Техническое задание: Разработка системы мониторинга Windows на основе WMI

1. Цель проекта

Разработать систему мониторинга для сбора и анализа данных о состоянии Windows-систем (персональных компьютеров и серверов) с использованием технологии Windows Management Instrumentation (WMI). Система состоит из облачного сервера и агентов, установленных на Windows-машинах. Агенты собирают метрики через WMI и передают их серверу по WebSocket. Сервер предоставляет API для доступа к данным и поддерживает реал-тайм мониторинг.

Проект включает два этапа:

- **MVP (Minimum Viable Product):** Реализация минимальной версии системы на языках Go и Java отдельно.
- **Полное решение:** Доработка MVP на одном из языков (Go или Java) для выполнения всех требований.

2. Общая архитектура

- **Агенты:** Программы, установленные на Windows-машинах, собирают метрики через WMI и отправляют их на облачный сервер по протоколу WebSocket. Агенты подключаются к серверу, используя адрес из конфигурационного файла.
- **Облачный сервер:** Принимает данные от агентов, хранит их, предоставляет REST API для получения информации о клиентах и метриках, а также поддерживает WebSocket для реал-тайм данных.
- **Взаимодействие:** Агенты регистрируются на сервере. В MVP регистрация автоматическая, в полном решении — с подтверждением через API. Сервер запрашивает метрики периодически или получает их по инициативе агента.
- **Хранение:** В MVP данные хранятся в памяти с ограничением по объёму. В полном решении метрики хранятся в колоночной базе данных, а информация о клиентах — в реляционной.

3. Требования к MVP

3.1. Функциональные требования

1. **Подключение агентов:**

- Агенты подключаются к серверу по WebSocket, используя URL из конфигурационного файла (например, `config.json` с полем `server_url`).
- При подключении агент отправляет уникальный идентификатор (UUID или hostname).
- Сервер сохраняет список подключённых агентов в памяти (map или список).

1. **Сбор метрик через WMI:**
   - Агенты собирают данные с локальной Windows-машины через WMI (namespace `root\CIMV2`).
   - **Информация о клиенте (статическая):**
     - Операционная система (например, `Win32_OperatingSystem.Caption`).
     - Модель CPU (`Win32_Processor.Name`).
     - Общий объём ОЗУ (`Win32_ComputerSystem.TotalPhysicalMemory`).
     - IP-адрес (`Win32_NetworkAdapterConfiguration.IPAddress`).
     - Uptime системы (`Win32_OperatingSystem.LastBootUpTime`).
     - Статус: connected/disconnected (определяется по активности WebSocket).
   - **Метрики (динамические):**
     - Нагрузка CPU (общая, `Win32_PerfFormattedData_PerfOS_Processor.PercentProcessorTime`).
     - Использование ОЗУ (% от общего, `Win32_OperatingSystem.FreePhysicalMemory`/`TotalVisibleMemorySize`).
     - Использование дисков (% заполненности, `Win32_LogicalDisk.FreeSpace`/`Size`).
   - Периодичность сбора: каждые 30 секунд (может изменяться в настройках).
1. **Передача данных:**
   - Агенты отправляют данные (информация о клиенте + метрики) на сервер по WebSocket в формате JSON.
   - Пример структуры:{
   - "client_id": "hostname-or-uuid",
   - "timestamp": "2025-09-29T17:00:00Z",
   - "client_info": {
   - "os": "Windows 10 Pro",
   - "cpu": "Intel Core i7",
   - "ram_total": 16384,
   - "ip": "192.168.1.100",
   - "uptime": 3600,
   - "status": "connected"
   - },
   - "metrics": {

```
    "cpu_load": 25.5,
    "ram_usage": 60.2,
    "disk_usage": 75.0
  }
}
```

1. **Хранение на сервере:**
   - Сервер хранит в памяти:
     - Список клиентов (client_id, client_info).
     - Метрики (client_id, timestamp, metrics) за последние 10 минут (ограничение: до 100 записей на клиента).
   - Старые метрики автоматически удаляются.
1. **API сервера:**
   - **GET /clients**: Возвращает список всех подключённых клиентов (client_id, client_info).
   - **GET /metrics/:client_id?start_time=...&end_time=...**: Возвращает метрики клиента за указанный период (из памяти).
   - **GET /metrics/:client_id/latest**: Возвращает последние метрики клиента.

3.2. Нефункциональные требования

- **Языки реализации:** Отдельные реализации на Go и Java.
  - Go: Использовать `github.com/go-ole/go-ole или github.com/microsoft/wmi` для WMI, `github.com/gorilla/websocket` для WebSocket.
  - Java: Использовать `com.github.wshackle/jacob` или `org.jinterop` для WMI, `org.springframework.boot:spring-boot-starter-websocket` для WebSocket или другие подходящие библиотеки на усмотрение разработчика.
- **Платформа:** Агенты работают только на Windows (напр. Windows 10/11 или Server 2019/2022). Сервер — любой ОС (Linux/Windows для облака).
- **Безопасность:** В MVP без аутентификации (доверительная сеть). WebSocket без TLS (http://).
- **Логирование:** Базовые логи (подключение/отключение агентов, ошибки WMI).
- **Тестирование:** Локально на 1-2 агентах, сервер в Docker или на localhost.

