

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Інститут інформатики та радіоелектроніки
Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти (освітній ступінь))

на тему СИСТЕМА МОНИТОРИНГУ СЕРВЕРНОГО ОБЛАДНАННЯ І
ЗАСТОСУНКІВ

Виконав: студент 2 курсу, групи КНТз-512м
спеціальності _____

123 «Комп'ютерна інженерія»

Освітня програма (спеціалізація)

«Комп'ютерні системи та мережі»

(код і назва спеціальності)

ХІЛЬКО К. В.

(прізвище та ініціали)

Керівник ГРУШКО С.С.

(прізвище та ініціали)

Рецензент СТЕПАНЕНКО О.О.

(прізвище та ініціали)

м. Запоріжжя
2023 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій
 Кафедра «Комп'ютерні системи та мережі»
 Ступінь вищої освіти (освітній ступінь) магістерський
 Спеціальність 123 Комп'ютерна інженерія
 (код і назва)
 Освітня програма (спеціалізація) Комп'ютерні системи та мережі
 (назва)

ЗАТВЕРДЖУЮ

Зав. кафедри Кудерметов Р.К.

“15” листопада 2023 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

ХІЛЬКО Катерина Володимирівна

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Система моніторингу серверного обладнання і застосунків

керівник проекту (роботи) к. т. н., доцент, Грушко Світлана Сергіївна

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від “24” жовтня 2023 року № 400

2. Строк подання студентом проекту (роботи) 01 грудня 2023 року

3. Вихідні дані до проекту (роботи) датчики, база даних mysql, серверне обладнання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) _____

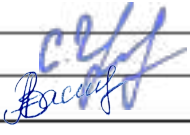
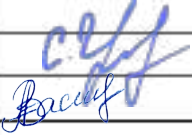
1) Аналіз існуючих систем

2) Розробка системи моніторингу.

3) Практична реалізація.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проекту (роботи)

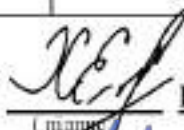
Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	ГРУШКО С.С., к. т. н., доцент		
Нормоконтроль	ПОЛЬСЬСКА О.В., ст. викл.		

7. Дата видачі завдання 01.10.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз технічного завдання	10.10.2023 р.	
2	Модель управління потоками пацієнтів у черзі на обслуговування.	15.10.2023 р.	
3	Проектування бази даних	01.10.2023 р.	
4	Побудова логічної моделі бази даних	25.11.2023 р.	
5	Проектування програмних засобів	01.12.2023 р.	
6	Оформлення отриманих результатів у ПЗ	10.11.2023 р.	
7	Оформлення графічного матеріалу	15.11.2023 р.	
8	Захист роботи	16.12.2023 р.	

Студент

Катерина ХІЛЬКО

(ініціали та прізвище)

Керівник проекту (роботи)

Світлана ГРУШКО

(ініціали та прізвище)

РЕФЕРАТ

ПЗ: 76 с., 18 рис., 9 табл., 18 джерел.

МОНІТОРІНГ, МОДЕЛЬ, МЕРЕЖІ ПЕТРІ, СИСТЕМА

Метою дослідження є розробка та реалізація проекту моніторингу служб та інфраструктури.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- розробка загального переліку вимог до моніторингу;
- дослідження існуючих рішень та обґрунтований вибір основи майбутнього моніторингу;
- розробка списку конкретизованих вимог до майбутнього моніторингу;
- практична реалізація моніторингу;
- тестування розробленого додатку.

У першій частині розглядаються основні існуючі рішення для моніторингу, виявляються їх позитивні та негативні сторони, у тому числі з погляду конкретних вимог певних проектів.

У другій частині роботи описуються методи створення власного моніторингу з урахуванням обраної системи, аргументується її вибір.

У третій частині роботи докладно розглянуто практичну реалізацію найважливіших частин системи: базові скриптові алгоритми стеження станом, використання сторонніх бібліотек.

В останній частині описані механізми та наслідки впровадження розробленої системи моніторингу у реальні функціонуючі проекти компанії.

У висновку підбиваються підсумки про досягнуті під час виконання роботи результати.

ABSTRACT

Explanatory note to the master's work: 76 p., 18 figures, 9 tables, 18 sources.

MONITORING, MODEL, PETRI NETS, SYSTEM

The aim of the study is to develop and implement a project for monitoring services and infrastructure.

To achieve this goal, the following tasks need to be solved:

- development of a general list of monitoring requirements;
- researching existing solutions and making a reasonable choice of the basis for future monitoring;
- development of a list of specific requirements for future monitoring;
- practical implementation of monitoring;
- testing of the developed application.

The first part discusses the main existing monitoring solutions, identifying their positive and negative aspects, including in terms of the specific requirements of certain projects.

The second part of the paper describes the methods of creating your own monitoring, taking into account the chosen system, and provides arguments for its choice.

The third part of the paper discusses in detail the practical implementation of the most important parts of the system: basic scripting algorithms for monitoring the state of the system, and the use of third-party libraries.

The last part describes the mechanisms and consequences of implementing the developed monitoring system in real operating projects of the company.

The conclusion summarises the results achieved in the course of the work.

