

Разработка системы мониторинга серверов с использованием Python и Prometheus

Эрдман Александр Алексеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данном исследовании рассматривается процесс создания и внедрения системы мониторинга системных ресурсов компьютера. Приложение разработано с использованием программы Prometheus, языка программирования Python и библиотек psutil для сбора данных, tkinter для создания графического интерфейса. Результатом исследования является готовая система мониторинга, которая позволит отслеживать и анализировать состояние системы в реальном времени.

Ключевые слова: python, tkinter, psutil

Development of a server monitoring system using Python and Prometheus

Erdman Alexander Alekseevich

Sholom-Aleichem Priamursky State University

Student

Abstract

This study examines the process of creating and implementing a computer system resource monitoring system. The application is developed using the Prometheus program, the Python programming language and the psutil libraries for data collection, tkinter for creating a graphical interface. The result of the study is a ready-made monitoring system that will allow you to monitor and analyze the state of the system in real time.

Keywords: python, tkinter, psutil

1 Введение

1.1 Актуальность

В условиях современного развития информационных технологий и увеличения объема данных, эффективное управление системными ресурсами становится критически важным для обеспечения стабильной и высокопроизводительной работы серверов. Мониторинг системных ресурсов позволяет своевременно выявлять и устранять узкие места, оптимизировать использование ресурсов и предотвращать сбои в работе системы. В данном исследовании предлагается разработка и внедрение системы мониторинга, которая использует современные инструменты и технологии, такие как Python и Prometheus, для сбора и анализа метрик использования CPU, памяти

и дисков. Результаты исследования могут быть полезны для администраторов систем, разработчиков программного обеспечения и специалистов по информационной безопасности, стремящихся к повышению надежности и производительности своих серверов.

1.2 Обзор исследований

В.А. Рыжков в своём исследовании описал создание универсальной системы мониторинга API функций программных систем на основе объектной модели компонентов [1]. Д.И. Понамаренко и А.М. Ломакина рассмотрели применение языка программирования Python в автоматизации процесса сбора данных [2]. А.А. Новичихина посвятила статью анализу и описанию методов создания графического приложения на языке программирования Python с помощью библиотеки tkinter [3]. И.А. Баланов и Д.В. Кондратов изучили проблемы различного ПО, которое позволяет собирать диагностические данные о персональном компьютере [4]. Ю.В. Редькин, И.И. Бузенков и Е.М. Чернышева в совместном исследовании представили новый подход к повышению качества систем сбора и статистической обработки данных [5].

1.3 Цель исследования

Целью исследования является создание и описание программы для сбора системных данных о компьютере с помощью языка программирования Python и программного обеспечения Prometheus.

2 Материалы и методы

Для создания программы используется язык программирования Python, библиотеки tkinter, prometheus_client и psutil, а также ПО Prometheus. В качестве IDE используется PyCharm.

3 Результаты и обсуждения

Перед началом создания приложения необходимо скачать дистрибутив программы Prometheus. Данная программа позволяет осуществлять мониторинг событий и оповещения о них, а также предоставляет инструменты для их визуализации. Данная программа будет представлять результаты мониторинга в веб-интерфейсе, а графическое приложение на Python будет их собирать. Данный способ получения и графического отображения данных актуален и удобен для администраторов компьютеров на предприятиях, где имеется собственный сервер – просмотр данных о системе в реальном времени, а также отследить изменения в период работы сервера. Данный способ лучшего всего себя должен показывать в работе систем на базе Unix, где отсутствует графическая оболочка.

Установка вышеописанной программы на компьютер происходит извлечением файлов программы в любую папку на диске. После чего происходит настройка конфигурационного файла “prometheus.yml” (рис. 1).

```
scrape_configs:
  - job_name: 'python_app'
    static_configs:
      - targets: ['localhost:8000']
    # The job name is added as a
    # label `job=<job_name>` to any timeseries
    # scraped from this config.
  - job_name: "prometheus"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

static_configs:
  - targets: ["localhost:9090"]
```

Рисунок 1. Конфигурационный файл “prometheus.yml”

В данной конфигурации указывается название приложения, с которого будут получены показания, а также порт данной программы. В качестве порта для входа в веб-интерфейс программы Prometheus используется “9090”.