4. Требования к полному решению

Полное решение дорабатывается на одном из языков (Go или Java) и включает весь функционал MVP плюс следующие дополнения:

4.1. Дополнительные функциональные требования

1. **Регистрация и подтверждение клиентов:**

- При подключении агент регистрируется как "неподтверждённый" (сохраняется в БД).
- Сервер предоставляет API:
  - **POST /clients/:client_id/confirm**: Переводит клиента в статус "зарегистрирован".
- Только с зарегистрированных клиентов принимаются и сохраняются метрики.
- Статусы клиента: `pending`, `registered`, `disconnected`.

1. **Хранение данных:**
   - **Клиенты:** Хранятся в реляционной БД (напр. PostgreSQL).
     - Таблица: `clients(client_id, status, client_info, registered_at)`.
   - **Метрики:** Хранятся в колоночной БД (напр. InfluxDB или ClickHouse).
     - Структура: `client_id`, `timestamp`, `cpu_load`, `ram_usage`, `disk_usage`, etc.
     - Время хранения: Настраивается (по умолчанию 7 дней, удаление старых записей).
   - Сервер очищает устаревшие данные.
2. **Расширенные метрики:**
   - Добавить к MVP:
     - Нагрузка CPU по ядрам (`Win32_PerfFormattedData_PerfOS_Processor` с `Name != _Total`).
     - Температура CPU/GPU (если доступно через WMI, например, `MSAcpi_ThermalZoneTemperature` или third-party providers).
     - Нагрузка на сеть (`Win32_PerfFormattedData_Tcpip_NetworkInterface.BytesTotalPerSec`).
     - I/O диска (`Win32_PerfFormattedData_PerfDisk_LogicalDisk.DiskBytesPerSec`).
     - Статус здоровья: `healthy`/`unhealthy` (на основе пороговых значений, например, CPU > 90% = unhealthy).
3. **Реал-тайм через WebSocket:**
   - Сервер предоставляет WebSocket-эндпоинт (`/ws/metrics/:client_id`) для подписки на реал-тайм метрики.
   - Формат сообщений аналогичен MVP (JSON с метриками).
4. **Настройка мониторинга (опционально):**
   - API для настройки:
     - **PATCH /clients/:client_id/config**: Установить периодичность сбора метрик (например, 10-60 секунд).

        - **PATCH /metrics/retention**: Установить время хранения метрик (1-30 дней).
    - Агенты читают настройки при подключении или получают через WebSocket.

4. Ожидаемые результаты

- **MVP (Go и Java):**
    - Рабочие агенты, собирающие базовые метрики через WMI и отправляющие по WebSocket.
    - Сервер с REST API для списка клиентов и метрик (latest + за период).
    - Хранение в памяти, логирование ошибок.
    - Документация: краткое описание API и запуск (README.md).
- **Полное решение:**
    - Полный функционал (регистрация, БД, расширенные метрики, WebSocket реал-тайм).
    - Настраиваемая периодичность и хранение (опционально).
    - Документация: полное описание API, БД-схемы, примеры запросов.

5. Ограничения

- Агенты работают только на Windows (WMI-ограничение).
- В MVP нет отказоустойчивости (например, reconnect агентов — в полном решении).
- Для температуры CPU/GPU WMI может зависеть от оборудования (если недоступно, указать в документации).

6. Технологический стек

- **Go:**
    - WMI: `github.com/go-ole/go-ole` или `github.com/yusufpapurcu/wmi` или другие на выбор.
    - WebSocket: `github.com/gorilla/websocket`.
    - HTTP: `github.com/gin-gonic/gin`.
    - Полное: PostgreSQL (`github.com/lib/pq`), InfluxDB (`github.com/influxdata/influxdb-client-go`).
- **Java:**
    - WMI: `com.github.wshackle/jacob` или `org.jinterop`.
    - WebSocket: `org.springframework.boot:spring-boot-starter-websocket`.
    - HTTP: Spring Boot (`org.springframework.boot:spring-boot-starter-web`).
    - Полное: PostgreSQL (JDBC), InfluxDB (`com.influxdb:influxdb-client-java`).

- **БД (полное):**
  - Реляционная: PostgreSQL.
  - Колоночная: InfluxDB или ClickHouse.
- **Протоколы:** WebSocket (ws:// в MVP, wss:// в полном), REST (HTTP/HTTPS).

Конечный выбор остаётся за разработчиком и может измениться в процессе разработки.

8. Критерии приемки

- **MVP:**
  - Агент подключается, отправляет данные не реже чем раз в 30 сек.
  - Сервер возвращает список клиентов и метрики через API.
  - Данные корректны (проверено вручную через `Get-WmiObject` в PowerShell).
- **Полное решение:**
  - Подтверждение регистрации через API.
  - Метрики хранятся в колоночной базе, клиенты — в PostgreSQL.
  - Реал-тайм работает через WebSocket.
  - Настройки применяются через API.