ЗМІСТ

Вступ.....	7
1 Аналіз існуючих систем	9
1.1 Функції системи моніторингу	9
1.2 Огляд Microsoft SCOM закладу	12
1.3 Огляд Zabbix	15
1.4 Огляд Nagios	20
1.5 Огляд Cacti	22
1.6 Порівняльний аналіз вільно поширюваних систем	25
2 Розробка системи моніторингу	26
2.1 Основні завдання	26
2.2 Вимоги до моніторингу	29
2.3 Загальні відомості про побудову моніторингу	30
2.4 Загальна структура скриптів. Спеціальні параметри	36
2.5 Відомості про додаткові модулі	37
3 Практична реалізація	48
3.1 Основні показники	48
3.2 Спеціалізовані показники. Підключення бібліотек	59
3.3 Інвентаризація обладнання	65
3.4 Результат впровадження моніторингу	68
Висновки	72
Перелік джерел посилання	75

ВСТУП

Моніторинг в інформаційній структурі, чи то маленька компанія, чи величезний дата-центр, потрібен, щоб системні адміністратори були сповіщені про поломки та проблеми в інфраструктурі раніше чи хоча б одночасно з користувачами. Необхідність прогнозування, а тим самим і запобігання, поломок, оповіщення про них та зберігання інформації про стан систем у будь-якій ІТ системі забезпечує актуальність даної роботи. По суті, моніторинг - це комплекс швидкого знаходження проблеми, оповіщення про неї адміністраторів, а також діагностики, що дає повну і точну інформацію про поломку.

Об'єктом дослідження є різні ІТ інфраструктури, предметом дослідження – методи моніторингу цих інфраструктур.

Мета роботи – створення унікальної системи моніторингу, що повністю відповідає вимогам конкретного проекту компанії, з мінімальними витратами.

Завдання, які вирішуються у зв'язку із зазначеною метою:

- аналіз переваг та недоліків існуючих систем моніторингу;
- збір вимог до системи, що розробляється;
- пошук оптимального шляху реалізації системи

Практична значущість та новизна роботи пояснюється неможливістю стеження за деякими важливими показниками працездатності сервісів та серверів за допомогою стандартних засобів моніторингу, або високою трудомісткістю таких методів. Створений моніторинг із мінімальними витратами та невисокою трудомісткістю вирішує поставлені завдання.

Введення розкриває проблему роботи, зумовлює її актуальність (практичну значущість роботи), описує об'єкт і предмет дослідження, і навіть розповідає і мети і поставлених шляху до неї завданнях.

У першій частині розглядаються основні існуючі рішення для моніторингу, виявляються їх позитивні та негативні сторони, у тому числі з погляду конкретних вимог певних проектів.

У другій частині роботи описуються методи створення власного моніторингу з урахуванням обраної системи, аргументується її вибір.

У третій частині роботи докладно розглянуто практичну реалізацію найважливіших частин системи: базові скриптові алгоритми стеження станом, використання сторонніх бібліотек.

В останній частині описані механізми та наслідки впровадження розробленої системи моніторингу у реальні функціонуючі проекти компанії.

У висновку підбиваються підсумки про досягнуті під час виконання роботи результати.

1 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ

1.1 Функції системи моніторингу

Основним завданням системи моніторингу є надання актуальної інформації для аналізу стану IT-інфраструктури та швидкого виявлення виниклої несправності та її оперативне усунення. Системи моніторингу продуктивності дозволяють IT-фахівцям вчасно помітити зниження продуктивності та визначити "вузькі місця" в IT-інфраструктурі. Постійний моніторинг допомагає уникнути простоїв у її роботі, підтримувати всі IT-сервіси у робочому стані та зберігати необхідний рівень їхньої якості, а також спланувати її модернізацію.

Раніше роль моніторингу здійснювали адміністратори, а інформація про стан систем у кращому разі збиралася ними ж у будь-яких неспеціалізованих програмах (через їхню відсутність), у гіршому взагалі ніяк не накопичувалася і не агрегувалася. Усі відомості про систему були прив'язані до практичного досвіду роботи з інфраструктурою у конкретного спеціаліста і повністю губилися під час його відходу.

Зараз з'явилося безліч напів-і повністю автоматизованих систем для моніторингу, які аналізують стан систем, збирають інформацію в колекції, які також можна вивчити при необхідності.

Існують досить специфічні види моніторингу, наприклад, від імені кінцевого користувача, коли задані проміжки часу циклічно емулюються його дії. Зазвичай це робот, планувальник завдань, що запускає спеціальний, заздалегідь визначений скрипт-сценарій, а потім рапортає про успіх виконання дій або про помилки, що виникли в процесі.

Для зберігання отриманої інформації зазвичай використовується база даних конфігурації під різними СКБД: інформація про об'єкти моніторингу представлена, як набір конфігураційних одиниць. Кожен сервер, кожен мережевий пристрій — це одиниця, усе це зберігається у централізованій базі даних. Таке уявлення дозволяє

потім інтегрувати систему моніторингу з візуальними уявленнями: діаграмами, графіками та ін.

Сама структура моніторингу значно видозмінюється з часом. Наприклад, одна з тонкощів виникла при появі та великому поширенні віртуалізації: якщо раніше була необхідність відслідковувати стан лише фізичних серверів, то тепер на кожному з них може бути ще кілька віртуальних.

