POST, а к заголовкам добавлена запись "Authorization: Bearer <authToken>" для авторизации.

Все написанные функции имеют схожий алгоритм, но разное количество входящих параметров, и также в них используются разные структуры для отправки запросов и получения ответов. В функции подаются параметры для формирования запроса к Zabbix API, функция собирает запрос, отправляет его, и записывает ответ от Zabbix в структуру, затем полученная структура передается в качестве ответа функции (Рисунок 48).

```
func ProblemGetResponse(id int) (EventGetResponse, error) {
    userData, err := getUserData(id)
    if err != nil {
        log.Println("Error in func getUserData()")
    }
    authToken, err := getAuthToken(userData.Id, userData.Login, userData.Password, userData.Server)
    if err != nil {
        return EventGetResponse{}, err
    }
    zabbixURL := "http://" + userData.Server + "/api_jsonrpc.php"

    eventGetRequest := newProblemGetRequest(userData.Id)
    buf, err := json.Marshal(eventGetRequest)
    if err != nil {
        log.Println("Error in func json.Marshal(): ", err)
        return EventGetResponse{}, err
    }

    client := &http.Client{}
    req, err := http.NewRequest("POST", zabbixURL, bytes.NewBuffer(buf))
    if err != nil {
        log.Println("Error in func http.NewRequest(): ", err)
        return EventGetResponse{}, err
    }
    req.Header.Set("Content-Type", "application/json")
    req.Header.Add("Authorization", "Bearer " + authToken.Result)
    resp, err := client.Do(req)
    if err != nil {
        log.Println("Error in func client.Do(): ", err)
        return EventGetResponse{}, err
    }
}

type EventGetResponse struct {
    jsonrpc string      `json:"jsonrpc"`
    Result []struct {
        Eventid string      `json:"eventid"`
        Name string      `json:"name"`
        Severity string      `json:"severity"`
        Clock string      `json:"clock"`
        Acknowledges []struct {
            Clock string      `json:"clock"`
            Username string      `json:"username"`
            Message string      `json:"message"`
        }
    }
    Id int      `json:"id"`
}

type ProblemGetRequest struct {
    jsonrpc string      `json:"jsonrpc"`
    Method string      `json:"method"`
    Id int      `json:"id"`
    Params struct {
        Id int      `json:"id"`
    }
}

func newProblemGetRequest(id int) ProblemGetRequest {
    ans := ProblemGetRequest{}
    ans.Jsonrpc = "2.0"
    ans.Method = "problem.get"
    ans.Id = id
    return ans
}
```

Рисунок 48 – Реализация функции для получения списка проблем

Итоговая структура (Рисунок 49) проекта содержит файл “main.go”, в котором происходит обработка полученных данных от Telegram Bot API, и отправка новых запросов к Telegram Bot API, файл “menus.go” содержит функции для формирования пользовательского интерфейса, в файле “userData.go” заданы инструкции для хранения пользовательских данных, файл “telegramModels.go” содержит структуры для запросов и ответов Telegram Bot API, файлы “ZabbixAPI.go” и “ZabbixModels.go” реализуют от отправку запросов и получение ответов от Zabbix API. В файле “config.env” – указан токен Telegram-бота.

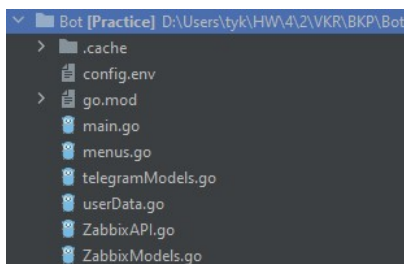


Рисунок 49 – Структура проекта

3.6 ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО TELEGRAM-БОТА

Основное тестирование проводилось параллельно этапу разработки, чтобы искать недочет в коде сразу после написания определенной логической части кода.

Финальное тестирование заключалось в использовании пользовательского меню в Telegram-боте: после ввода команды “/start”, введены пользовательские данные Zabbix-пользователя в указанном формате. Для тестирования вызваны срабатывания триггеров разного уровня важности, проверены разные комбинации возможных введенных действий, включая закрытие проблемы, отправку сообщения и изменения важности события.

Для теста была выбрана одна из текущих проблем – “File for UserParameters are not exist”, для нее были выбраны действия “Add Message” и “Change Severity” (Рисунок 50).



Рисунок 50– Меню обновления события

После этого была выбрана новая важность для события и написан комментарий. После отправки комментария от системы мониторинга Zabbix пришло оповещение об обновлении события (Рисунок 51).

