МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки / специальности)

Мобильные и сетевые технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка системы мониторинга серверов»

Обучающийся К.С. Круткин _____
(Инициалы Фамилия) (личная подпись)

Руководитель к.п.н., доцент, О.В. Оськина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к.п.н., доцент, А.В. Егорова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Бакалаврская работа выполнена на тему «Разработка системы мониторинга серверов».

Целью работы является проектирование и создание простого программного решения для мониторинга серверов компании "ВорлдИнтерТех РУС", позволяющего системным администраторам своевременно выявлять проблемы, отслеживать состояние серверов и повышать надёжность ИТ-инфраструктуры.

Приложение реализовано с использованием языка программирования Python и библиотек Paramiko и Matplotlib, обеспечивает удалённый сбор данных через SSH и их визуализацию в виде графиков.

Выпускная квалификационная работа состоит из введения, трёх глав, заключения и списка литературы, включающего 20 источников. Общий объем работы 61 страница, содержащая 23 рисунка, 4 таблицы и 4 диаграммы.

Во введении обоснована актуальность темы, определены объект и предмет исследования, сформулированы цели и задачи. Первая глава посвящена анализу предметной области, обзору существующих решений и выбору ключевых метрик мониторинга серверов. Во второй главе описаны этапы проектирования системы: выбор технологий, разработка UML-диаграмм и формирование требований к аппаратно-программному обеспечению. Третья глава посвящена реализации системы разработке функционала сбора и визуализации метрик, тестированию программы на виртуальной машине и анализу результатов её работы.

В заключении представлены основные выводы о проделанной работе, подтверждающие соответствие разработанной системы поставленным требованиям, а также указаны возможные направления её дальнейшего развития.

Abstract

The title of the graduation work is *Development of a Server Monitoring System*.

The graduation work consists of an introduction, three parts, 19 figures, 4 tables, 1 diagram, a conclusion, and a list of 20 references including foreign sources.

The aim of this graduation work is to design and implement a simple software solution for monitoring company servers at WorldInterTech RUS. The system allows system administrators to detect problems in real time, track server status, and improve the reliability of IT infrastructure.

The object of the graduation work is the process of server monitoring in corporate IT environments.

The subject of the graduation work is the development of a monitoring system using Python programming language along with Paramiko and Matplotlib libraries to collect remote data via SSH and visualize it in graphical form.

The key issue of the graduation work is to provide an efficient and user-friendly way for system administrators to monitor critical server metrics.

The graduation work may be divided into several logically connected parts which are literature review, system design, and implementation.

The first part describes in details the current state of server monitoring technologies, analyzes existing solutions, and identifies key performance indicators that should be monitored.

The second part outlines the system design phase, including technology selection, UML diagrams, and requirements for hardware and software environments.

The third part consists of the implementation of data collection and visualization modules, testing the system on a virtual machine, and evaluating its performance.

In conclusion we’d like to stress that the developed system meets the specified requirements and provides reliable tools for server monitoring. However, more experimental data and long-term testing are required to assess its scalability and performance under heavy load.

The work is of interest for a wide circle of readers interested in server monitoring systems and software development for IT infrastructure management.

Оглавление

Введение

В современном мире цифровые технологии играют ключевую роль в обеспечении бесперебойной работы информационных систем, которые являются основой функционирования бизнеса, государственных учреждений и других организаций. Особенно значимым является мониторинг серверов, так как их отказ или снижение производительности могут привести к серьезным последствиям, таким как потеря данных, сбои в работе приложений и финансовые убытки. Для своевременного выявления проблем и оптимизации работы IT-инфраструктуры необходимы надежные системы мониторинга, способные собирать, анализировать и визуализировать данные о состоянии серверов в режиме реального времени.

Цель работы.

Целью данной выпускной квалификационной работы является разработка и внедрение системы мониторинга серверов, реализованной в виде Python-приложения. Данное приложение позволяет в режиме реального времени собирать метрики производительности (загрузка CPU, использование оперативной памяти, заполнение дискового пространства) с удаленных серверов через SSH-подключение, а также визуализировать данные в виде графиков для удобства анализа.

Объект исследования.

Объектом исследования выступает процесс мониторинга состояния серверов, который включает сбор данных о ключевых метриках производительности (CPU, RAM, Disk), их обработку и визуализацию для последующего анализа.

Предмет исследования.

Предметом исследования является разработанная система мониторинга серверов, реализованная с использованием Python и библиотек paramiko для SSH-подключения и matplotlib для построения графиков. Программа предоставляет функционал для:

– сбора данных о загрузке CPU, использовании оперативной памяти и дискового пространства;

– визуализации метрик в виде графиков;

– сохранения результатов мониторинга в файл для дальнейшего анализа.

Задачи исследования.

Для достижения поставленной цели были выполнены следующие задачи:

– изучение предметной области и существующих решений для мониторинга серверов (например, Nagios, Zabbix, Prometheus);

– анализ проблем в существующих подходах к мониторингу IT-инфраструктуры;

– выбор технологий разработки (Python, Paramiko, Matplotlib) и их обоснование;

– разработка программы для мониторинга серверов, включая сбор данных через SSH и визуализацию метрик;

– тестирование разработанной программы на реальных серверах с использованием инструментов нагрузочного тестирования (stress-ng, dd);

– оценка эффективности программы на основе экспериментальных данных.

Разрабатываемое программное обеспечение направлено на автоматизацию процесса мониторинга серверов и предоставляет системным администраторам следующие функции:

– программа позволяет отслеживать ключевые параметры состояния сервера, такие как уровень загрузки процессора, объем используемой оперативной памяти и степень заполнения дискового пространства. При превышении допустимых значений система сигнализирует о возможных сбоях, что дает возможность быстро принять меры для предотвращения аварийных остановок;

– собранные данные отображаются в виде наглядных графиков, что

упрощает восприятие текущего состояния сервера и делает анализ более эффективным;

– результаты мониторинга могут сохраняться в файл для последующего анализа, составления отчетов или долгосрочного хранения информации.

Для проверки корректности работы системы и её устойчивости к нагрузкам тестирование будет проводиться на виртуальной машине под управлением Ubuntu Server 24.04 LTS. Для имитации высокой нагрузки будут использованы утилиты stress-ng и dd. Эти инструменты позволят протестировать точность измерения метрик и поведение программы при изменении состояния сервера, включая сетевые сбои и резкие скачки нагрузки.

Научная новизна

Разработка системы мониторинга серверов на основе Python и открытых библиотек позволяет создать гибкое и масштабируемое решение, которое адаптируется под конкретные потребности организации. Особая научная новизна заключается в следующем:

– адаптивность решения. Система разработана с учётом специфики компании "ВорлдИнтерТех РУС" и её ИТ – инфраструктуры, что делает её более эффективной и соответствующей реальным задачам;

– минимизация зависимости от сторонних инструментов. В отличие от существующих решений, такие как Nagios, Zabbix и Prometheus, предлагаемая система минимизирует использование внешних зависимостей, что упрощает внедрение и поддержку;

– оптимизация затрат. Разработка собственной системы позволяет снизить расходы на внедрение и поддержку готовых решений, а также избежать необходимости покупки лицензий или платных сервисов.

Практическое применение

Разработанная система имеет широкое практическое применение в различных организациях, где требуется постоянный контроль за состоянием серверов. Её преимущества включают:

– безопасность. Использование протокола SSH обеспечивает шифрование данных и защиту от несанкционированного доступа;

– благодаря выбору Python и библиотеки Paramiko, система легко масштабируется и может быть доработана под новые требования;

– надёжность. Тестирование программы на виртуальной машине с Ubuntu Server 24.04 LTS показало её стабильную работу как при низкой, так и при высокой нагрузке на сервер;

– простота использования. Графическая часть системы, созданная с помощью Matplotlib, обеспечивает наглядное представление информации, что упрощает анализ данных для системных администраторов.

Кроме того, система может быть адаптирована для работы в гибридных или облачных средах, что делает её актуальной для компаний, использующих распределённую ИТ-инфраструктуру. Возможность кастомизации отчётов и уведомлений позволяет ИТ-специалистам быстрее реагировать на инциденты и снижать простой оборудования.

Глава 1 Теоретические основы разработки систем мониторинга серверов

1.1Характеристика организации

В данном разделе рассматриваются технико-экономические характеристики IT-компании «ВорлдИнтерТех РУС», в структуре которой функционирует отдел по разработке программного обеспечения. Основным направлением деятельности компании является содействие цифровой трансформации бизнеса, включая создание облачных решений, разработку программных продуктов и предоставление услуг в сфере интернет-продвижения.

Организация «ВорлдИнтерТех РУС» ориентирована на обеспечение стабильной работы корпоративных информационных систем, а также на развитие и модернизацию IT – инфраструктуры своих клиентов. В перечень предлагаемых услуг входят комплексные решения по администрированию серверного оборудования, сопровождению сетевых систем и обслуживанию баз данных, что позволяет повысить эффективность и надёжность бизнес-процессов.

Среди основных направлений деятельности можно выделить:

– разработку программного обеспечения для автоматизации бизнес-процессов;

– внедрение систем мониторинга и управления IT-ресурсами;

– обеспечение бесперебойной работы серверов и сетей;

– оптимизацию производительности IT – систем и баз данных;

– проведение аудита информационной безопасности и технической поддержки.

Организационная структура компании, которая включает в себя несколько ключевых подразделений:

– отдел разработки программного обеспечения, который отвечает за создание и поддержку программных решений, включая системы мониторинга серверов;

– отдел технической поддержки, обеспечивающий оперативное решение проблем клиентов;

– отдел тестирования, занимающийся проверкой работоспособности и надежности разработанных решений;

– управляющий состав, координирующий работу всех подразделений и взаимодействующий с заказчиками;

Организационная структура представлена на рисунке 1.

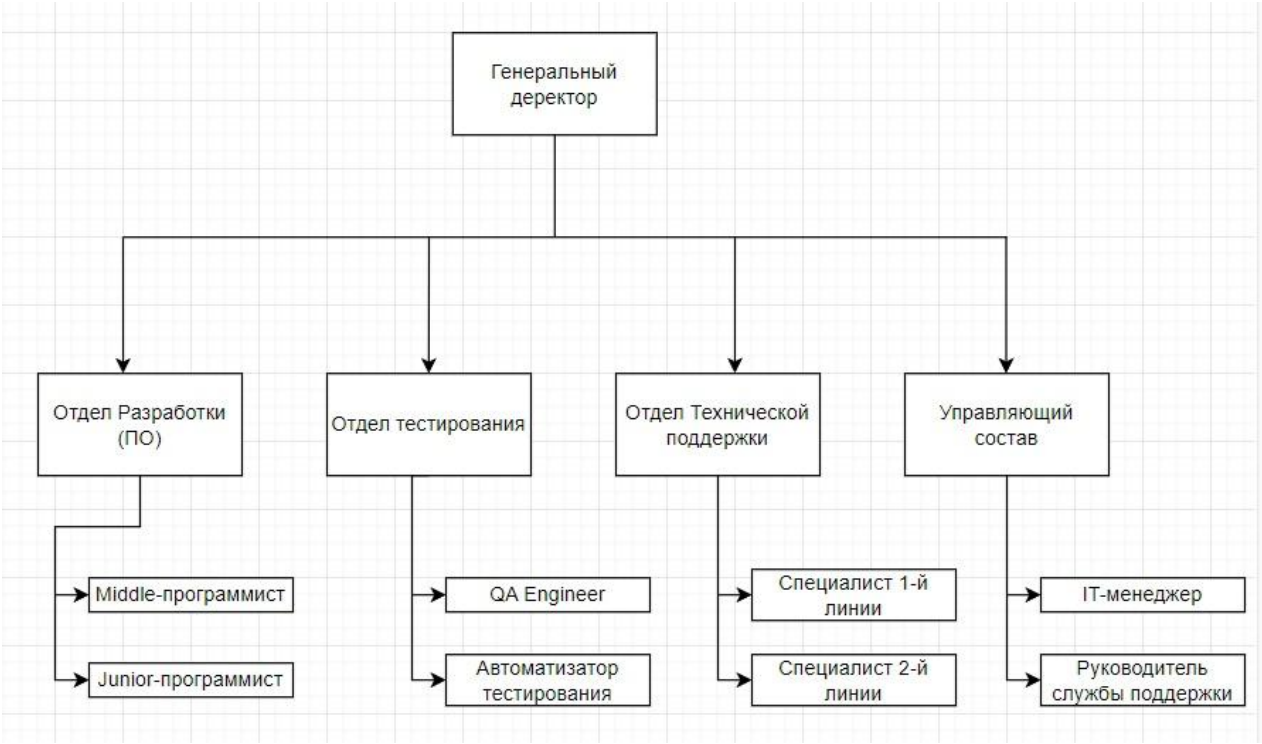

Рисунок 1 – Организационная структура

В качестве предметной области выступает система мониторинга серверов, которая представляет собой проект, реализуемый в рамках отдела разработки программного обеспечения. Этот проект направлен на создание инструмента для сбора, анализа и визуализации данных о состоянии серверов

(загрузка CPU, использование оперативной памяти, дискового пространства). Процесс разработки системы мониторинга включает в себя, Анализ требований к системе и выбор технологий, Проектирование архитектуры программы, реализацию функционала для сбора данных через SSH и их визуализации, тестирование программы на тестовых серверах с применением нагрузочных инструментов (stress – ng, dd).

Разработка системы мониторинга серверов требует тщательного планирования и четкого понимания задач, которые она должна решать. На первом этапе специалисты определяют ключевые метрики, которые необходимо отслеживать: загрузку процессора (CPU), использование оперативной памяти (RAM) и дискового пространства (Disk). Эти метрики играют ключевую роль в оценке работоспособности серверов и позволяют оперативно выявлять возможные сбои. Для реализации проекта были выбраны проверенные и современные технологии. В качестве основного языка разработки был использован Python, благодаря своей простоте, читаемости кода и широкому набору библиотек. Для организации удалённого подключения к серверам применялась библиотека Paramiko, обеспечивающая безопасное взаимодействие через протокол SSH. Для отображения собранной информации в наглядном виде была интегрирована библиотека Matplotlib, позволяющая строить графики в реальном времени.