Також системи моніторингу можна налаштувати виконання будь-яких стандартних сервісних дій. Наприклад, очищати кошик під час його заповнення або активувати архівування для будь-яких файлів, коли певний відсоток дискового простору стає зайнятим.

При виборі, розробці, впровадженні систем моніторингу спочатку необхідно визначитися з об'єктами, які будуть підлягати спостереженню, а також критичні події та показники, які визначають кількість оповіщень при поломці, частоту сканування та інші параметри та наслідки. Для великих інфраструктур, на кшталт дата-центрів, перед фінальним використанням зазвичай розгортають тестовий майданчик, де можна оцінити доцільність зроблених рішень та визначення параметрів порогових значень.

Впровадження подібних рішень особливо важливе при використанні сервісного підходу до діяльності ІТ-підрозділів, коли всі процеси переглядаються з точки зору ІТ-сервісів, що надаються підрозділом. Кожен бізнес-сервіс корпоративної системи, по можливості, інтерпретується як ІТ-сервіс, задається певний рівень якості його надання. Далі він описується у системі моніторингу як набір взаємозалежних компонентів ІТ-інфраструктури.

У результаті формується Угода про рівень якості сервісів (Service Level Agreement, SLA). Відповідно до SLA система здійснює збирання та зберігання інформації про якість надання ІТ-сервісів. На основі накопиченої інформації формуються звіти за певний період. Аналіз звітної інформації допомагає здійснювати:

- перегляд рівня надання ІТ-сервісів;
- реорганізацію діяльності ІТ-підрозділу;

- модернізацію IT-інфраструктури.

Системи моніторингу можуть бути спрямовані на споживачів різного рівня. Для великих систем зазвичай використовується величезна кількість різноманітних функцій, для маленьких зазвичай досить загального аналізу вузлів та надсилання оповіщень. Серед основних функцій моніторингів можна виділити такі:

Спостереження. Основна функція, що включає періодичний збір показників з вузлів обладнання, сервісів і т.п.

Зберігання інформації. Доповнення до стеження. Здійснюється збір інформації за основними показниками кожного об'єкта моніторингу, для зберігання зазвичай використовуються бази даних.

Побудова звітів. Здійснюється як з урахуванням поточних даних стеження, і за довгостроково збереженої інформації. Наприклад, довгостроковий моніторинг навантаження на сервер може попередити, що споживані ресурси постійно збільшуються, отже необхідно збільшити доступні кошти або перенести частину завдань на інший сервер, вибір якого також можна здійснити на основі довгострокового звіту.

Візуалізація. Звіти у візуальному поданні: у вигляді графіків, підказок, діаграм. Допомагають легкому сприйняттю інформації, а також можливий вибір для візуалізації кількох найважливіших індикаторів, тоді як у звітах будуть представлені всі показники.

Пошук вузьких місць. На основі аналітичних даних моніторингу можна дізнатися, в якому місці інфраструктури найбільше знижує загальні показники продуктивності.

Автоматизація сценаріїв. Функція звільняє адміністраторів від рутинних завдань.

Завдяки наявності коштів для реалізації всіх цих функцій адміністратору більше не потрібно перевіряти вручну стан кожної складової системи, проблеми вирішуються та уламки усуваються оперативніше, діагностика здійснюється багатомірно і точно, а також можна планувати розширення інфраструктури.

Використання систем моніторингу та управління дозволяє:

- оптимізувати використання інформаційних ресурсів;
- підвищити якість IT-сервісів та швидкість усунення збоїв у роботі обладнання та програмного забезпечення, мінімізувати час простою сервісів;
- забезпечити надійність, безпеку та узгоджене функціонування всіх компонентів IT-інфраструктури;
- полегшити модернізацію IT-інфраструктури;
- у кілька разів підвищити ефективність роботи IT-підрозділу.

1.2 Огляд Microsoft SCOM закладу

System Center Operations Manager – система наскрізного моніторингу від Microsoft, в тому числі активного стеження за станом мереж (спостереження за будь-якими мережевими пристроями, що підтримують SNMP, аж до рівня портів, а також виявлення віртуальних локальних мереж та комутаторів у таких мережах). В останніх версіях з'явилася можливість стеження не лише за системами, під керуванням операційних систем сімейства Windows, а й за гетерогенними середовищами, що включають UNIX та Linux. System Center Operations Manager призначений головним чином для організацій з числом машин понад 500 та числом серверів більше 30. Для менших організацій існує продукт System Center Essentials, що включає частину функціоналу продуктів System Center Operations Manager і System Center Configuration Manager, але призначений для малих і середніх підприємств.

Сам продукт, починаючи з версії 2012 року, є сервісом високої доступності завдяки відсутності серверів керування. У пулі з кількома серверами навантаження балансується та забезпечується доступність. На кожному сервері працює

конфігураційна служба, причому зберігання даних реалізовано не в пам'яті або XML-файлах, а в базі даних.

Microsoft також надає можливість інтеграції продукту із System Center Service Manager, завдяки чому з'являється можливість автоматичного створення інцидентів на основі оповіщень SCOM.