Запуск программы происходит путём консольной команды “./prometheus --config.file=prometheus.yml”, после чего можно войти в веб-интерфейс (рис. 2).

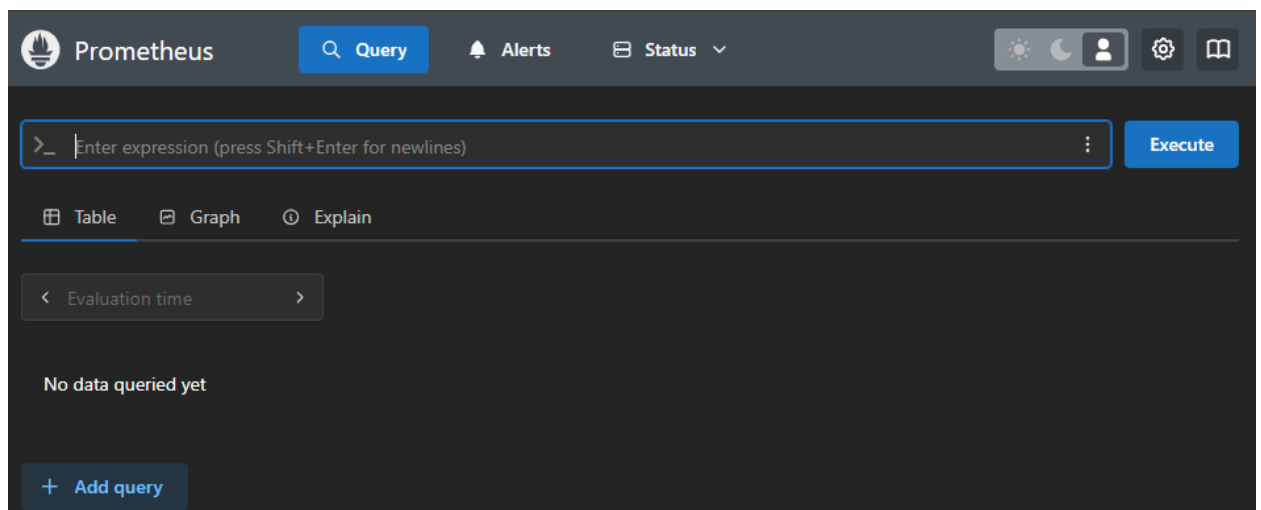


Рисунок 2. Веб-интерфейс программы Prometheus

Программа готова для работы. Следующим шагом является создание приложения. В начале проекта импортируются необходимые для работы библиотеки (рис. 3).

```
import tkinter as tk
from tkinter import ttk
import threading
import time
import random
from prometheus_client import start_http_server, Summary, Gauge
import psutil
import logging
```

Рисунок 3. Импорт библиотек

Как было сказано ранее, в этом блоке импортируются необходимые библиотеки: `tkinter` и `ttk` для создания графического интерфейса; `threading` для многопоточного выполнения задач; `time` и `random` для управления временем и генерации случайных чисел; `prometheus_client` для экспорта метрик; `psutil` для сбора информации о системе; `logging` для логирования ошибок и событий.

Настройка логирования осуществляется строчкой `“logging.basicConfig(level=logging.INFO, format='%(%asctime)s - %(levelname)s - %(message)s')”` для отслеживания работы программы и обнаружения ошибок.

Далее происходит создание метрик для экспорта в Prometheus (рис. 4).

```
REQUEST_TIME = Summary(name='request_processing_seconds', documentation='Время обработки запроса')
CPU_USAGE = Gauge(name='cpu_usage_percent', documentation='Процент использования CPU')
MEMORY_USAGE = Gauge(name='memory_usage_percent', documentation='Процент использования памяти')
DISK_USAGE = Gauge(name='disk_usage_percent', documentation='Процент использования диска')
```