Далее начался этап проектирования системы, на котором акцентировалось внимание на выборе подходящих инструментов и архитектурных решений. Задача заключалась не только в корректном сборе данных, но и в их удобном представлении для пользователей – в первую очередь для системных администраторов. Интерфейс и логика работы программы проектировались таким образом, чтобы система могла масштабироваться и легко интегрироваться в уже существующие IT-инфраструктуры компаний. Такой подход обеспечивает создание решения, которое будет не только технически устойчивым и функциональным, но и практичным в повседневной эксплуатации.

1.2 Обзор существующих решений для мониторинга серверов

В настоящее время существует множество готовых решений для мониторинга серверов, каждое из которых имеет свои преимущества и недостатки. Рассмотрим наиболее популярные инструменты: Nagios, Zabbix, Prometheus и их особенности.

Nagios представляет собой одну из первых и проверенных временем систем мониторинга, предназначенных для контроля состояния ИТ-инфраструктуры. Она активно применяется для наблюдения за работой серверов, сетевых компонентов, приложений и сервисов в режиме реального времени. Основным достоинством продукта считается его устойчивая работа даже в сложных условиях эксплуатации. Способность функционировать без сбоев на протяжении длительного времени делает Nagios популярным решением для организаций, где критически важно минимизировать простои.

Функции Nagios включают:

– контроль основных параметров производительности, таких как загрузка процессора, объём используемой оперативной памяти, состояние дискового пространства и сетевые показатели;

– уведомления о возникающих проблемах через электронную почту, SMS или другие доступные каналы связи;

– гибкое расширение функционала за счёт использования сторонних плагинов, что позволяет адаптировать систему под специфические задачи.

К положительным сторонам можно отнести:

– высокую отказоустойчивость и надёжность, которая достигается благодаря хорошо отлаженной архитектуре;

– масштабируемость и гибкость, позволяющую использовать решение в различных средах и подстраивать его под конкретные потребности.

Однако у Nagios есть и ряд ограничений:

– непростая настройка, которая может вызвать трудности у новых

пользователей или специалистов без опыта работы с подобными инструментами;

– отсутствие современного встроенного интерфейса, что требует дополнительных усилий по его доработке или интеграции с другими средствами представления информации.

Пользовательский интерфейс Nagios представлен на рисунке 2.

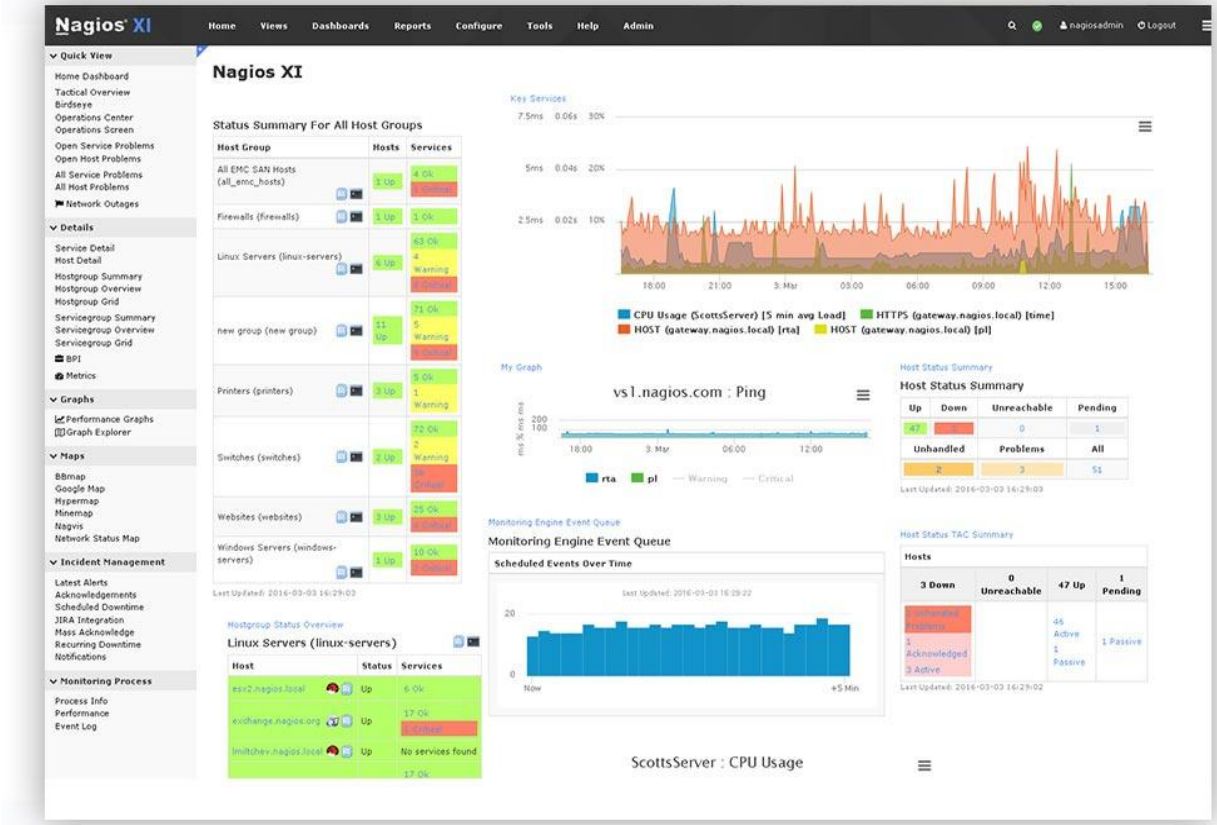

Рисунок 2 – Пользовательский интерфейс Nagios

Zabbix представляет собой многофункциональное решение для наблюдения за состоянием серверов, сетевых компонентов и приложений. Одним из ключевых достоинств платформы является её удобный веб-интерфейс, обеспечивающий простоту навигации и настройки. Также система поддерживает автоматическое определение подключенных устройств и запущенных сервисов, что существенно упрощает интеграцию в

масштабируемые ИТ – среды.

Функции Zabbix включают:

– автоматическое обнаружение оборудования – система способна самостоятельно находить доступные устройства и сервисы в сети, что ускоряет стартовую настройку;

– готовые шаблоны мониторинга – предоставляются пред настроенные профили для отслеживания параметров популярных приложений и служб, что снижает трудозатраты на конфигурацию;

– настройка уведомлений и триггеров – реализована гибкая система оповещений, позволяющая реагировать на изменения метрик в зависимости от заданных условий.

К положительным сторонам можно отнести:

– понятный интерфейс, подходящий как для опытных администраторов, так и для новичков;

– возможность масштабирования, что делает Zabbix привлекательным для использования в крупных организациях.

Однако у Zabbix есть и ряд ограничений:

– высокие аппаратные требования – для корректной работы системы на больших объектах необходим мощный сервер;

– сложности с нетиповыми сценариями – при необходимости реализовать нестандартные условия мониторинга требуется дополнительная доработка и глубокая настройка.

Пользовательский интерфейс Zabbix представлен на рисунке 3.

Рисунок 3 – Пользовательский интерфейс Zabbix

Prometheus представляет собой современное решение для мониторинга, которое активно применяется в облачных и контейнерных средах, таких как Kubernetes. Одним из ключевых достоинств этой системы является её высокая производительность и способность к масштабированию. Prometheus использует временные ряды для хранения метрик, что обеспечивает эффективный сбор, обработку и анализ больших объемов информации.

Функции Prometheus включают

– система метрик на основе временных рядов позволяет сохранять данные с временной меткой, что упрощает отслеживание изменений и выявление тенденций;

– интеграция с Grafana предоставляет возможность построения интерактивных графиков и дашбордов для наглядного представления данных;

– наличие экспортеров даёт возможность собирать информацию с различных источников, включая серверы, базы данных и приложения.

К положительным сторонам можно отнести:

– высокая скорость работы и возможность расширения под большие инфраструктуры;

– активное сообщество разработчиков и регулярное обновление продукта.

Однако у Prometheus есть и ряд ограничений:

– сложность настройки для небольших проектов и стартапов;

– отсутствие встроенной поддержки протокола SNMP, что требует дополнительной интеграции и настройки.

Пользовательский интерфейс Prometheus представлен на рисунке 4.

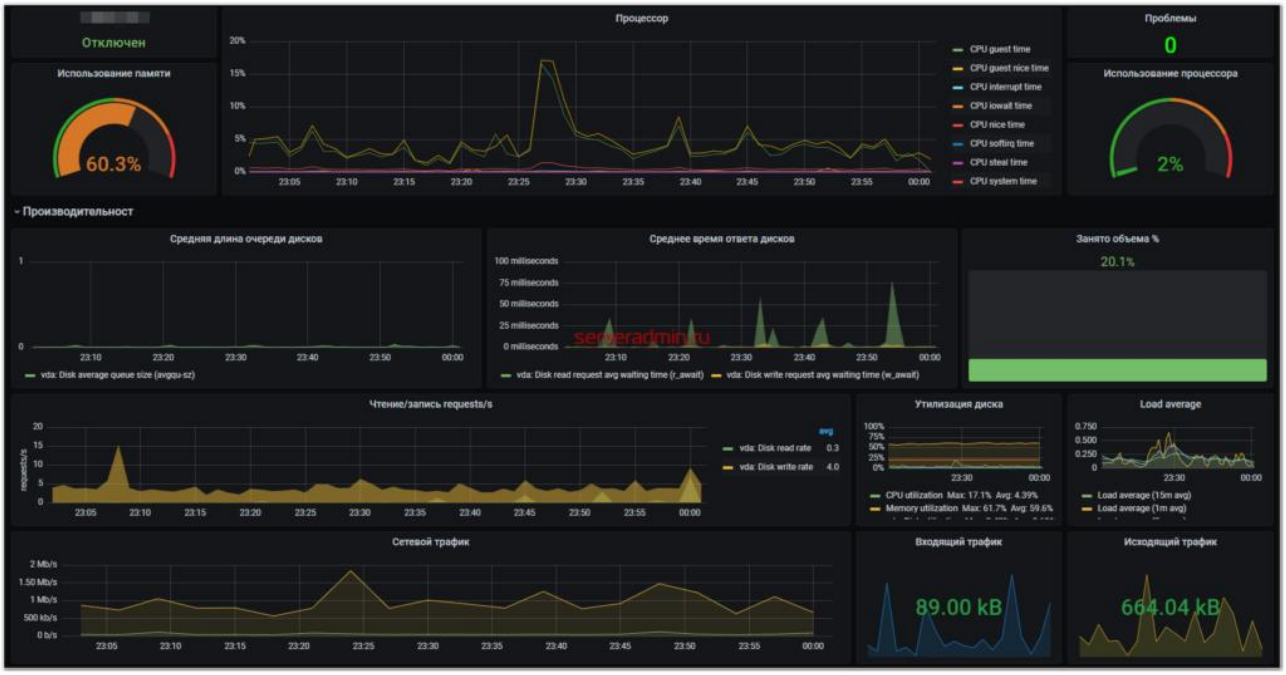

Рисунок 4 – Пользовательский интерфейс Prometheus

Анализ проблем в существующих подходах. Несмотря на наличие множества готовых решений для мониторинга серверов, они имеют ряд ограничений, которые могут быть критичными в определённых условиях:

– сложность внедрения – многие инструменты требуют значительных затрат времени на настройку и обучение сотрудников;

– зависимость от сторонних продуктов – например, Prometheus не имеет встроенного механизма визуализации, поэтому для отображения данных требуется интеграция с Grafana;

– высокая ресурсоёмкость – некоторые решения, такие как Zabbix, предъявляют серьёзные требования к аппаратным ресурсам сервера, особенно при работе с большими объёмами данных;

– ограниченная гибкость – стандартные решения часто не учитывают специфику конкретной организации или уникальные задачи ИТ-инфраструктуры.

Таблица 1 – Сравнительный анализ существующих систем

Характеристика	Nagios	Zabbix	Prometheus
Гибкость настройки	Очень высокая	Средняя	Высокая
Простота настройки	Сложно	Средне	Сложно
Интерфейс и визуализация	Нет современного интерфейса	Встроенный и понятный	Требуется Grafana
Ресурсоёмкость	Средняя	Высокая	Зависит от нагрузки
Зависимость от других инструментов	Требует доработки UI	Минимальна	Часто требуется Grafana

Обоснования необходимости разработки собственной системы мониторинга. Учитывая перечисленные выше проблемы, становится очевидной актуальность разработки собственного решения для мониторинга серверов. Такая система позволит:

– адаптировать функционал под конкретные потребности компании, обеспечивая более точное соответствие бизнес – задачам;

– минимизировать зависимость от сторонних инструментов, что повысит автономность и снизит риски, связанные с обновлениями или прекращением поддержки внешних решений;

– сократить расходы на внедрение и сопровождение, исключив необходимость использования дорогостоящих лицензий и сложных интеграций.

Таким образом, создание собственной системы мониторинга на основе технологий Python, библиотек Paramiko и Matplotlib является оправданным шагом. Это решение способно обеспечить гибкость, простоту внедрения и экономически эффективную эксплуатацию, полностью соответствующую целям и задачам организации.

1.3 Основные метрики мониторинга серверов

В процессе эксплуатации серверов особое значение имеет контроль за их техническим состоянием и производительностью. Это позволяет заранее выявлять потенциальные проблемы, избегать сбоев и эффективно управлять ресурсами. Для этого необходимо отслеживать ключевые метрики, которые дают полное представление о работе системы. Рассмотрим основные параметры, подлежащие мониторингу, и разберем их значение в контексте стабильной работы ИТ-инфраструктуры.

Процессор CPU – это «мозг» сервера, отвечающий за выполнение всех операций, связанных с работой программного обеспечения. Уровень его загрузки является важным индикатором общей производительности системы.

Постоянно высокая нагрузка на CPU может быть признаком перегруженности сервера или неэффективного использования ресурсов. Например, если загрузка процессора регулярно превышает 80–90%, это может негативно сказаться на отзывчивости системы и даже вызвать её зависание.

Мониторинг позволяет фиксировать такие моменты и принимать

корректирующие действия: масштабировать вычислительные мощностей, оптимизировать запущенные процессы или распределять нагрузку между ядрами более равномерно. Особенно важно следить за балансом нагрузки в многопроцессорных системах. Если одно ядро работает на пределе, а другие простаивают, это может указывать на некорректную настройку программного окружения.

Оперативная память RAM – один из самых критичных ресурсов сервера, поскольку она напрямую влияет на скорость обработки данных. При недостатке свободной RAM система начинает использовать область подкачки (swap), которая значительно медленнее, чем физическая память. Это приводит к замедлению работы приложений и увеличению времени отклика.