Що стосується тонкого стеження за віртуальними середовищами, є засоби для інтеграції з пакетом System Center Virtual Machine Manager, який буде передавати System Center Operations Manager інформацію про віртуальні машини, служби, приватні хмари та вузли.

- виняткова продуктивність та працездатність додатків для програмних середовищ Microsoft;
- забезпечує наскрізне керування службами для сервісів вашого центру обробки даних;
- сприяє поліпшенню ефективності та керування середовищами центрів обробки даних;
- уніфікований контроль у рамках приватних та загальнодоступних хмарних сервісів.
- підтримка Windows PowerShell 2.0 із набором нових командлетів.

Одна з головних переваг System Center Operations Manager - просунута візуалізація всього величезного зібраного набору даних, в основному у вигляді графіків і діаграм, причому візуалізація доступна не лише у спеціальній консолі програми, а й через веб-інтерфейс. Елементи уявлення можна піддавати тонкій настройці, приклад інтерфейсу рисунку 1.1.

Версія 2012 підтримує розширене спостереження за змішаними середовищами, а саме за машинами під керуванням Unix та Linux (так званих «ніх систем: агент *ніх підтримує HP-UX 11 і версії 2 або 3 на базі PA-RISC і IA64, Sun Solaris 9 на базі SPARC та 10 на базі SPARC та 32-розрядної платформи, Red Hat Enterprise Linux 4, 5 та 6 на 32- та 64-розрядних платформах, Novell SuSE Linux Enterprise Server 9 на 32-розрядній платформі, 10 SP1 та 11 на 32- та 64-розрядних платформах, а також IBM AIX 5.3, 6.1 та 7.1 на базі POWER: Для моніторингу

використовуються 2 ключі: sudo (для конфігурування стандартного облікового запису з потрібним рівнем доступу) та SSH (для безпечного обслуговування агента).