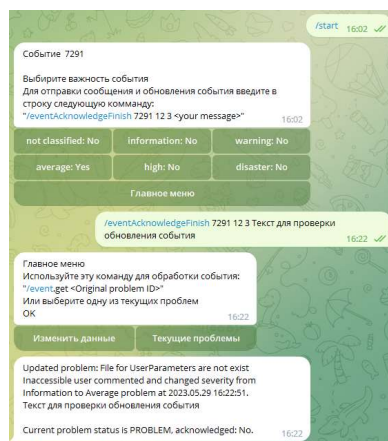


Рисунок 51 – Результат обновления события

Далее проверено выполнение скриптов: на Zabbix-сервере созданы скрипты, для перезапуска сервисов Nginx, Zabbix-agent2 и восстановления файлов пользовательских параметров Zabbix. Скрипты запускают указанный плейбук Ansible на сервере, с указанием хоста, на котором для выполнения задач плейбука (Рисунок 52).

<input type="checkbox"/> Restart Nginx	Manual event action	Script	Server	sudo -u zabbix ansible-playbook /etc/ansible/ubuntuservers/restartNginx.yml -i {HOST.NAME},
<input type="checkbox"/> Restart Zabbix Agent 2	Manual event action	Script	Server	sudo -u zabbix ansible-playbook /etc/ansible/ubuntuservers/restartZabbixAgent2.yml -i {HOST.NAME},
<input type="checkbox"/> Restore userParameters files	Manual event action	Script	Server	sudo -u zabbix ansible-playbook /etc/ansible/ubuntuservers/restoreZabbixUserParameters.yml -i {HOST.NAME},

Рисунок 52 – Подготовленные скрипты для тестирования

После выбора проблемы и нажатия кнопки “Скрипты”, выведен список скриптов, которые доступны для исполнения для выбранного события. После выбора скрипта будет показан статус выполнения скрипта и ответ скрипта (Рисунок 53).

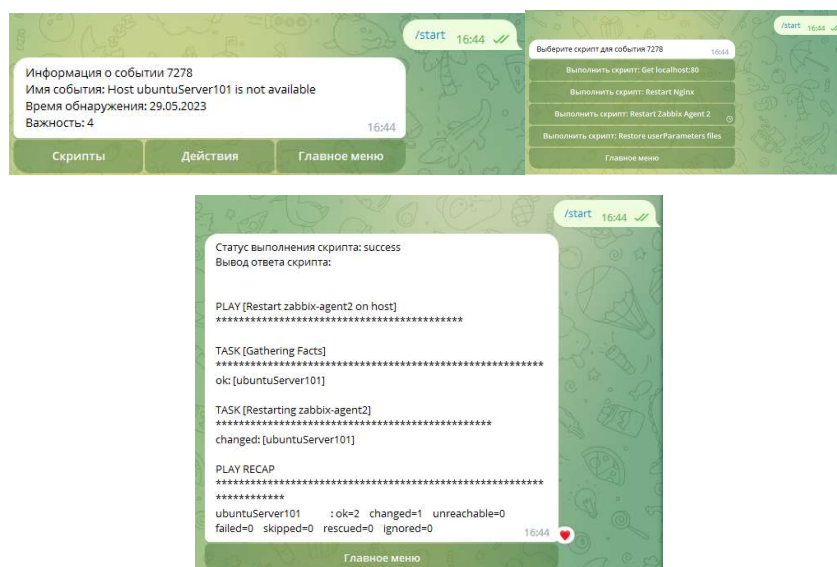


Рисунок 53 – Выполнение и вывод скрипта

ЗАКЛЮЧЕНИЕ

В ходе практики были изучены документации для Zabbix и Telegram API, развернута система мониторинга Zabbix с системой управления конфигурациями Ansible, написаны пользовательские параметры Zabbix, настроена авторегистрация хостов в Zabbix, написан плейбук Ansible для развертывания и настройки Zabbix-агента, включая доставку файлов пользовательских параметров на хосты. Настроено оповещение пользователей Zabbix через Telegram-бот, написан Telegram-бот для обновления событий в системе мониторинга Zabbix. Telegram-бот способен показать текущие проблемы, отображаемые в Zabbix, предложить пользователю обновить событие, или выполнить скрипт для этого события. При обновлении события доступен выбор от одного до трех действий: закрытие проблемы, написание комментария к обновлению и изменение важности события. В случае выполнения скрипта бот выводит ответ выбранного скрипта. Данная работа укрепила знания необходимые для поддержания инфраструктуры компании, дала опыт использования Zabbix, Ansible и Telegram Bot API, а также опыт написания веб-приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация Zabbix [Электронный ресурс] – Режим доступа: <https://www.zabbix.com/documentation/6.0/ru/manual> Дата доступа: 22.05.2023
2. Документация Ansible [Электронный ресурс] – Режим доступа: <https://docs.ansible.com/ansible/latest/index.html> Дата доступа: 22.05.2023
3. Документация Telegram Bot API [Электронный ресурс] – Режим доступа: <https://core.telegram.org/bots/> Дата доступа: 22.05.2023
4. Документация PostgreSQL [Электронный ресурс] – Режим доступа: <https://postgrespro.ru/docs/postgresql/15/index> Дата доступа 22.05.2023
5. Документация Nginx [Электронный ресурс] – Режим доступа: <https://nginx.org/ru/docs/> Дата доступа 22.05.2023