Контроль над уровнем использования RAM помогает вовремя реагировать на дефицит памяти. Например, можно закрыть малоиспользуемые процессы или увеличить объём доступной памяти. Также важно отслеживать соотношение между физической памятью и swap для предотвращения падения производительности.

Современные ОС активно используют свободную RAM для кэширования данных, что повышает общую эффективность. Однако при критическом снижении объема свободной памяти система начинает очищать кэш, что может также вызвать задержки. Поэтому мониторинг должен включать не только общий уровень занятой памяти, но и детальный анализ её распределения по процессам, кэшу и области подкачки.

Дисковые ресурсы также требуют постоянного наблюдения. Полное заполнение хранилища может привести к невозможности записи новых данных, что вызывает сбои в работе сервисов и приложений. Например, если лог-файлы или временные данные занимают весь объем диска, это может привести к остановке сервера.

Важно не только контролировать объем свободного места, но и отслеживать скоростные показатели чтения и записи. Замедление дисковой

подсистемы может свидетельствовать о аппаратных проблемах или перегрузке. Мониторинг позволяет своевременно очищать старые файлы, архивировать информацию или увеличивать ёмкость хранилища.

Отдельное внимание уделяется типу используемых накопителей. Твердотельные диски (SSD) обеспечивают высокую скорость работы, однако имеют ограниченный ресурс циклов записи. Чтобы избежать внезапного выхода из строя, рекомендуется использовать диагностические утилиты, такие как smartctl, которые предоставляют информацию о состоянии дисков и уровне износа.

Эффективное управление серверами невозможно без комплексного подхода к анализу метрик. Показатели загрузки CPU, использование RAM и состояние дисков тесно взаимосвязаны. Например, нехватка оперативной памяти может вызвать повышенную активность подкачки, что увеличит нагрузку на диск и, как следствие, повысит I/O wait на процессоре.

Поэтому при организации мониторинга важно учитывать не только отдельные параметры, но и их взаимодействие друг с другом. Такой подход позволяет не просто фиксировать возникшие проблемы, но и понимать их причины. Например, одновременно высокая загрузка процессора и оперативной памяти в сочетании с медленной работой дисков могут указывать на проблемы с вводом-выводом.

Также необходимо учитывать специфику задач, выполняемых сервером. Для веб-сервера критичным может быть быстрый отклик на запросы, тогда как для баз данных важна скорость обработки данных. В зависимости от назначения сервера, следует адаптировать набор отслеживаемых метрик и пороговые значения, чтобы добиться максимальной эффективности мониторинга.

1.4 Принципы работы SSH, сбора данных и их визуализации

Для реализации эффективной системы мониторинга серверов необходимо понимать и грамотно использовать ключевые технологии, обеспечивающие сбор и отображение данных. К таким технологиям относятся протокол SSH, методы получения информации с удалённых серверов и инструменты визуализации.

Протокол SSH (Secure Shell) используется как основной инструмент для безопасного удалённого подключения к серверам. Он обеспечивает зашифрованное соединение между клиентом и сервером, что позволяет выполнять команды на удалённой машине без риска перехвата информации. Протокол активно применяется системными администраторами для управления серверами, диагностики состояния систем и сбора данных.

В рамках разработанной системы мониторинга SSH служит средством получения информации о текущем состоянии сервера – например, уровня загрузки процессора, объёма используемой оперативной памяти и доступного дискового пространства. Для автоматизации взаимодействия с серверами в Python применяется библиотека Paramiko, которая позволяет программно устанавливать SSH-соединение и выполнять удалённые команды.

Сбор данных с серверов является важным этапом в работе системы мониторинга. Метрики можно получать разными способами, в зависимости от типа сервера и его конфигурации. Один из подходов — выполнение стандартных команд через SSH, таких как top, free и df, которые возвращают данные о загрузке CPU, использовании RAM и состоянии дисков. Также возможен анализ логов и файлов, содержащих информацию о сетевой активности или ошибках приложений. В некоторых случаях может применяться установка специальных агентов на сервер для передачи данных на центральный узел.

После получения данных важно представить их в понятном виде. Для

этого используются инструменты визуализации. Например, в Python для построения графиков применяется библиотека Matplotlib, которая позволяет отображать динамику изменения метрик в режиме реального времени. Альтернативой могут служить более продвинутые инструменты, такие как Grafana, предоставляющая возможность создания интерактивных дашбордов. Также данные можно сохранять в формате CSV или JSON для последующего анализа и формирования отчётов.

Таким образом, использование протокола SSH, методов сбора данных и средств визуализации составляет основу функционирования системы мониторинга. Это позволяет не только собирать актуальную информацию о состоянии серверов, но и представлять её в удобной форме для принятия оперативных решений.

Выводы по главе 1

В первой главе был проведен детальный анализ предметной области, включая описание компании "ВорлдИнтерТех РУС" и её основных направлений деятельности. Особое внимание уделялось обзору существующих решений для мониторинга серверов, таких как Nagios, Zabbix и Prometheus. На основе тщательной оценки их функциональных возможностей, архитектурных особенностей и уровня сложности поддержки была обоснована необходимость создания собственной системы мониторинга, адаптированной под конкретные условия эксплуатации и минимизирующей зависимость от стороннего программного обеспечения. Такой подход позволяет более точно учитывать внутренние процессы организации и оперативно вносить изменения в систему без ограничений, связанных с внешними платформами.

Кроме того, были выделены ключевые метрики, обеспечивающие объективную оценку состояния серверов: загрузка центрального процессора (CPU), использование оперативной памяти (RAM) и дискового пространства (Disk). Эти параметры легли в основу архитектуры будущей системы, определив её функциональные требования: сбор данных в удалённом режиме

по защищённому протоколу SSH, их обработка и наглядная визуализация. Акцент был сделан на надёжности передачи информации и возможности масштабирования системы без серьёзных изменений её структуры.

Выбор технологий также не был случайным. Язык Python и библиотеки Paramiko и Matplotlib были выбраны за счёт своей гибкости, богатого функционала и активного сообщества, что гарантирует поддержку и развитие проекта в долгосрочной перспективе. Их применение обеспечивает не только техническую реализацию основных задач, но и упрощает процесс доработки и расширения системы под изменяющиеся требования бизнеса.

Проведённый анализ позволил не только определить направление разработки, но и сформировать целостное понимание задач, стоящих перед системой мониторинга. Этот этап стал ключевым для выработки технического решения, способного обеспечить непрерывный контроль за ИТ-инфраструктурой, повысить её надёжность и оперативность реагирования на возможные сбои.

Глава 2 Проектирование web-приложения.

2.1 Выбор технологий и инструментов

Выбор технологий и инструментов играет важную роль на этапе проектирования системы мониторинга серверов, поскольку от этого зависит не только функциональность и производительность решения, но и удобство его разработки, тестирования и дальнейшего сопровождения. В рамках данного исследования рассматриваются ключевые аспекты выбора языка программирования, библиотек, виртуализационной среды и прочих компонентов, которые будут формировать архитектуру и функциональные возможности будущей системы.

Для реализации проекта был проведен сравнительный анализ нескольких популярных языков программирования – Python, JavaScript и Java. Основными критериями оценки стали: простота синтаксиса, наличие стандартной и сторонней библиотечной поддержки, кроссплатформенность, масштабируемость, производительность, а также активность сообщества и доступность документации.

Результат сравнительного анализа языков программирования представлен в таблице 2.

Таблица 2 – Сравнительный анализ языков программирования

Характеристика	Python	JavaScript	Java
Простота и читаемость кода	Python выделяется своим понятным и лаконичным синтаксисом, что упрощает чтение и поддержку кода	JavaScript более гибкий, но его сложнее воспринимать из-за разнообразия подходов к написанию кода	Java требует более строгого и детального синтаксиса, что может затруднять чтение кода

Продолжение таблицы 2

Характеристика	Python	JavaScript	Java
Широкая стандартная библиотека	Python предлагает богатую встроенную библиотеку с множеством инструментов для разработки приложений	JavaScript имеет базовую библиотеку, которая менее обширна и универсальна по сравнению с Python	Java также предоставляет широкую стандартную библиотеку, но она менее гибкая, чем у Python
Кроссплатформенность	Python работает на разных операционных системах, что делает его универсальным выбором	JavaScript также поддерживает кроссплатформенность, но для серверной части часто требуется Node.js	Java работает на любой платформе, но требует установки Java Virtual Machine (JVM)
Обширное сообщество и экосистема	Python обладает активным сообществом и огромным количеством сторонних библиотек, что упрощает разработку	JavaScript имеет большое сообщество, но его экосистема менее структурирована и предсказуема	Java имеет развитое сообщество, но оно менее динамично и инновационно по сравнению с Python

На основании данных из таблицы 2 можно сделать вывод, что Python обладает рядом важных преимуществ – включая понятный и лаконичный синтаксис, богатую стандартную библиотеку, кроссплатформенную совместимость и поддержку активного сообщества разработчиков. Эти характеристики делают его оптимальным выбором для реализации системы мониторинга серверов.

Выбор подходящих библиотек и фреймворков играет важную роль в ускорении и упрощении разработки системы мониторинга. Эти инструменты предоставляют готовые решения для выполнения стандартных задач, таких как сетевое взаимодействие, сбор и обработка данных, а также их визуализация. При подборе библиотек учитывались ключевые критерии – функциональная полнота, удобство применения, возможность расширения, наличие подробной документации и активная поддержка сообществом

разработчиков. Данный подход позволил значительно сократить время на реализацию функционала и повысить надёжность разрабатываемого решения.

Таблица 3 – Сравнительный анализ библиотек

Характеристика	Paramiko (PYTHON)	Fabric (PYTHON)	Netmiko (PYTHON)
Поддержка SSH	Полноценная поддержка SSH, включая выполнение команд и передачу файлов	Упрощенная работа с SSH, но менее гибкая в сравнении с Paramiko	Специализированная библиотека для работы с сетевыми устройствами через SSH
Легкость использования	Требует базовых знаний Python, но достаточно прост в освоении	Очень прост в использовании, но менее мощный	Более сложен в освоении, но предоставляет больше возможностей
Интеграция с другими библиотеками	Легко интегрируется с Matplotlib для визуализации данных	Ограниченная интеграция с другими библиотеками	Хорошая интеграция с библиотеками для анализа данных
Сообщество и документация	Активное сообщество и подробная документация	Меньшее сообщество и ограниченная документация	Узкоспециализированное сообщество, но с хорошей поддержкой

Таблица 3 показывает, что библиотека Paramiko отличается рядом сильных сторон, среди которых – полная реализация протокола SSH, простота в применении и удобная совместимость с другими инструментами Python. Благодаря этим качествам, она становится одним из наиболее подходящих решений при создании системы для мониторинга серверной инфраструктуры.

Система виртуализации играет важную роль в тестировании и развертывании системы мониторинга серверов. Перед выбором системы необходимо учитывать требования к производительности, масштабируемости и удобству использования.

Результат сравнительного анализа систем виртуализации представлен в таблице 4.

Таблица 4 – Сравнительный анализ систем виртуализации

Характеристика	Virtualbox	Vmware	Docker
Простота установки и использования	Простая установка и использование, не требует сложной настройки	Требует установки и настройки, что может быть более сложным процессом	Очень прост в использовании, но требует знания контейнеризации
Эффективность для тестирования	Идеально подходит для тестирования небольших и средних проектов	Подходит для проектов любого размера, но может быть избыточным для небольших приложений	Подходит для тестирования микросервисов и контейнеризированных приложений
Производительность	Обеспечивает высокую производительность при работе с ограниченными ресурсами	Может потреблять больше ресурсов, чем VirtualBox	Очень эффективен, но требует поддержки контейнеризации
Распространенность и поддержка	Широко распространен и поддерживается сообществом разработчиков	Имеет активное сообщество, но менее доступен для начинающих	Одна из самых популярных систем контейнеризации

Продолжение таблицы 4

Масштабируемость	Ограниченная масштабируемость. Подходит только для небольших и средних проектов.	Высокая масштабируемость. Поддерживает большие проекты и распределенные системы	Высокая масштабируемость для микросервисов и контейнеризированных приложений
Безопасность	Минимальные встроенные механизмы безопасности. Подходит для локального тестирования	Предоставляет расширенные возможности безопасности, такие как шифрование и изоляция	Высокий уровень безопасности, но требует дополнительной настройки политик

Из проведенного сравнения видно, что VirtualBox обладает рядом значительных преимуществ перед другими системами виртуализации, особенно в контексте тестирования небольших и средних проектов, а также в условиях ограниченных ресурсов.

Прежде чем приступить к этапу программной реализации, необходимо выбрать подходящую интегрированную среду разработки (IDE), которая будет обеспечивать удобство и эффективность в процессе написания кода. IDE представляет собой комплексное программное обеспечение, включающее в себя инструменты, необходимые для создания, запуска, отладки и тестирования программ. При разработке на языке Python программист активно использует такие компоненты среды, как текстовые редакторы, библиотеки, отладчики, а также средства для контроля версий и тестирования. Использование IDE позволяет существенно повысить продуктивность разработки, улучшить структуру и читаемость кода, а также сократить количество ошибок.

Ключевые возможности современных интегрированных сред разработки можно представить следующим образом:

– наличие редактора с подсветкой синтаксиса и функцией автодополнения;

– встроенный отладчик, облегчающий выявление и устранение ошибок;

– поддержка интеграции с системами контроля версий (например, Git);

– совместимость с различными языками программирования и технологиями;

– средства для организации проектов и управления файлами;

– инструменты для автоматизации рутинных задач (шаблоны, генераторы кода и др.);

– встроенные модули для взаимодействия с базами данных;

– поддержка написания и выполнения модульных и интеграционных тестов;

– кроссплатформенность и адаптация под разные операционные системы;

– возможность индивидуальной настройки среды под особенности проекта или предпочтения разработчика.

В рамках данной работы в качестве основной среды разработки был выбран PyCharm IDE, ориентированная на разработку на языке Python.

Главная страница сайта представлена на рисунке 5.