Рисунок 4. Создание экспорта метрик

Затем осуществляется обновление метрических данных (рис. 5)

```
def process_request(t):
    time.sleep(t)

def update_metrics():
    while True:
        try:
            process_request(random.random())
            CPU_USAGE.set(psutil.cpu_percent(interval=1))
            MEMORY_USAGE.set(psutil.virtual_memory().percent)
            DISK_USAGE.set(psutil.disk_usage('/').percent)
            time.sleep(1)
        except Exception as e:
            logging.error(f"Ошибка обновления метрик: {e}")
```

Рисунок 5. Функция обновления метрик

Функция “process_request” используется для измерения времени обработки запросов. Функция “update_metrics” обновляет метрики каждую секунду и обрабатывает возможные ошибки. Данное решение позволяет отслеживать текущие значения использования CPU, памяти и диска.

После программируется обновление графического интерфейса (рис. 6). Функция “update_gui” обновляет графический интерфейс каждую секунду, отображая текущие значения использования CPU, памяти и диска. Это позволяет пользователям видеть данные нагрузки на системные ресурсы.

```
def update_gui():
    while True:
        try:
            cpu_usage = psutil.cpu_percent(interval=1)
            memory_usage = psutil.virtual_memory().percent
            disk_usage = psutil.disk_usage('/').percent

            cpu_label.config(text=f"Использование CPU: {cpu_usage}%")
            memory_label.config(text=f"Использование памяти: {memory_usage}%")
            disk_label.config(text=f"Использование диска: {disk_usage}%")

            update_process_list()
            update_disk_info()
            update_memory_info()

            time.sleep(1)
        except Exception as e:
            logging.error(f"Ошибка обновления графического интерфейса: {e}")
```

Рисунок 6. Функционал обновления графического интерфейса

Для отображения списка процессов создаётся функция “update_process_list” (рис. 7). Данная функция обновляет список процессов, сортированных по использованию CPU, и применяет фильтры (по имени процесса, по проценту нагрузки), что позволяет пользователям видеть, какие процессы наиболее нагружают систему.

Следующий программный код связан с дисками. В нём реализуется функция “update_disk_info”, которая обновляет информацию о дисках, обрабатывая ошибки для каждого диска отдельно. Это позволяет пользователям видеть, какие диски наиболее загружены. Метод “get_disk_usage_by_process” получает информацию о нагрузке на диск для каждого процесса, что в свою очередь позволит отслеживать, какие процессы наиболее активно используют диск. Для обновления информации о нагрузке на диск процессов, а также отображения её в таблице “disk_usage_tree”, используется функция “update_disk_usage_by_process” (рис. 8).

```

def update_process_list():
    try:
        processes = sorted(psutil.process_iter(['pid', 'name', 'cpu_percent']), key=lambda p: p.info['cpu_percent'], reverse=True)

        pid_filter = pid_entry.get()
        name_filter = name_entry.get()
        cpu_filter = cpu_entry.get()

        filtered_processes = []
        for proc in processes:
            if (not pid_filter or str(proc.info['pid']) == pid_filter) and \
                (not name_filter or name_filter.lower() in proc.info['name'].lower()) and \
                (not cpu_filter or float(cpu_filter) <= proc.info['cpu_percent']):
                filtered_processes.append(proc)

        process_tree.delete(*process_tree.get_children())

        for proc in filtered_processes[:10]:
            process_tree.insert(parent='', index='end', values=(proc.info['pid'], proc.info['name'], proc.info['cpu_percent']))
    except Exception as e:
        logging.error(f"Ошибка обновления списка процессов: {e}")