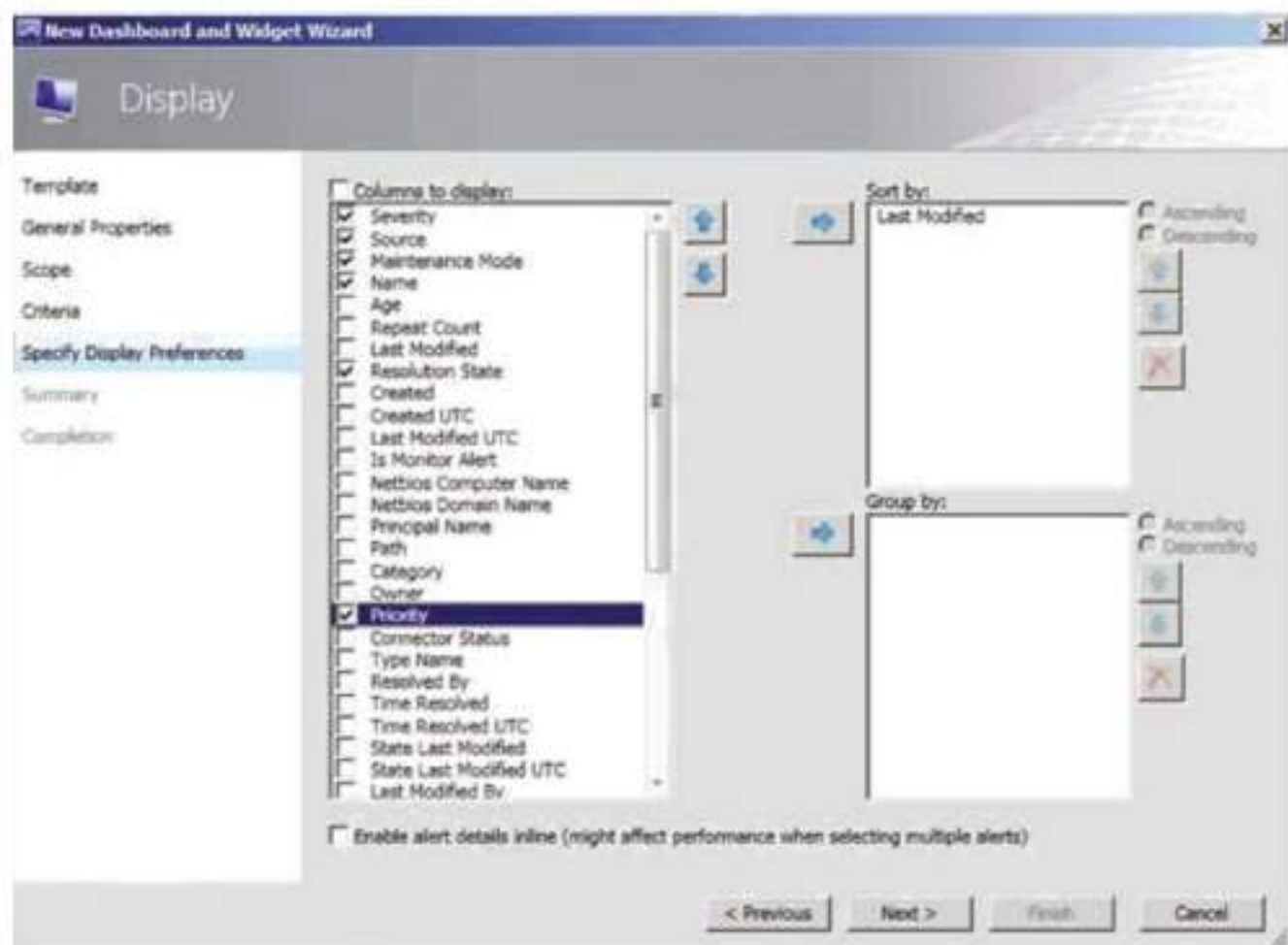


Рисунок 1.1 -Налаштування візуального представлення панелі SCOM

Отже, можна сказати, що System Center Operations Manager - це моніторинг високої доступності зі спрощеною інфраструктурою, що використовується в організаціях з великим парком машин під управлінням різних сімейств операційних систем, до складу якого безліч різноманітних засобів стеження, зокрема за мережевим устаткуванням, і навіть розширені засоби представлення зібраної інформації.

Однак дана система має ряд недоліків з точки зору вирішення конкретного технічного завдання:

- система моніторингу охоплює безліч загальних показників системи, але непридатна для стеження специфічними параметрами;
- досі робота з операційними системами поза сімейством Windows нестабільна;
- необхідність встановлення сервісу агента;
- неймовірна громіздкість і трудомісткість налаштування продукту «під себе»: система довше підходить для моніторингу загального стану та збору основних відомостей про велику структуру (наприклад, безліч клієнтських та серверних машин у домені).

Останній недолік зумовлює відмову від цієї системи як специфічного моніторингу проекту. Необхідно розгортати глобальний сервіс, встановлювати агентське програмне забезпечення на всі машини, що відстежуються, і налаштовувати безліч параметрів - скасовувати показники, які збираються за замовчуванням. Система відноситься до продуктів для контролю загального стану великої ІТ-структури без стеження за конкретними специфічними показниками, а значить зовсім не підходить під поставлені проектом цілі.

1.3 Огляд Zabbix

Zabbix – система, що вільно розповсюджується для комплексного моніторингу мережевого обладнання, серверів і сервісів. Складається із чотирьох частин:

- Сервер моніторингу (ядро) – виконує періодичне опитування та отримання даних, обробляє їх, аналізує, також здійснює запуск скриптів для розсилки оповіщень. Може віддалено перевіряти мережеві послуги, є сховищем, в якому зберігаються всі конфігураційні, статистичні та оперативні дані. Не

може розташовуватись на сервері під керуванням операційної системи сімейства Windows, а також OpenBSD.

- Проксі - збирає дані про продуктивність та доступність від імені Zabbix сервера. Всі зібрані дані заносяться в буфер на локальному рівні і передаються серверу Zabbix, до якого належить проксі-сервер. Zabbix проксі є ідеальним рішенням для централізованого віддаленого моніторингу місць, філій, мереж, які не мають локальних адміністраторів. Він може бути також використаний для розподілу навантаження одного сервера Zabbix. У цьому випадку, проксі лише збирає дані, тим самим на сервер лягає менше навантаження на ЦПУ та на введення/виведення диска.
- Агент – спеціальний демон, який запускається на об'єктах, що відстежуються і надає дані серверу, здійснюючи контроль локальних ресурсів і додатків (таких як жорсткі диски, пам'ять, статистика процесора і т. д.) на мережевих системах, тобто. ці системи повинні працювати з запущеним Zabbix агентом (проте моніторинг можна проводити не тільки за допомогою нього, але й за SNMP версій 1, 2, 3, запуском зовнішніх скриптів, що видають дані, та кілька видів зумовлених вбудованих перевірок, таких як ping, запит по http, ssh, ftp та інших протоколів, а також замір часу відповіді цих сервісів: Zabbix агенти є надзвичайно ефективними через використання вбудованих системних викликів для збору інформації про статистику. Zabbix-агенти підтримуються не тільки на *nix операційних системах, але AIX та Windows: Підтримувані платформи наведено в таблиці 1.1

Веб-інтерфейс - засіб візуального представлення Zabbix, реалізований на PHP, для запуску вимагає наявності веб-сервера, представлений на рисунку 1.2