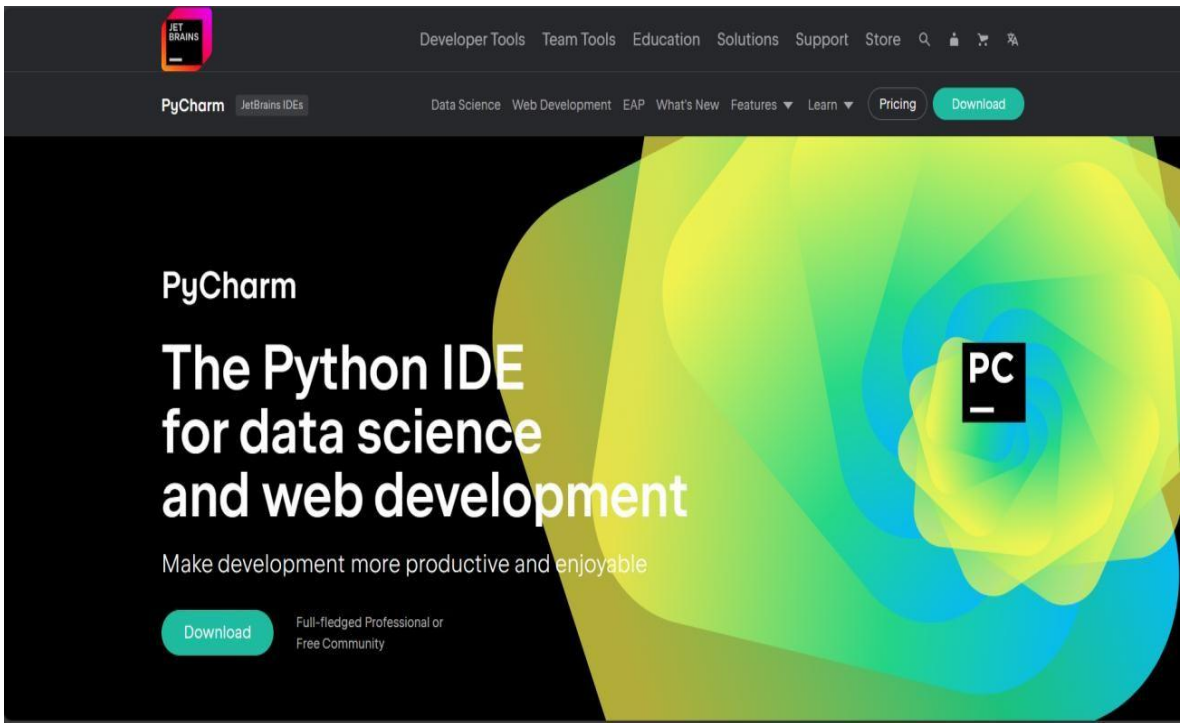

Рисунок 5 – Pycharm главная страница сайта

PyCharm представляет собой интегрированную среду разработки (IDE), ориентированную на язык программирования Python и созданную компанией JetBrains. Данная среда предоставляет разработчику обширный набор инструментов, охватывающих все этапы программной деятельности — от написания и отладки кода до тестирования и управления проектной структурой. Благодаря своей функциональности, удобству использования и регулярной поддержке, PyCharm занимает ведущие позиции среди IDE, применяемых для разработки на Python, и пользуется широкой популярностью в профессиональном сообществе программистов.

Преимущества PyCharm:

– широкий набор инструментов для разработки на Python;
– автодополнение кода и подсветка синтаксиса;
– интеграция с системами контроля версий;
– поддержка виртуальных окружений;
– отладка кода;

– поддержка различных фреймворков;

– доступность в двух версиях. Community Edition и Professional Edition.

Недостатки PyCharm:

– некоторые функции могут быть сложными для новичков;

– не все функции доступны в Community Edition;

– некоторые пользователи могут считать интерфейс PyCharm слишком громоздким.

На основании вышеизложенного, я выбрал IDE PyCharm.

2.2 Разработка логической модели автоматизированной информационной системы

В данном разделе осуществляется разработка логической модели автоматизированной информационной системы (АИС) с применением нотации Unified Modeling Language (UML). Логическая модель включает в себя диаграмму прецедентов, схему развертывания, диаграмму активности и схемы совместной работы.

Диаграмма прецедентов занимает одно из центральных мест в методологии объектно-ориентированного анализа и проектирования ПО. Ее основное назначение — графическое отображение функциональных возможностей системы с точки зрения взаимодействия с внешними акторами. Данный тип диаграмм позволяет четко определить, какие действия должна выполнять система и каким образом эти действия будут задействованы пользователями или другими системами. Таким образом, диаграмма прецедентов служит важным инструментом для формализации и уточнения требований на ранних этапах разработки программного обеспечения.

На Рисунке 6 представлена диаграмма вариантов использования.

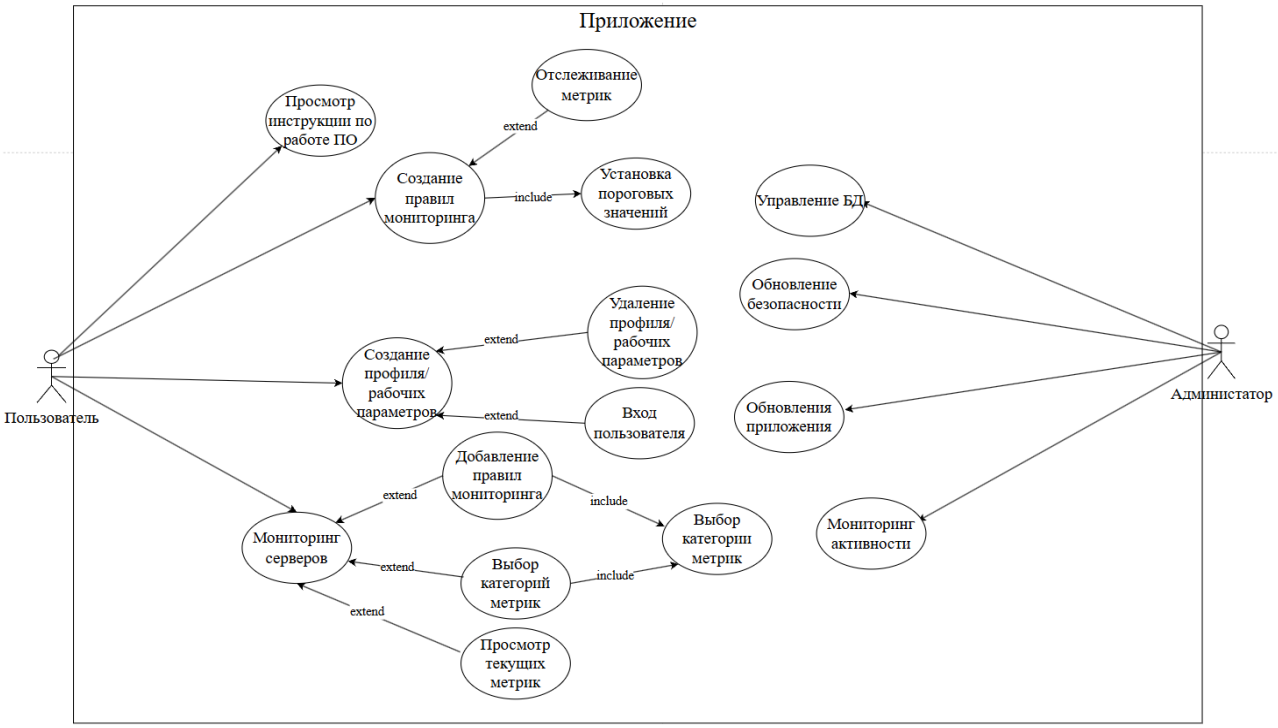

Рисунок 6 – Диаграмма вариантов использования

Диаграмма включает два типа пользователей «Пользователь» и «Администратор», каждый из которых взаимодействует с системой через определённые функции (прецеденты).

Пользователь:

– вход пользователя: базовый прецедент, обеспечивающий аутентификацию пользователя и предоставление доступа к функциям системы;

– отслеживание метрик: прецедент, позволяющий пользователю наблюдать за текущими значениями метрик работы серверов (например, загрузка CPU, память, дисковое пространство);

– выбор категории метрик: расширяет (extend) прецедент "Мониторинг серверов", предоставляя возможность фильтрации метрик по категориям (например, производительность, ресурсы, сетевые показатели);

– просмотр текущих метрик: дополнительный прецедент, который позволяет детально анализировать актуальные значения метрик в режиме реального времени;

– установка пороговых значений: прецедент, позволяющий пользователю настроить пороговые значения метрик;

– добавление правил мониторинга: расширяет прецедент, позволяя добавлять новые условия отслеживания;

– удаление профиля/рабочих параметров: прецедент, обеспечивающий возможность очистки или удаления настроек рабочего профиля, если они нужны;

– просмотр инструкций по работе ПО: прецедент, предоставляющий доступ к справочной информации и руководствам по использованию системы.

Администратор:

– управление БД: прецедент, обеспечивающий администрирование базы данных системы, включая её настройку, обновление и обслуживание;

– обновление безопасности: прецедент, направленный на обеспечение защиты системы от внешних угроз и внутренних нарушений;

– обновление приложения: прецедент, связанный с внедрением новых версий программного обеспечения, и добавлением функционала;

– мониторинг активности: прецедент, позволяющий отслеживать все действия пользователей и администраторов в системе, включая логирование событий.

Взаимодействия и зависимости:

– прецедент «Создание правил мониторинга» включает в себя «Установка пороговых значений», что указывает на необходимость настройки пороговых значений как части процесса создания правил;

– прецедент «Мониторинг серверов» расширяется дополнительными прецедентами, такими как «Выбор категории метрик» и «Просмотр текущих метрик», что позволяет пользователю более настраивать и анализировать данные;

– функционал администратора сосредоточен на глобальном управлении системой, включая безопасность, обновления и контроль

активности пользователей.

Схема развертывания описывает физическую архитектуру системы, включая распределение компонентов и их размещение на аппаратном обеспечении.

На Рисунке 7 показана схема развертывания разрабатываемой АИС.

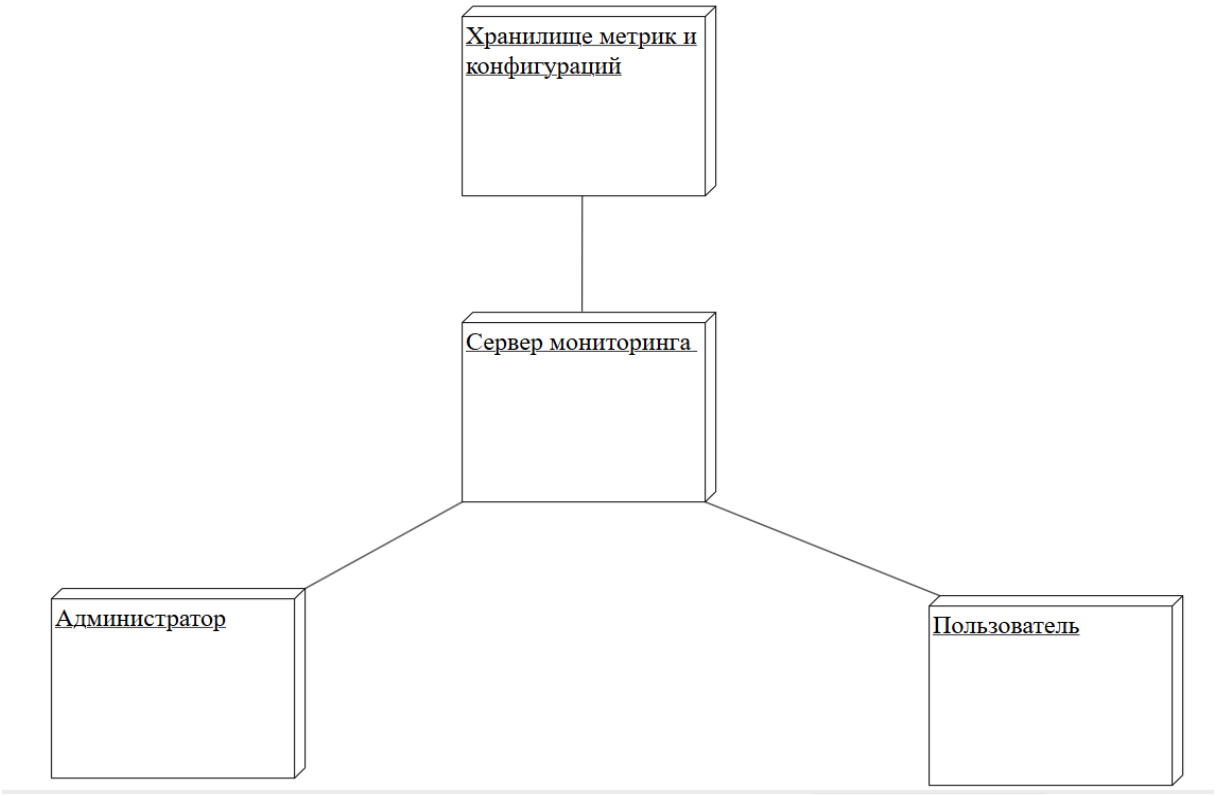

Рисунок 7 – Схема развертывания

В верхней части диаграммы располагается «Хранилище метрик и конфигураций», которое служит централизованным хранилищем данных о состоянии системы, конфигурационных параметров и собранных метрик. Это хранилище тесно связано с «Сервером мониторинга», находящимся ниже, который отвечает за сбор, обработку и анализ информации о состоянии серверов и других компонентов IT-инфраструктуры.

Сервер мониторинга выполняет роль центрального узла системы и взаимодействует с двумя типами участников: Администратором и Пользователем.

Администратор использует сервер мониторинга для настройки системы, управления конфигурациями, создания правил оповещений и диагностики сбоев.

Пользователь обращается к серверу мониторинга для просмотра визуализированных метрик, анализа производительности и получения отчетов о состоянии системы.

Такая архитектура обеспечивает эффективное разделение ролей и централизованный контроль над инфраструктурой мониторинга.

Диаграмма активности представляет собой последовательность действий или процессов в системе, позволяя визуализировать шаги, необходимые для выполнения определенной задачи или достижения цели. На Рисунке 8 приведена диаграмма активности, иллюстрирующая основные процессы в рамках разрабатываемой АИС.