```

Рисунок 7. Функция обновления списка процессов

```

def update_disk_info():
    try:
        partitions = psutil.disk_partitions()
        disk_info = []
        for partition in partitions:
            try:
                usage = psutil.disk_usage(partition.mountpoint)
                disk_info.append((partition.device, partition.mountpoint, usage.total, usage.used, usage.free, usage.percent))
            except Exception as e:
                logging.error(f"Ошибка получения информации о диске {partition.device}: {e}")

        disk_tree.delete(*disk_tree.get_children())

        for disk in disk_info:
            disk_tree.insert(parent='', index='end', values=disk)
    except Exception as e:
        logging.error(f"Ошибка обновления информации о дисках: {e}")

def update_disk_usage_by_process():
    try:
        processes = psutil.process_iter(['pid', 'name'])

        disk_usage_tree.delete(*disk_usage_tree.get_children())

        for proc in processes:
            try:
                disk_usage = get_disk_usage_by_process(proc)
                disk_usage_tree.insert(parent='', index='end', values=(proc.info['pid'], proc.info['name'], disk_usage, disk_usage / 1024 / 1024))
            except Exception as e:
                logging.error(f"Ошибка получения информации о диске для процесса {proc.pid}: {e}")
    except Exception as e:
        logging.error(f"Ошибка обновления информации о диске для процессов: {e}")

def get_disk_usage_by_process(process):
    try:
        io_counters = process.io_counters()
        read_bytes = io_counters.read_bytes
        write_bytes = io_counters.write_bytes
        total_bytes = read_bytes + write_bytes
        return total_bytes
    except Exception as e:
        logging.error(f"Ошибка получения информации о диске для процесса {process.pid}: {e}")
        return 0

```

Рисунок 8. Функции сбора, обновления и отображения информации о дисках

Информация о памяти обновляется с помощью метода “update_memory_info” (рис. 9). В данном методе происходит получение данных о памяти и своп-памяти, а также отображает список процессов, сортированных по использованию памяти, что позволяет видеть, какие процессы наиболее нагружают память.

```
def update_memory_info():
    try:
        memory = psutil.virtual_memory()
        swap = psutil.swap_memory()

        total_memory_label.config(text=f"Всего памяти: {memory.total} байт")
        available_memory_label.config(text=f"Доступно памяти: {memory.available} байт")
        used_memory_label.config(text=f"Использовано памяти: {memory.used} байт")
        memory_percent_label.config(text=f"Использование памяти: {memory.percent}%")
        swap_total_label.config(text=f"Всего своп: {swap.total} байт")
        swap_used_label.config(text=f"Использовано своп: {swap.used} байт")
        swap_percent_label.config(text=f"Использование своп: {swap.percent}%")

        processes = sorted(psutil.process_iter(['pid', 'name', 'memory_percent']), key=lambda p: p.info['memory_percent'], reverse=True)

        memory_tree.delete(*memory_tree.get_children())

        for proc in processes[:10]:
            memory_tree.insert(parent='', index='end', values=(proc.info['pid'], proc.info['name'], proc.info['memory_percent']))
    except Exception as e:
        logging.error(f"Ошибка обновления информации о памяти: {e}")
```

Рисунок 9. Функция сбора данных об оперативной памяти

Следующим действием в ходе разработки приложения является оформление внешнего вида приложения с помощью библиотеки tkinter (рис. 10).

```
root = tk.Tk()
root.title("Монитор системы")

notebook = ttk.Notebook(root)
notebook.pack(expand=1, fill="both")

main_frame = ttk.Frame(notebook, padding="10")
main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

cpu_label = ttk.Label(main_frame, text="Использование CPU: 0%")
cpu_label.grid(row=0, column=0, sticky=tk.W)

memory_label = ttk.Label(main_frame, text="Использование памяти: 0%")
memory_label.grid(row=1, column=0, sticky=tk.W)

disk_label = ttk.Label(main_frame, text="Использование диска: 0%")
disk_label.grid(row=2, column=0, sticky=tk.W)

process_frame = ttk.Frame(notebook, padding="10")
process_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

pid_label = ttk.Label(process_frame, text="PID:")
pid_label.grid(row=0, column=0, sticky=tk.W)
pid_entry = ttk.Entry(process_frame)
pid_entry.grid(row=0, column=1, sticky=tk.W)

name_label = ttk.Label(process_frame, text="Имя:")
name_label.grid(row=0, column=2, sticky=tk.W)
name_entry = ttk.Entry(process_frame)
name_entry.grid(row=0, column=3, sticky=tk.W)

cpu_label = ttk.Label(process_frame, text="CPU %:")
cpu_label.grid(row=0, column=4, sticky=tk.W)
cpu_entry = ttk.Entry(process_frame)
cpu_entry.grid(row=0, column=5, sticky=tk.W)

process_tree = ttk.Treeview(process_frame, columns=('PID', 'Имя', 'CPU %'), show='headings')
process_tree.heading('PID', text='PID')
process_tree.heading('Имя', text='Имя')
process_tree.heading('CPU %', text='CPU %')
process_tree.grid(row=1, column=0, columnspan=6, sticky=(tk.W, tk.E, tk.N, tk.S))

disk_frame = ttk.Frame(notebook, padding="10")
disk_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

disk_tree = ttk.Treeview(disk_frame, columns=('Устройство', 'Точка монтирования', 'Всего', 'Использовано', 'Свободно', 'Процент'), show='headings')
disk_tree.heading('Устройство', text='Устройство')
disk_tree.heading('Точка монтирования', text='Точка монтирования')
disk_tree.heading('Всего', text='Всего')
disk_tree.heading('Использовано', text='Использовано')
disk_tree.heading('Свободно', text='Свободно')
disk_tree.heading('Процент', text='Процент')
disk_tree.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
```