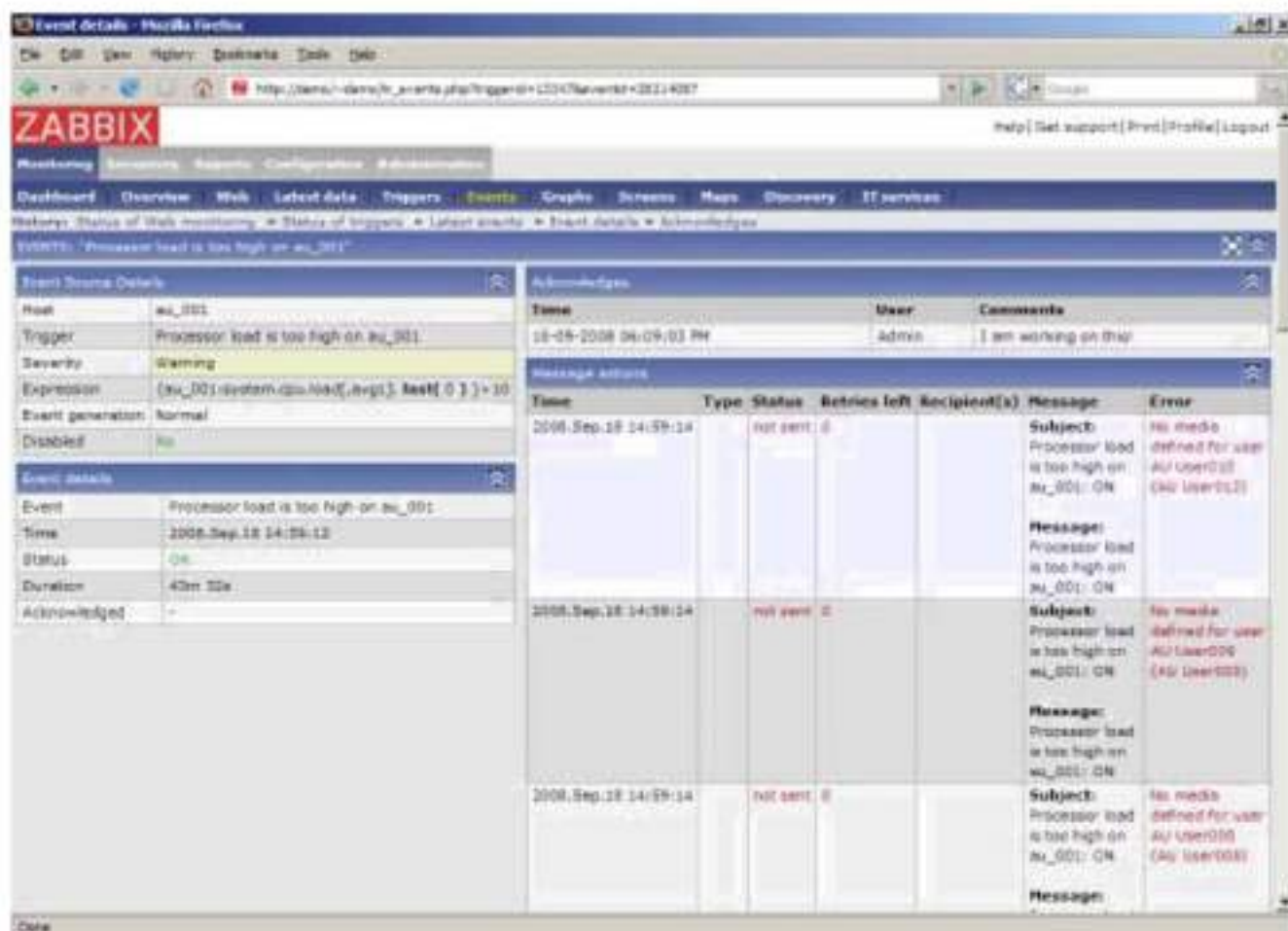


Рисунок 1.2 - Веб-інтерфейс моніторингу Zabbix

Zabbix підтримує платформи, зазначені у таблиці 1.1

Таблиця 1.1 - Підтримувані платформи

Платформа	ZABBIX-сервер	ZABBIX-агент
AIX	Підтримується	Підтримується
FreeBSD	Підтримується	Підтримується
HP-UX	Підтримується	Підтримується
Linux	Підтримується	Підтримується
Mac OS X	Підтримується	Підтримується
Novell Netware	-	Підтримується
Open BSD	Підтримується	Підтримується
SCO Open Server	Підтримується	Підтримується
Solaris	Підтримується	Підтримується
Tru64/OSF	Підтримується	Підтримується
Windows	-	Підтримується

За допомогою Zabbix можна здійснювати розподілений моніторинг до 10 00 вузлів, де конфігурація молодших вузлів контролюється старшими в ієрархії. Також продукт включає централізований моніторинг лог-файлів, можливість створювати карти мереж (вручну за шаблоном), виконання запитів до різних баз даних, генерацію звітів та тенденцій, виконання сценаріїв на основі моніторингу, підтримку інтелектуального інтерфейсу управління платформами (IPMI).

Zabbix надає гнучкі можливості з налаштування умов-тригерів, які включаються при аваріях та неполадках, і система починає моргати лампочками (насправді червоними квадратами), сповіщаючи адміністратора про можливу поломку. Також, при включенні тригера, веб-інтерфейс навіть починає попискувати на зразок будильника,

Zabbix досить самостійний і зможе відправити повідомлення на пошту, в jabber або sms за допомогою gsm-модему, або навіть спробувати самостійно підняти сервіс, що впав, виконавши заздалегідь певні дії, які запускаються при спрацьовуванні певних тригерів.

Для відображення логічної структури мережі можна створювати вручну карти мережі (приклад якої представлений на рисунку 1.3), що відображають саме розташування вузлів мережі та зв'язків між ними, причому поточний стан вузлів буде відображатися на карті.