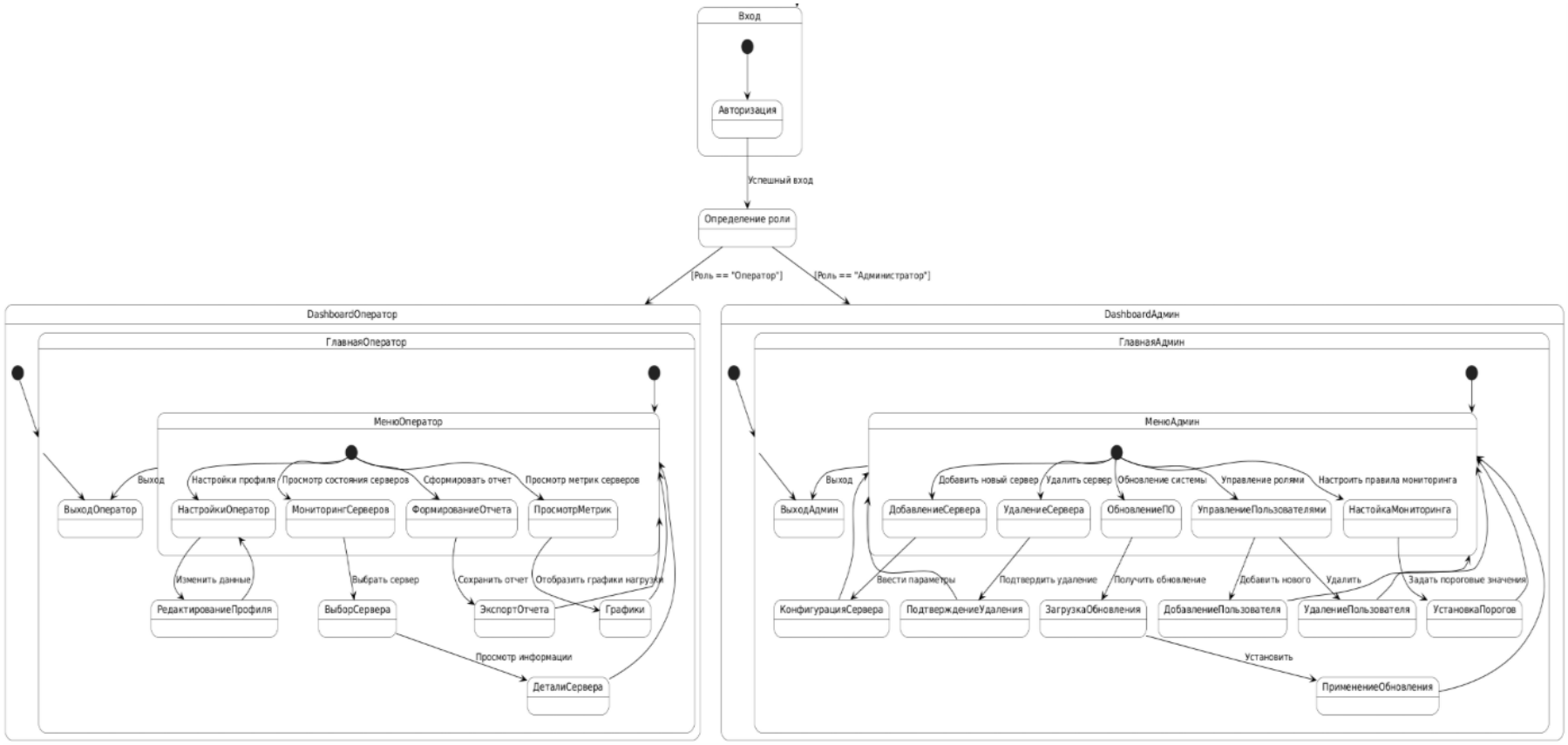

Рисунок 8 – Диаграмма активности

Диаграмма начинается с начальной точки – входа в систему, обозначенной черным кружком. Пользователь инициирует процесс авторизации в блоке «Вход». В случае неуспешной авторизации пользователь перенаправляется обратно на повторную попытку входа. Если вход осуществляется успешно, выполняется переход к этапу определения роли пользователя в системе.

После определения роли пользователь может быть отнесен к одной из двух категорий: «Оператор» или «Администратор». В зависимости от присвоенной роли осуществляется переход к соответствующему интерфейсу, предоставляющему доступ к определённому функционалу.

Если пользователь входит как Оператор, он попадает на главную панель, откуда доступны следующие действия:

– мониторинг серверов оператор выбирает интересующий сервер, просматривает информацию о его текущем состоянии, после чего возвращается в главное меню;

– просмотр метрик осуществляется отображение графиков нагрузки и других показателей производительности выбранных серверов. По завершении просмотра пользователь возвращается в меню;

– формирование отчета оператор формирует отчет по текущим или архивным данным мониторинга, экспортирует его в подходящем формате и возвращается в меню;

– настройки профиля предоставляется возможность изменения персональных данных пользователя;

– выход завершение текущей сессии и возврат к начальному этапу.

Если пользователь входит как Администратор, ему предоставляется расширенный набор функций:

– добавление сервера включает ввод конфигурационных параметров нового сервера и его регистрация в системе;

– удаление сервера предоставляет возможность исключения сервера из

списка с предварительным подтверждением действия;

– настройка мониторинга. Администратор задает пороговые значения, определяет правила оповещения и параметры логирования;

– управление пользователями осуществляется добавление или удаление учетных записей, а также изменение их ролей;

– обновление программного обеспечения включает загрузку и установку новых версий системы;

– выход завершение работы администратора и выход из системы.

Таким образом, представленная диаграмма активности иллюстрирует логическую последовательность действий пользователей системы мониторинга серверов, начиная с момента входа и заканчивая выполнением задач в рамках своей роли. Такое распределение ролей и действий обеспечивает четкое разграничение полномочий, надежность эксплуатации и удобство взаимодействия с системой.

Диаграмма классов наглядно демонстрирует архитектуру системы мониторинга серверов. Каждый класс выполняет чётко определённую роль, что обеспечивает модульность и удобство дальнейшего развития программы. Диаграмма классов представлена рисунок 9

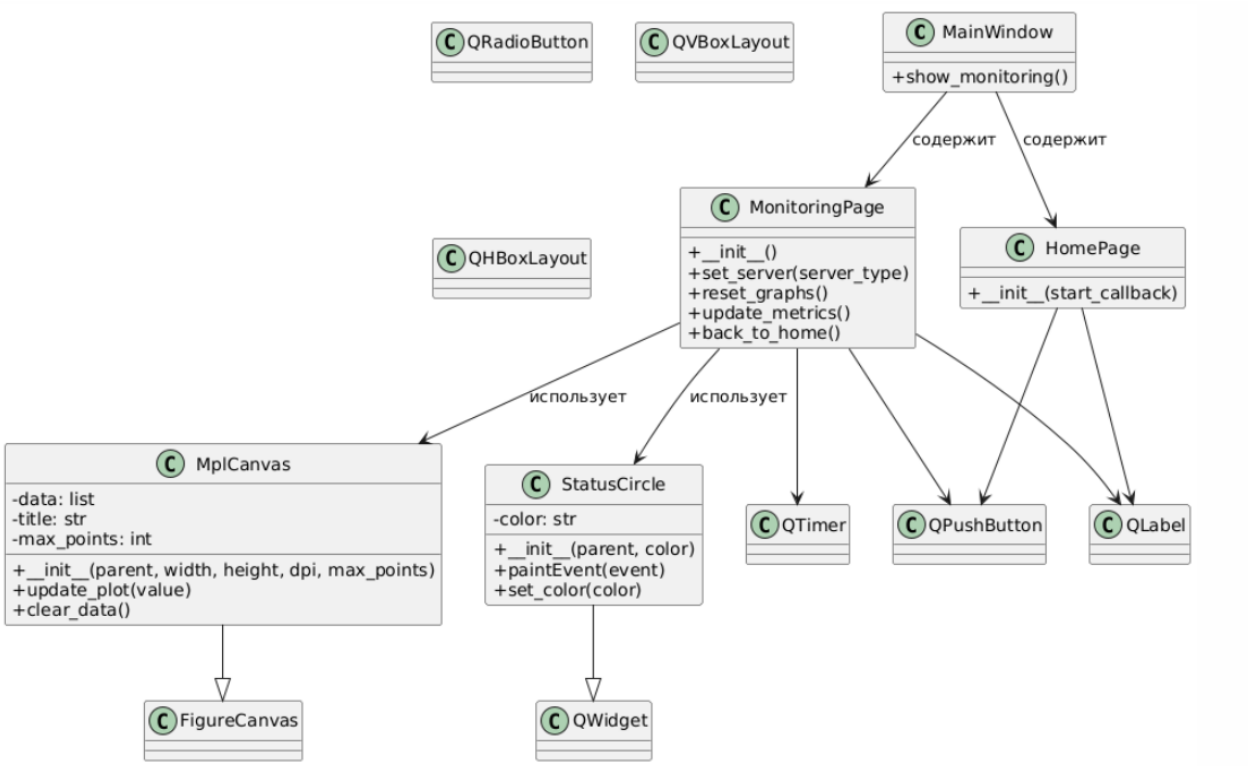

Рисунок 9 – Диаграмма классов

На рисунке представлена диаграмма классов, которая отражает структуру разработанного приложения для мониторинга серверов. Система состоит из нескольких ключевых компонентов, которые взаимодействуют между собой для сбора метрик, визуализации данных и управления интерфейсом пользователя.

2.3 Требования к аппаратно-программному обеспечению автоматизированной информационной системы

Разработка системы мониторинга серверов требует обеспечения соответствующего уровня аппаратно-программной поддержки, способной обеспечить стабильную, надежную и производительно функционирующую архитектуру. При этом важно учитывать существующие на предприятии вычислительные ресурсы и стремиться к их максимально эффективному использованию без необходимости в кардинальном обновлении инфраструктуры.

В состав технического комплекса, обеспечивающего функционирование системы мониторинга, должны входить следующие категории оборудования. Сервер, на котором размещается система управления базами данных и логика мониторинга клиентские устройства пользователей (операторов и администраторов системы).

Минимальные технические требования к серверу мониторинга:

– процессор с тактовой частотой не ниже 2.4 ГГц;

– объем оперативной памяти – не менее 4 ГБ;

– свободное пространство на жестком диске – от 25 ГБ и выше (в зависимости от объема собираемых метрик и продолжительности хранения);

– стабильное подключение к сети Интернет или локальной сети предприятия;

– операционная система семейства Windows (рекомендуется Windows Server последних версий).

– требования к пользовательским компьютерам:

– процессор с частотой от 2 ГГц;

– объем оперативной памяти – от 2 ГБ;

– доступ к сети Интернет или внутренней сети предприятия;

– наличие современной операционной системы (Windows, Linux, MacOS и др.);

– установленный веб-браузер, поддерживающий современные веб-стандарты (Google Chrome, Mozilla Firefox, Opera и т.д.).

Особое внимание стоит уделить совместимости программного обеспечения с различными платформами, что обеспечивает гибкость при выборе клиентских устройств и снижает порог вхождения для персонала. Такой подход позволяет использовать существующие на предприятии рабочие станции без необходимости дополнительных вложений в новое оборудование.

После того как были сформулированы и обоснованы требования к аппаратно-программному обеспечению разрабатываемой системы мониторинга серверов, в следующем разделе будут подведены итоги второй главы.

Выводы по главе 2

Во второй главе была проведена комплексная работа, направленная на обоснование и выбор оптимальных технологий и инструментов для разработки системы мониторинга серверов. Были рассмотрены и сравнены различные языки программирования, включая Python, C++, Java и другие. На основании таких критериев, как удобство разработки, читаемость кода, поддержка сообщества и наличие готовых библиотек, было принято решение использовать Python в качестве основного языка реализации.

Особое внимание уделялось выбору средств удалённого подключения к серверам. Среди проанализированных SSH-библиотек, таких как Paramiko, Netmiko и AsyncSSH, предпочтение отдано Paramiko за счёт её устойчивости, наличия хорошей документации и активной поддержки. Это позволило организовать надёжное сетевое взаимодействие и выполнение команд на удалённых машинах через безопасное SSH-соединение.

Также были изучены варианты систем виртуализации, включая VirtualBox, VMware и Hyper-V. В результате анализа выбран VirtualBox, поскольку он сочетает в себе простоту настройки, широкую совместимость с различными операционными системами и развитые возможности

автоматизации. Его использование оказалось особенно удобным при создании тестовой среды для проверки работы программы.

Для обеспечения комфортной и эффективной разработки был выбран инструмент PyCharm мощная IDE, специально ориентированная на работу с Python и предоставляющая широкие возможности по отладке, контролю версий и управлению проектом. Интеграция с Git, автодополнение кода и гибкая система настройки делают этот редактор идеальным выбором для создания распределённых решений.

Проведённый анализ и принятые решения обеспечивают технологическую основу для дальнейшего проектирования и реализации системы. Выбор технологий выполнен с учётом актуальных требований и современных тенденций, что позволяет рассчитывать на надёжность, масштабируемость и удобство сопровождения разрабатываемого программного продукта.

Кроме того, были учтены аспекты производительности и простоты внедрения решений в существующую ИТ – инфраструктуру компании. Python показал высокую эффективность при работе с сетевыми протоколами и сбором данных о состоянии серверов. Простой интерфейс библиотеки Paramiko и богатый функционал Matplotlib дали возможность быстро реализовать ключевые компоненты системы.

Выбранный комплекс технологий позволяет создать гибкое, надёжное и легко адаптируемое решение, подходящее как для текущего проекта, так и для последующего расширения возможностей.

Глава 3 Реализация системы мониторинга серверов.

3 Разработка системы мониторинга серверов

Первым этапом стало создание программы для локального мониторинга сервера. Для этого в начале была настроена среда разработки и установлены необходимые зависимости (библиотеки psutil, matplotlib, paramiko), установка на рисунке 10.