Рисунок 10. Часть программного кода внешнего вида приложения (вкладки основных метрик и дисков)

Вкладки создаются для отображения основных метрик, процессов, информации о дисках, памяти и нагрузке на диск, что предоставляет возможность легко переключаться между различными видами информации.

После реализации главного функционала прописывается запуск HTTP-сервера для экспорта метрик в Prometheus, в который осуществляется вход через веб-интерфейс. После запускаются потоки для обновления метрик, графического интерфейса и информации о нагрузке на диск для процессов. В конце инициализируется запуск главного цикла графического интерфейса (рис. 11).

```
start_http_server(8000)
logging.info("Сервер метрик запущен на http://localhost:8000")

metrics_thread = threading.Thread(target=update_metrics)
metrics_thread.daemon = True
metrics_thread.start()

gui_thread = threading.Thread(target=update_gui)
gui_thread.daemon = True
gui_thread.start()

disk_usage_thread = threading.Thread(target=update_disk_usage_by_process)
disk_usage_thread.daemon = True
disk_usage_thread.start()

root.mainloop()
```

Рисунок 11. Функции запуска сервера, потоков для обновления метрик и главного цикла графического приложения

При запуске программы появится графическая оболочка приложения, где можно видеть получаемые данные о системе (рис. 12).

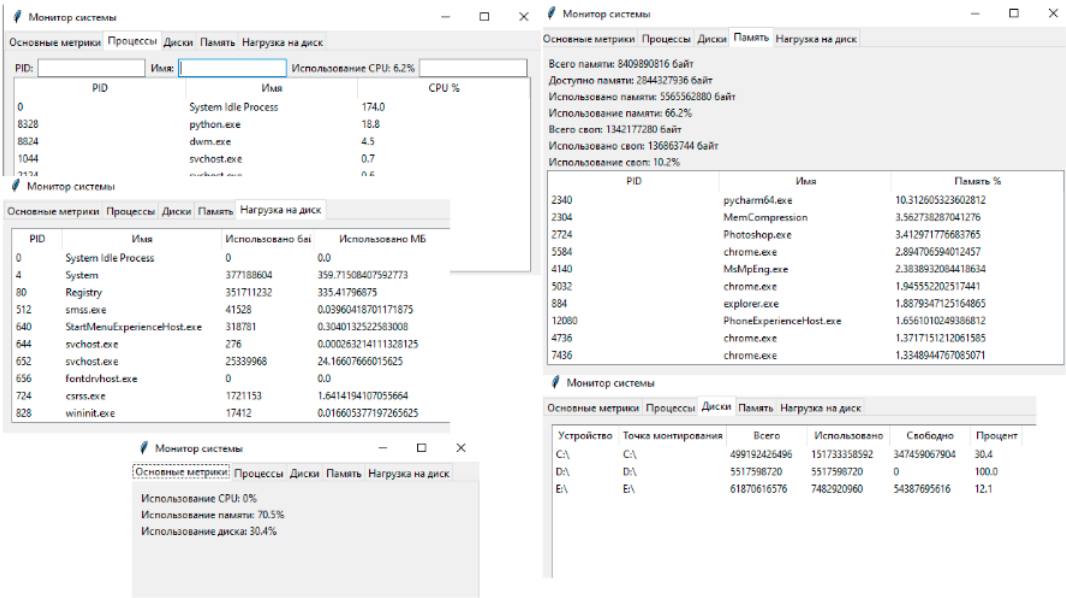


Рисунок 12. Результат работы приложения

Все полученные данные экспортируются в веб-интерфейс Prometheus, где их можно представить в виде графика, а также указать необходимые временные интервалы для более точного мониторинга (рис. 13).