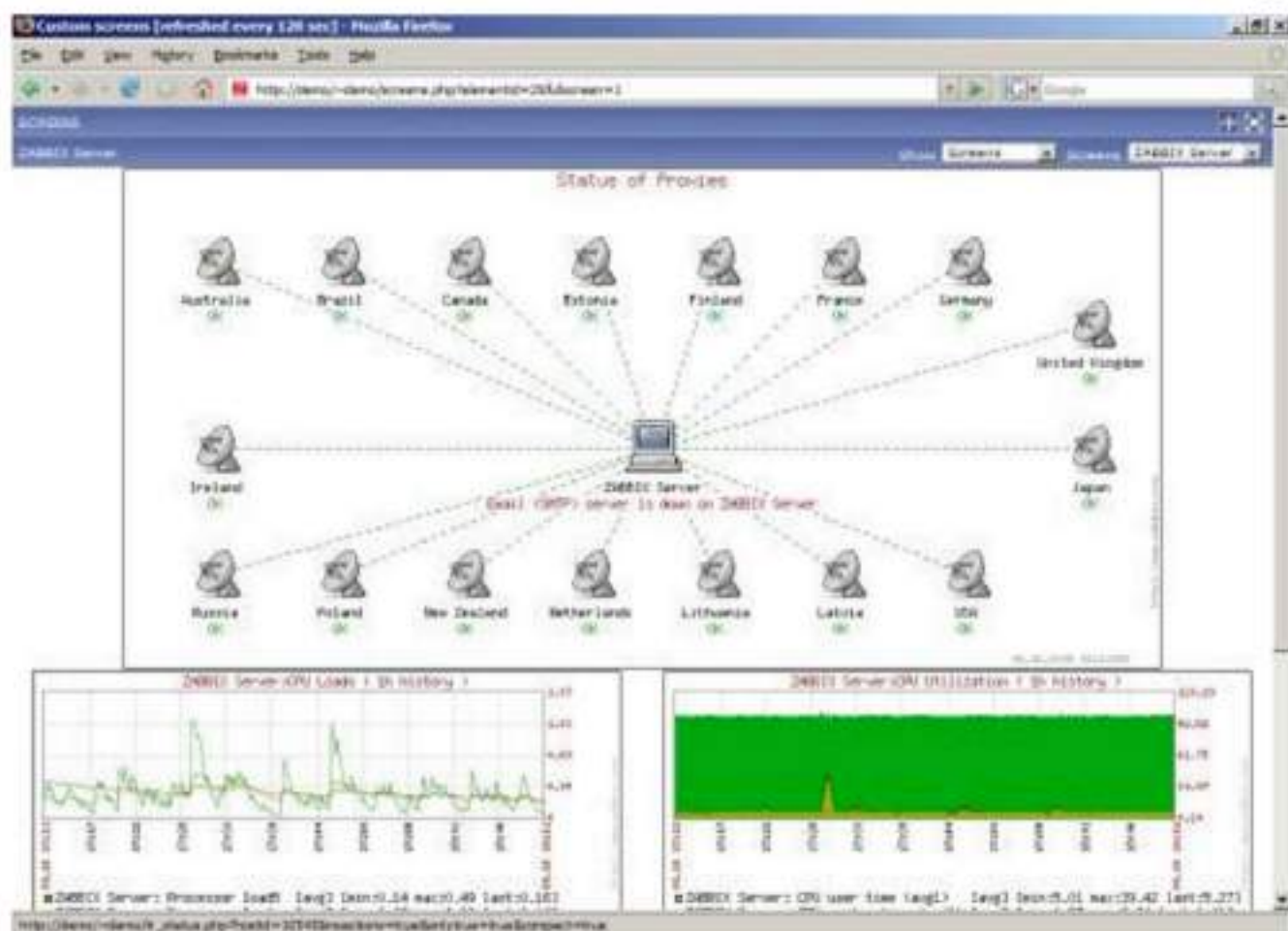


Рисунок 1.3 - Карти мереж в Zabbix

Автоматичне виявлення:

- автоматичне виявлення по діапазону IP-адрес, доступним сервісам та SNMP перевірка;
- автоматичний моніторинг виявлених пристроїв;
- автоматичне видалення відсутніх хостів;
- розподіл за групами та шаблонами залежно від повертається
- результату.

Запас у Zabbix є ще півдесятка функцій, які дозволяють ще більше спростити спостереження за мережею, такі як моніторинг стану веб-сайту за допомогою автоматичного виконання сценарію на кшталт імітації користувацьких дій на сайті. У результаті це одна з найпотужніших та найширших систем моніторингу.

У результаті ми отримуємо найбільш підходящу для наших цілей систему, яку також можна використовувати як «скелет» до власних скриптів моніторингу. Однак у черговий раз варто відзначити громіздкість сервісу, відсутність повної документованості можливостей проекту, а також необхідність встановлення програмного забезпечення агентства на всі машини.

Як додатковий мінус варто відзначити складність делегування прав – машина з сервісом найчастіше управляється операційною системою сімейства *nix, що робить трудомісткою взаємодію з доменними користувачами та правами з Active Directory (Windows системи).

1.4 Огляд Nagios

Nagios (спочатку Netsaint) – програма для моніторингу систем і мереж, що вільно розповсюджується. Спочатку розроблена для операційних систем на базі Linux, зараз однаково добре працює також під Sun Solaris, FreeBSD, AIX і HP-UX. За допомогою цієї програми доступні комплексне спостереження за всією IT-інфраструктурою, виявлення проблем одразу після їх виникнення, можливість ділитися отриманими при спостереженні даними із зацікавленими особами, моніторинг безпеки системи та, як наслідок, скорочення часу простою та комерційних втрат.