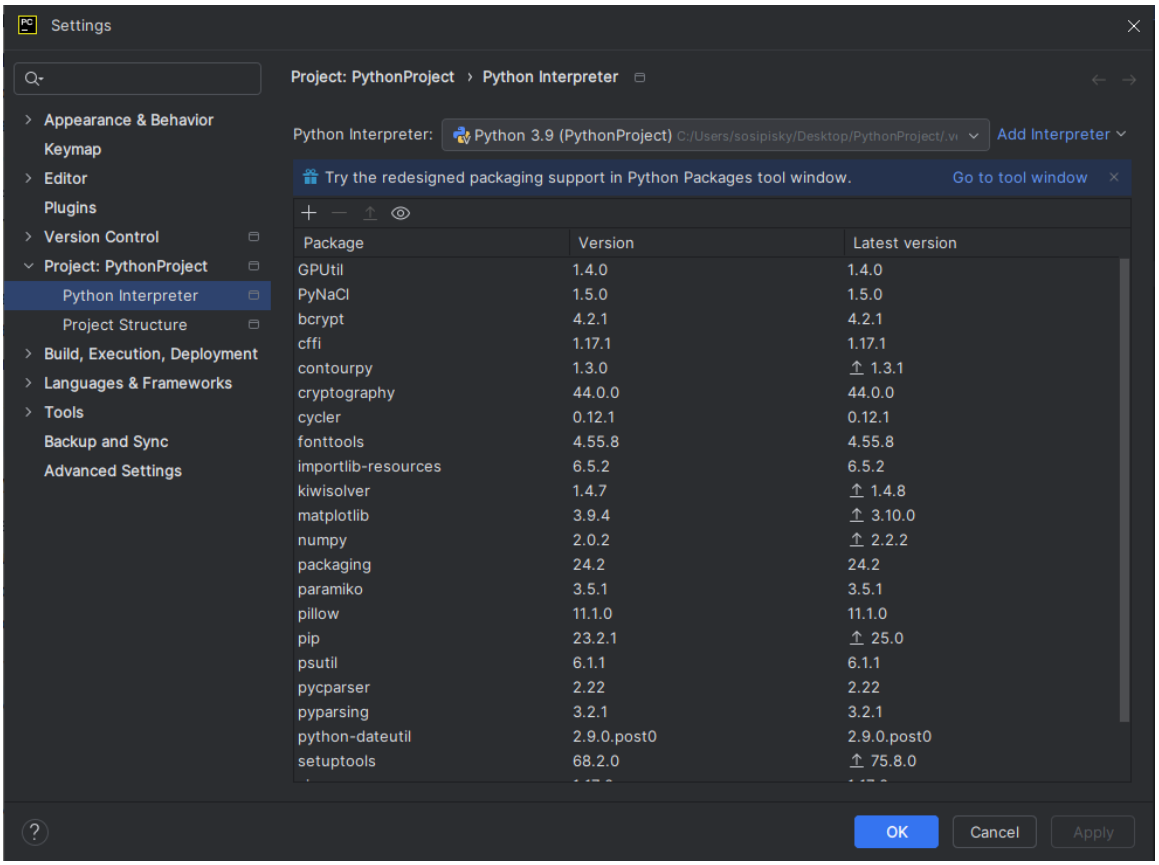

Рисунок 10 – Установка зависимостей

Затем был создан Python-скрипт для сбора метрик с использованием библиотеки psutil. В методе инициализации программы были определены основные функции для получения данных о загрузке CPU, RAM и дискового пространства. Вот пример кода для этих функций рисунок 11.

```python
def get_cpu_usage():  1 usage
    """Получение загрузки CPU (%)"""
    return psutil.cpu_percent(interval=1)

def get_memory_usage():  1 usage
    """Получение использования оперативной памяти (%)"""
    memory = psutil.virtual_memory()
    used_percent = (memory.used / memory.total) * 100
    return round(used_percent, 2)

def get_disk_usage(path="/"):  1 usage
    """Получение использования дискового пространства (%)"""
    if platform.system() == "Windows":
        path = "C:\\"
    disk = psutil.disk_usage(path)
    return disk.percent
```

Рисунок 11 – Функции сбора метрик

После успешной реализации локального мониторинга была начата работа над удаленным мониторингом сервера. Для подключения к удаленному серверу использовалась библиотека paramiko, которая обеспечивает безопасное SSH-соединение. В файле конфигурации были указаны параметры подключения: IP-адрес сервера, имя пользователя и пароль (или путь к SSH-ключу) рисунок 12.

```python
# Создаем SSH-клиент
ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

def connect_to_server():  2 usages
    """Подключение к удаленному серверу"""
    try:
        ssh_client.connect(SERVER_IP, username=USERNAME, password=PASSWORD)
        print(f"Successfully connected to {SERVER_IP}")
    except Exception as e:
        print(f"Failed to connect: {e}")
        return False
    return True
```

Рисунок 12 – Конфигурация параметров подключения

На следующем этапе были реализованы функции для сбора удаленных метрик. Были использованы системные команды (top, free, df), выполняемые через SSH-соединение. Результаты выполнения команд

преобразовывались в числовые значения для их дальнейшего использования. Вот пример кода для сбора метрик рисунок 13.

```python
def get_remote_cpu_usage():  1 usage
    """Получение загрузки CPU (%)"""
    cmd = "top -bn1 | grep 'Cpu(s)' | awk '{print $2}' | cut -d'.' -f1"
    stdin, stdout, stderr = ssh_client.exec_command(cmd)
    cpu = stdout.read().decode().strip()
    return int(cpu) if cpu.isdigit() else 0

def get_remote_memory_usage():  1 usage
    """Получение использования RAM (%)"""
    cmd = "free -m | awk 'NR==2{printf \"%.2f\", $3*100/$2 }'"
    stdin, stdout, stderr = ssh_client.exec_command(cmd)
    memory = stdout.read().decode().strip()
    return float(memory) if memory.replace( _old: '.',  _new: '',  _count: 1).isdigit() else 0

def get_remote_disk_usage():  1 usage
    """Получение использования диска (%)"""
    cmd = "df -h | awk '$NF==\"/\"{print $5}' | tr -d '%'"
    stdin, stdout, stderr = ssh_client.exec_command(cmd)
    disk = stdout.read().decode().strip()
    return int(disk) if disk.isdigit() else 0
```

Рисунок 13 – Функции сбора удаленных метрик

Графическая часть состоит из трех отдельных графиков:

– cpu usage. Отображает загрузку процессора;

– memory usage. Показывает использование оперативной памяти;

– disk usage. Демонстрирует занятость дискового пространства.

Реализация графиков. Рисунок 14

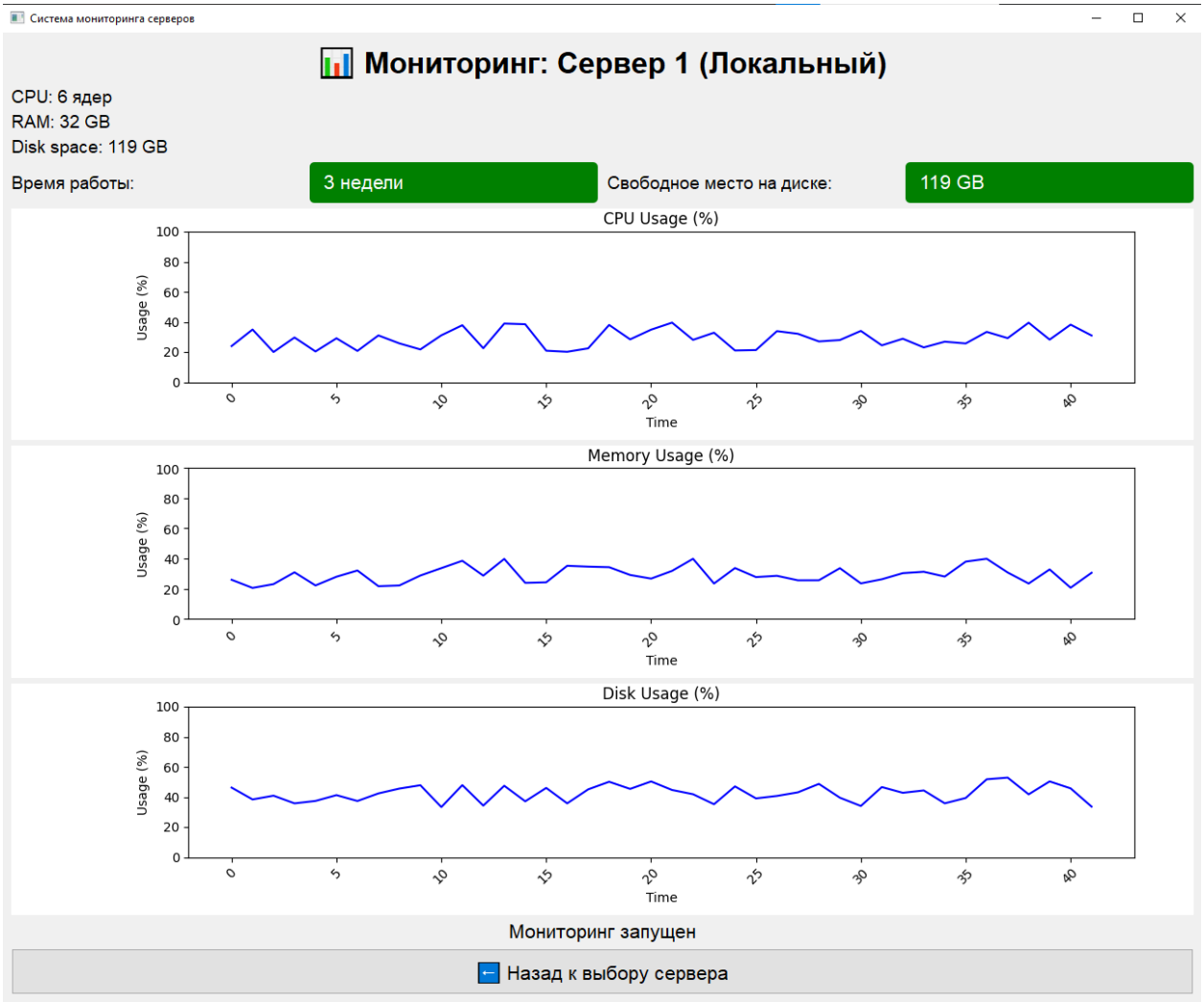

Рисунок 14 – Реализация графиков

Реализация графиков имеет светлую тему с синими линиями для лучшего восприятия. Оси X и Y имеют подписи:

– ось Y: "Usage (%)";

– ось X: "Time (last 50 measurements)".

Это делает графики более информативными и удобными для анализа.

Для тестирования удаленного мониторинга была создана виртуальная машина (ВМ) с помощью VirtualBox. Основные шаги включали:

Установка операционной системы:

– была установлена Ubuntu Server 24.04 LTS Рисунок 15;

– настройка сети;

– выбран режим "Bridged Adapter", чтобы ВМ получила свой собственный IP-адрес в локальной сети рисунок 16;

– установка SSH-сервера;

– для обеспечения удаленного доступа был установлен SSH-сервер рисунок 17.

– настройка файрвола. Рисунок 18.