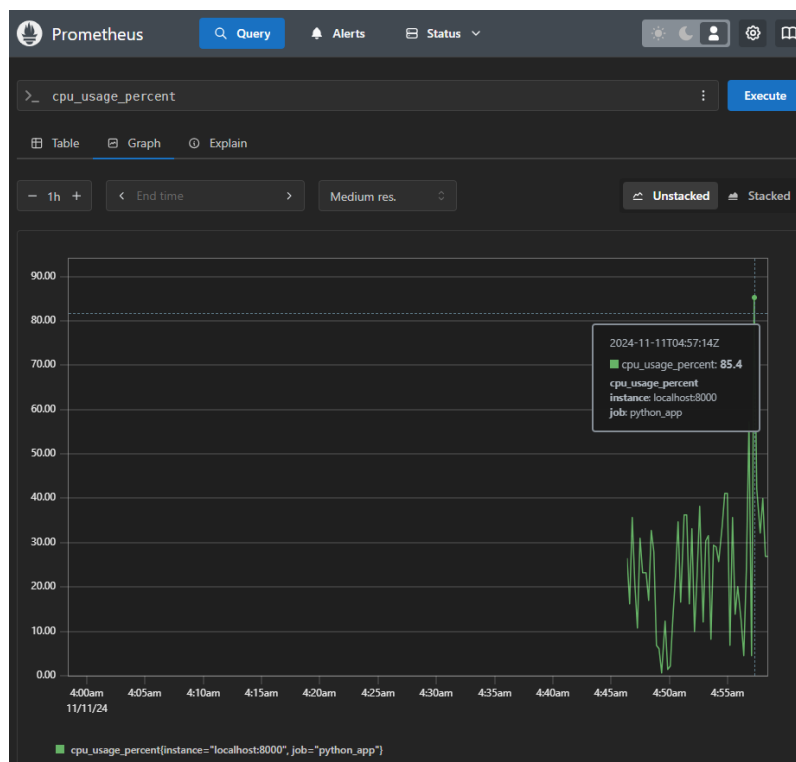


Рисунок 13. Импорт данных в веб-интерфейс Prometheus и визуализация данных в виде графика

Выводы

В результате исследования была разработана система для мониторинга ресурсов компьютера с возможностью транслирования полученных данных в программу Prometheus для дальнейшего визуализирования. Приложение написано на языке программирования Python с использованием библиотек “tkinter” и “psutil”.

Библиографический список

1. Рыжков В.А. Разработка универсальной системы мониторинга API функций программных систем на основе объектной модели компонентов // В сборнике: ИНФОРМАТИКА: ПРОБЛЕМЫ, МЕТОДОЛОГИЯ, ТЕХНОЛОГИИ. материалы XV международной научно-методической конференции. 2015. С. 367-370.
2. Понамаренко Д.И., Ломакина А.М. Применение языка Python в автоматизации сбора данных // Наука и образование транспорту. 2022. № 1. С. 333-336.
3. Новичихина А.А. Разработка простых приложений средствами tkinter в Python // Актуальные исследования. 2023. № 4-1 (134). С. 32-35.
4. Баланов И.А., Кондратов Д.В. Проблемы автоматического сбора

диагностических данных о персональном компьютере // Математическое моделирование, компьютерный и натурный эксперимент в естественных науках. 2023. № 2. С. 45-51.

5. Редькин Ю.В., Бузенков И.И., Чернышева Е.М. Решение задачи сбора и обработки данных в целях повышения качества контроля // В сборнике: Современные информационные технологии в управлении качеством. сборник статей VI Международной научно-прикладной конференции. 2018. С. 31-36.