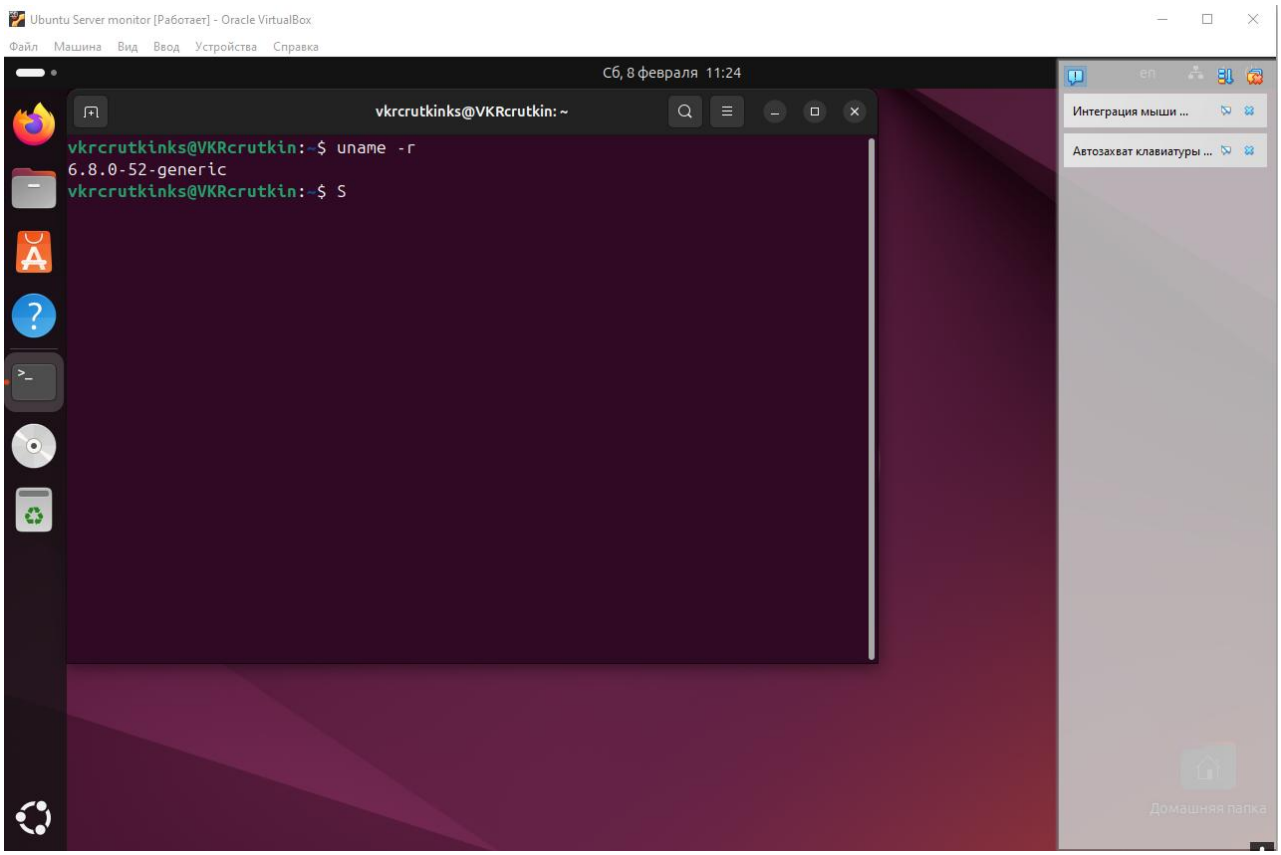

Рисунок 15– Установка операционной системы

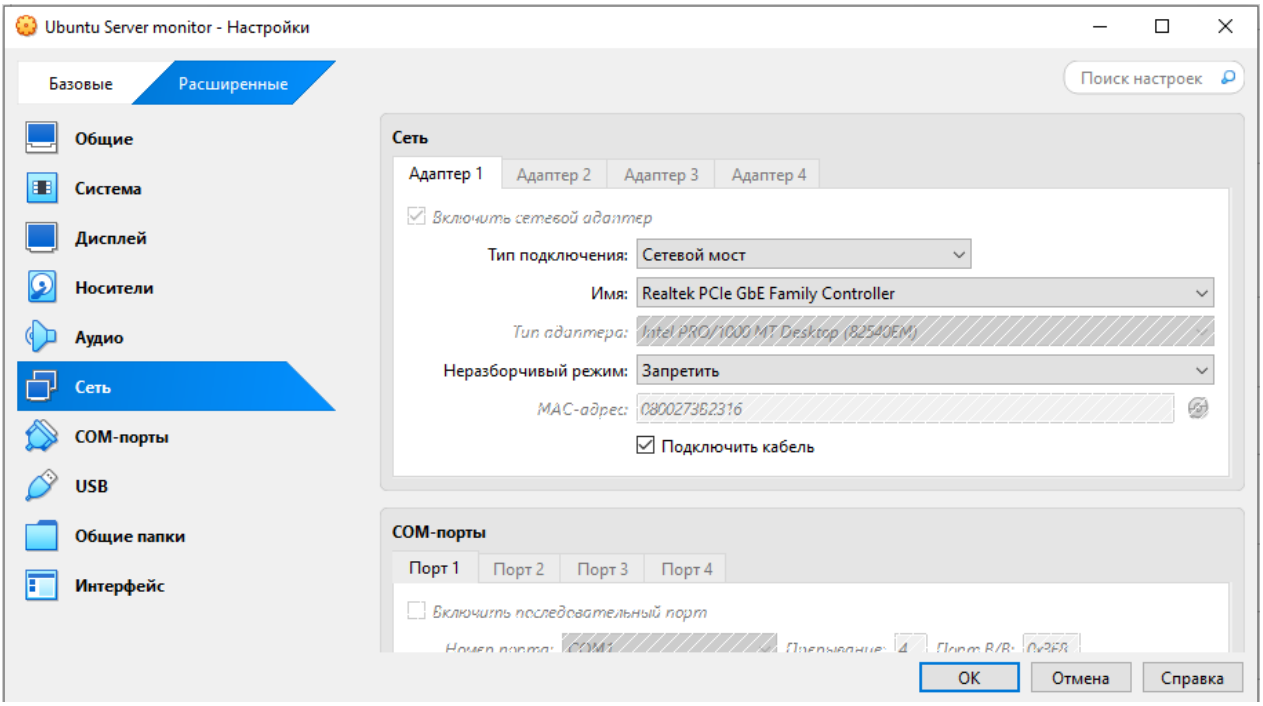

Рисунок 16 – Настройка сети

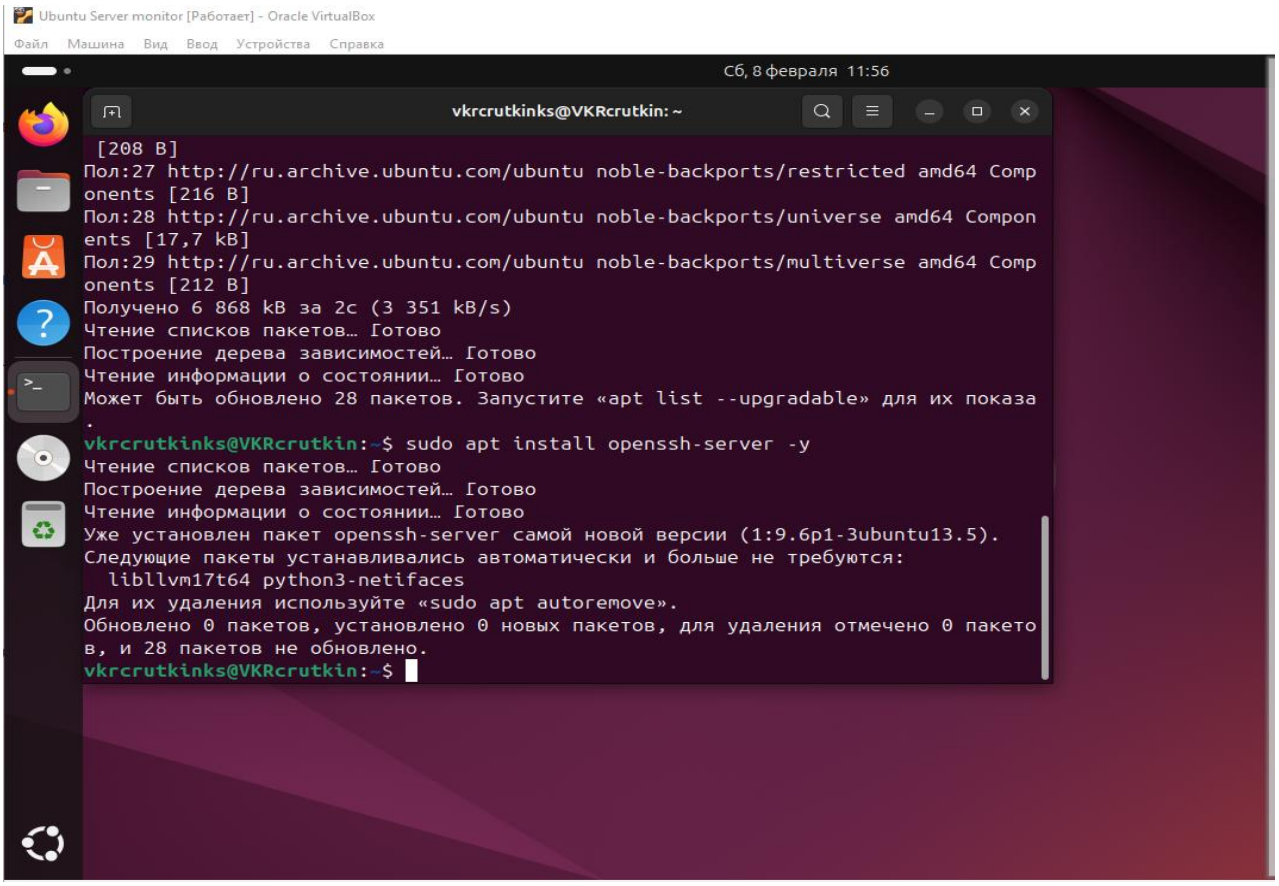

Рисунок 17 – Установка SSH сервера

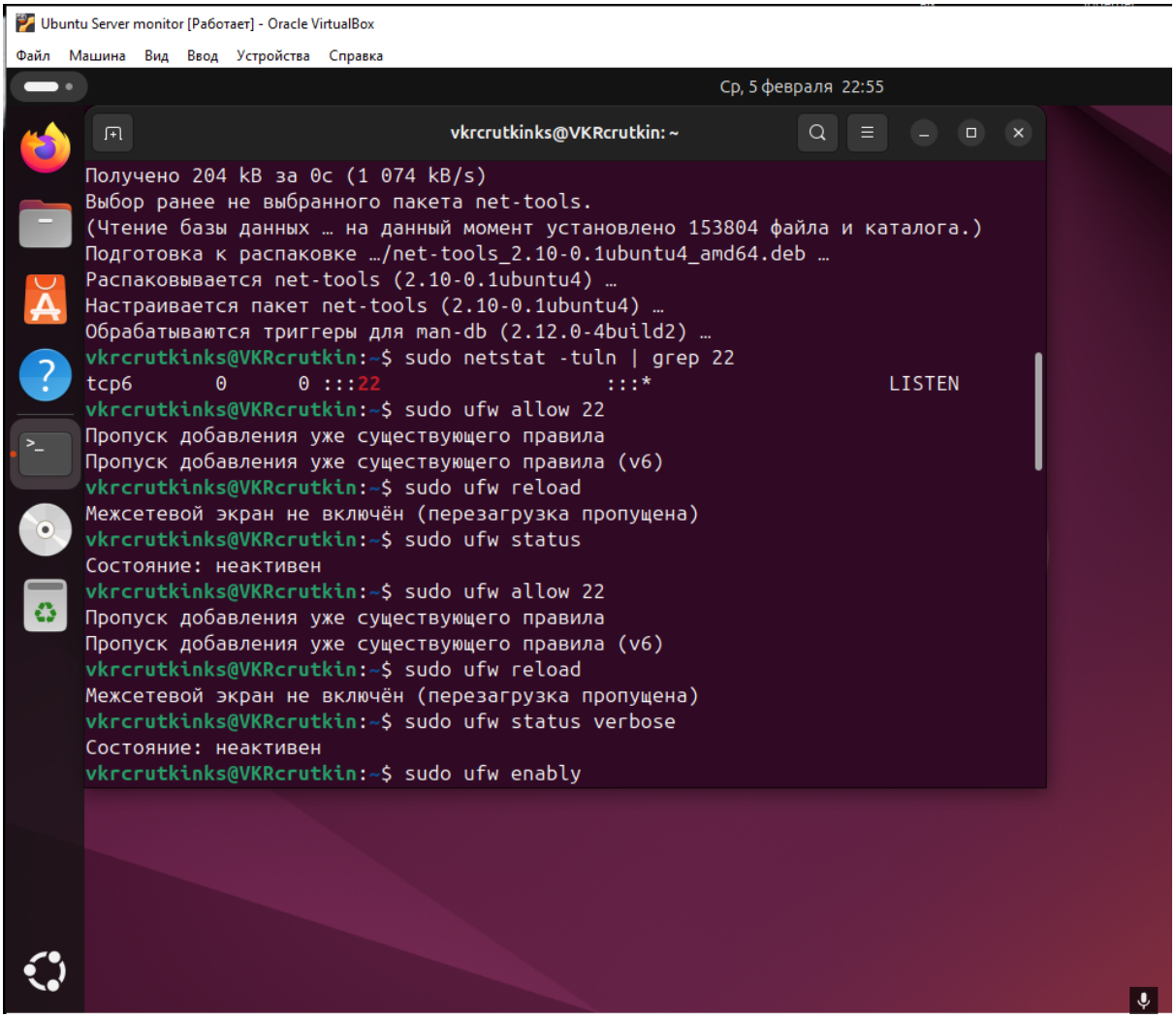

Рисунок 18 – Настройка файрвола

Таким образом, была создана полностью настроенная виртуальная машина, готовая к работе как удаленный сервер для мониторинга.

Для обеспечения удаленного доступа к ВМ был установлен SSH сервер. После установки была проверена работоспособность SSH через cmd рисунок 19.

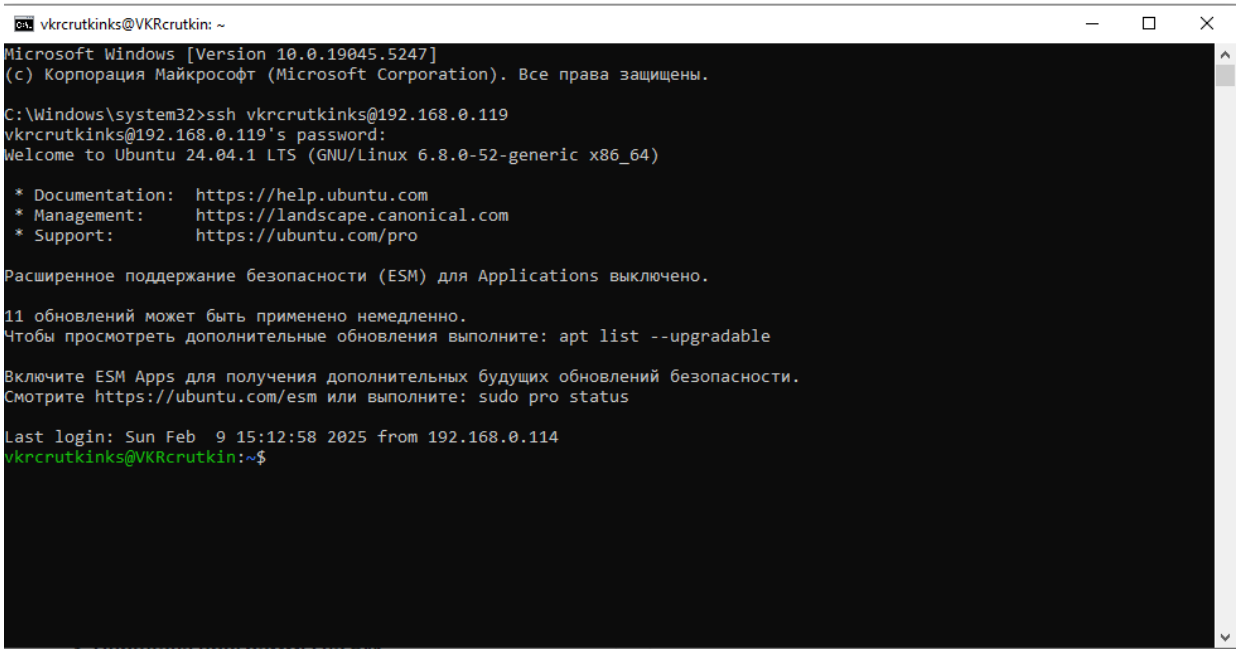

Рисунок 19 – Установка и проверка SSH сервера

Далее написанная команда на локальном компьютере была передана на виртуальную машину с помощью утилиты scp. Это позволило использовать её для мониторинга удаленного сервера. Вот команда для передачи файла рисунок 20.

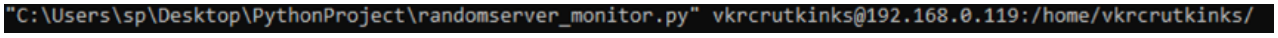

Рисунок 20 – Передача программы через SCP

Чтобы команда работала в ВМ необходимо было установить зависимости paramiko и matpolib. Так же было создано виртуальное окружение myenv из-за ошибки externally-managed-environment, которое не позволяло запускать программу рисунок 21.

```
vkrcrutkinks@VKRcrutkin:~$ python3 -m venv myenv
sovkrcrutkinks@VKRcrutkin:~$ source myenv/bin/activate
(myenv) vkrcrutkinks@VKRcrutkin:~$ pip list
Package         Version
--------------- -----------
bcrypt          4.2.1
cffi            1.17.1
contourpy       1.3.1
cryptography    44.0.0
cycler          0.12.1
fonttools       4.56.0
kiwisolver      1.4.8
matplotlib      3.10.0
numpy           2.2.2
packaging       24.2
paramiko        3.5.1
pillow          11.1.0
pip             25.0
pycparser       2.22
PyNaCl          1.5.0
pyparsing       3.2.1
python-dateutil 2.9.0.post0
six             1.17.0
(myenv) vkrcrutkinks@VKRcrutkin:~$
```

Рисунок 21 – Создание виртуального окружения

После создания всех условий для корректной работы программы в ВМ был произведен тест с искусственной нагрузкой на сервер рисунок 22.

```
(myenv) vkrcrutkinks@VKRcrutkin:~$ stress-ng --vm 1 --vm-bytes 50% --timeout 80s &
[5] 29633
(myenv) vkrcrutkinks@VKRcrutkin:~$ stress-ng: info:  [29633] setting to a 1 min, 20 secs run per stressor
stress-ng: info:  [29633] dispatching hogs: 1 vm

(myenv) vkrcrutkinks@VKRcrutkin:~$ stress-ng --cpu 2 --timeout 80s &
[6] 29636
(myenv) vkrcrutkinks@VKRcrutkin:~$ stress-ng: info:  [29636] setting to a 1 min, 20 secs run per stressor
stress-ng: info:  [29636] dispatching hogs: 2 cpu

(myenv) vkrcrutkinks@VKRcrutkin:~$ python randomserver_monitor.py
Successfully connected to 192.168.0.119
Raw Cpu Data: 57
Raw Memory Data: 32.06%
Cpu: 57%, Memory: 32.06%, Disk: 51%
График сохранен в server_metircs.png
Successfully connected to 192.168.0.119
Мониторинг сервера запущен...
Raw Cpu Data: 18
Raw Memory Data: 49.79%
Cpu: 18%, Memory: 49.79%, Disk: 51%
Raw Cpu Data: 30
Raw Memory Data: 52.98%
Cpu: 30%, Memory: 52.98%, Disk: 51%
Raw Cpu Data: 42
Raw Memory Data: 59.37%
Cpu: 42%, Memory: 59.37%, Disk: 51%
Raw Cpu Data: 42
Raw Memory Data: 59.37%
Cpu: 42%, Memory: 59.37%, Disk: 51%
```

Рисунок 22 – Тестирование программы с нагрузкой

Графики адекватно реагировали на изменения метрик, что подтвердило работоспособность программы рисунок 23.

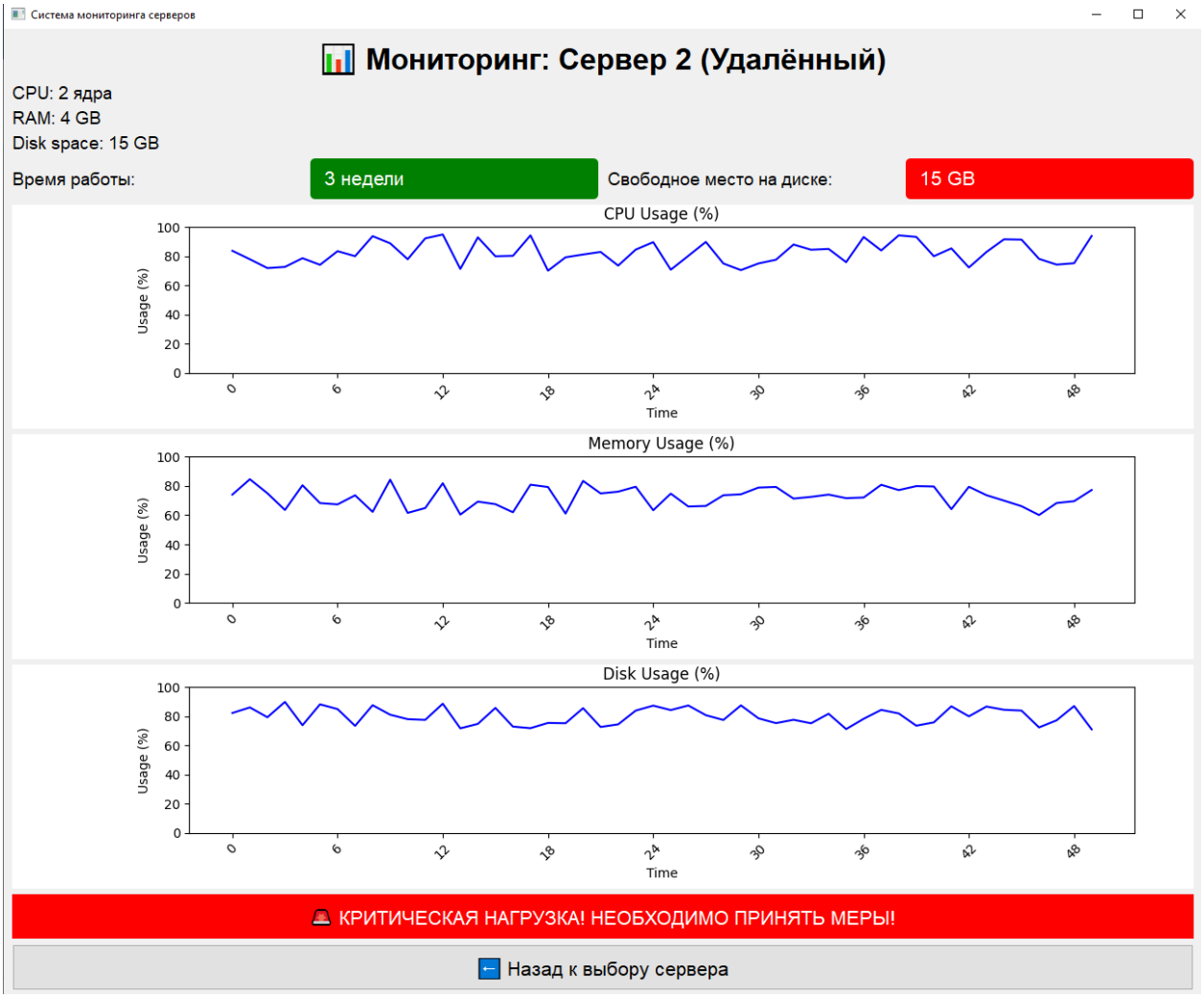

Рисунок 23 – Показатели после тестирования

Таким образом, используя разработанную программу мониторинга серверов, мне удалось успешно реализовать сбор и визуализацию ключевых метрик производительности (CPU, RAM, Disk). Были минимизированы сложности настройки SSH-подключения и сбора данных с удаленного сервера, что позволило сосредоточиться на корректности обработки информации и качестве графического представления. Это решение

обеспечивает не только базовый функционал мониторинга, но и гибкость для дальнейшего расширения возможностей системы.

Выводы по главе 3

В третьей главе была реализована система мониторинга серверов, соответствующая всем сформулированным ранее требованиям. Была разработана программа для локального и удаленного мониторинга серверов, основанная на использовании языка Python и библиотек psutil, paramiko и matplotlib. Реализация включала в себя этапы настройки среды разработки, установки необходимых зависимостей, сбора метрик производительности (загрузка CPU, использование RAM, состояние дискового пространства), а также их визуализации в виде графиков в реальном времени.

Для реализации удалённого взаимодействия с серверами в проекте было организовано подключение по протоколу SSH с использованием библиотеки Paramiko, что обеспечило безопасное выполнение команд на удалённой машине и позволило получать данные о текущем состоянии системы. В целях тестирования разработанного решения была создана виртуальная среда с использованием VirtualBox, что дало возможность моделировать эксплуатационные условия, максимально приближённые к рабочим. На базе виртуальной машины была развернута операционная система Ubuntu Server 24.04 LTS. Для корректной работы сетевого взаимодействия применён режим подключения "Bridged Adapter", а также выполнена установка и настройка SSH-сервера. Дополнительно был настроен брандмауэр UFW, что обеспечило базовую защиту среды.

Мониторинг состояния сервера осуществлялся путём запуска стандартных системных утилит, таких как top, free, df и других. Полученные данные преобразовывались в числовые значения, пригодные для дальнейшей визуализации. Для отображения метрик в графическом виде использовалась библиотека matplotlib, с помощью которой отслеживались ключевые параметры: загрузка центрального процессора, объём используемой оперативной памяти, занятость дискового пространства, а при наличии

соответствующего оборудования — также нагрузка на графический процессор (GPU).. Графики обновлялись каждую секунду благодаря функции FuncAnimation.

Тестирование программы показало её устойчивую работу как при низкой, так и при высокой нагрузке на сервер. Искусственная нагрузка, созданная с помощью утилит stress-ng и dd, не привела к сбоям в работе системы, а графиковая часть корректно отреагировала на изменение состояния сервера. Программа показала себя с лучшей стороны – она не только стабильно функционировала в процессе тестирования, но и обеспечивала своевременное обновление данных, а также корректно восстанавливалась после временных сетевых перебоев. Все это подтверждает её пригодность для внедрения в реальные IT-инфраструктуры.

Таким образом, разработанная система представляет собой практичный инструмент, который может быть использован как самостоятельное программное обеспечение или интегрирован в более широкую экосистему управления IT-ресурсами организации.

Заключение

В ходе выполнения выпускной квалификационной работы была разработана система мониторинга серверов, предназначенная для автоматизации процессов сбора, анализа и визуализации ключевых метрик производительности серверов. Работа охватывала все этапы жизненного цикла программного обеспечения – от анализа предметной области до реализации и тестирования.

В первой главе был проведен подробный анализ предметной области, изучены существующие решения для мониторинга серверов (Nagios, Zabbix, Prometheus), выявлены их преимущества и недостатки. На основании этого обоснована необходимость создания собственной системы, ориентированной на гибкость, простоту внедрения и минимизацию зависимости от сторонних инструментов. Также были определены ключевые метрики, подлежащие мониторингу: загрузка CPU, использование оперативной памяти и дискового пространства. Было проведено концептуальное моделирование системы, включая описание бизнес-процессов и формулировку функциональных требований.

Во второй главе были обоснованы и выбраны технологии, используемые при разработке. В качестве языка программирования был выбран Python благодаря своей читаемости, широкой библиотечной поддержке и активному сообществу. Для реализации SSH-подключения к удаленным серверам использовалась библиотека Paramiko, а для визуализации данных – Matplotlib. Для тестирования системы применялась виртуальная машина, созданная с помощью VirtualBox. Интегрированная среда разработки PyCharm была выбрана за свои мощные функции отладки, автодополнения и контроля версий. Были спроектированы UML-диаграммы: прецедентов, активности и развертывания, которые позволили формализовать поведение системы и её архитектуру. Также были сформулированы аппаратно-программные требования к системе,

обеспечивающие её надежное функционирование.

Третья глава была посвящена непосредственной разработке системы мониторинга. Была реализована программа для локального и удаленного мониторинга серверов, основанная на использовании Python и библиотек psutil, paramiko и matplotlib. Были реализованы модули сбора метрик, визуализации данных в виде графиков в реальном времени, а также механизм передачи команд через SSH-соединение. Система была протестирована на виртуальной машине с Ubuntu Server 24.04 LTS. Проведенные испытания показали высокую устойчивость программы к сетевым сбоям и нагрузкам на сервер, а также точность сбора данных. Графики, реализованные с использованием библиотеки Matplotlib, обеспечили наглядное представление информации, что повысило удобство использования системы.

Тестирование программы проводилось в различных условиях нагрузки с применением утилит stress-ng и dd. Результаты тестов подтвердили корректную работу системы как при минимальной, так и при высокой нагрузке на сервер. Программа демонстрировала стабильное поведение, своевременное обновление данных и корректное восстановление после временных сбоев сети. Это позволяет говорить о её готовности к практическому использованию в реальных IT-инфраструктурах.

Таким образом, в результате проделанной работы была создана полноценная система мониторинга серверов, позволяющая в режиме реального времени отслеживать состояние серверов и своевременно выявлять потенциальные проблемы. Использование современных технологий обеспечило высокую производительность, надежность и масштабируемость разработанного решения. Реализованная система может быть использована как самостоятельное решение или как часть более крупной системы управления ИТ-инфраструктурой.

Данная работа открывает возможность дальнейших исследований в области расширения функционала системы, добавления автоматических уведомлений, интеграции с популярными платформами мониторинга.

Список используемой литературы и используемых источников

1. Власов С.Л. Информационные технологии: основы и практика. – М.: Форум, 2020. – 432 с.

2. Гончаров И.П. Мониторинг и диагностика серверов: практическое руководство. – М.: РГГУ, 2021. – 210 с.

3. Гостев А.И. Методы защиты информации в компьютерных системах. – М.: БХВ-Петербург, 2022. – 304 с.

4. Григорьев С.А. Визуализация данных с использованием Python. – М.: Наука, 2021. – 288 с.

5. Гуреев, Р.М. Введение в разработку систем мониторинга [Текст]: научная статья. – Информационные технологии, 2020 – 5 с.

6. Евдокимов В.В. Системное программирование в Linux. – СПб.: БХВ-Петербург, 2020. – 416 с.

7. Ефремова, О.С. Требования охраны труда при работе на персональных электронно-вычислительных машинах (ПК) [Текст]: пособие. – М.: Лабиринт, 2019 – 60 с.

8. Медведев А.В. Информационные системы и технологии. – М.: Юрайт, 2022. – 345 с.

9. Нардин Д. Разработка и отладка программ на Python: практическое руководство. – М.: ДМК Пресс, 2021. – 320 с.

10. Негруца И.П. SSH-протокол: принципы безопасного удаленного доступа. – Харьков: СТАРТ, 2019. – 152 с.

11. Оськин В.В. Сетевые технологии и администрирование: учебное пособие. – М.: Академия, 2021. – 256 с.

12. Таненбаум Э.С., Уэзеролл Х. Современные операционные системы. – СПб.: Питер, 2020. – 1120 с.

13. Чесноков Д.А. Тестирование и отладка программного обеспечения. – СПб.: Питер, 2022. – 368 с.

14. Data Visualization with Python and JavaScript: Scrape, Clean, Explore, and Transform Your Data [Текст] / by Kyran Dale – No Starch Press, 2020 – 368 p.

15. Linux Performance Monitoring and Tuning [Текст]: учебное пособие / by Karthik Gurumurthy – Packt Publishing, 2019 – 384 p.

16. Matplotlib: A 2D Graphics Environment / by John D. Hunter – IEEE Computing in Science and Engineering, 2007 – 9 p.

17. Paramiko Documentation [Электронный ресурс] – Режим доступа: https://docs.paramiko.org/en/stable/ (дата обращения: 05.11.2023).

18. Stress Testing Tools for Linux [Электронный ресурс] – Режим доступа: https://wiki.ubuntu.com/StressTesting (дата обращения: 05.11.2023).

19. Unix Network Programming: The Sockets Networking API / by W. Richard Stevens – Addison-Wesley Professional, 2011 – 1000 p.

20. VirtualBox User Manual [Электронный ресурс] – Режим доступа: https://www.virtualbox.org/manual/UserManual.html (дата обращения: 05.11.2023).