Можливості:

- моніторинг мережевих служб (SMTP, POP3, HTTP, NNTP, ICMP, SNMP);
- моніторинг стану хостів (завантаження процесора, використання диска, системні логи) у більшості мережевих операційних систем;
- підтримка віддаленого моніторингу через шифровані тунелі SSH або SSL;
- можливість побудови карт мереж (приклад рисунку 1.4).

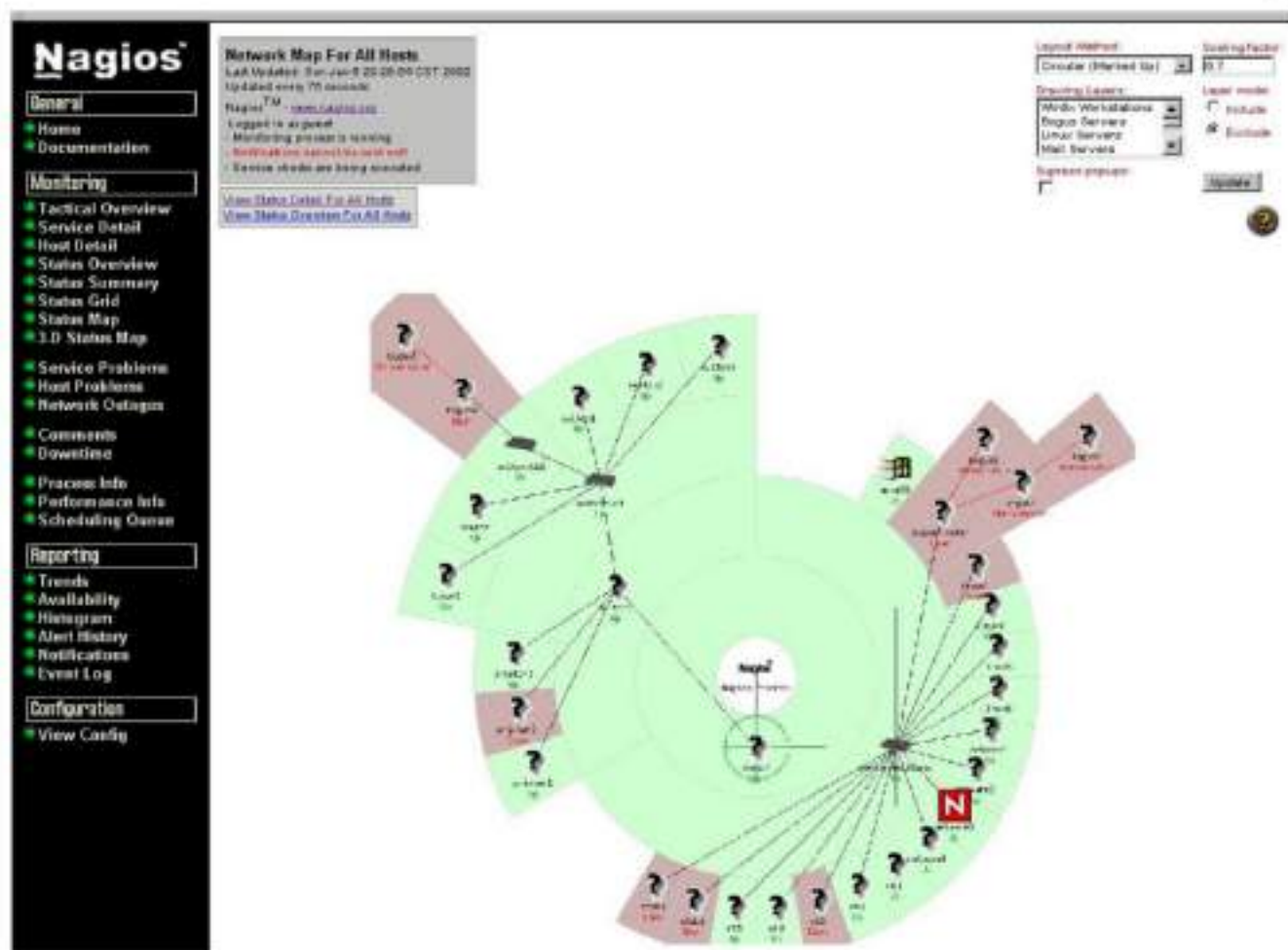


Рисунок 1.4 - Карта мережі Nagios

- проста архітектура модулів розширень (плагінів) дозволяє, використовуючи будь-яку мову програмування на вибір (Shell, C++, Perl, Python, PHP, C# та інші) легко розробляти свої власні способи перевірки служб;
- паралельна перевірка служб;
- можливість визначати ієрархії хостів мережі за допомогою «батьківських» хостів, дозволяє виявляти та розрізняти хости, що вийшли з ладу, та ті, які недоступні;
- надсилання оповіщень у разі виникнення проблем зі службою або хостом (за допомогою пошти, пейджера, смс або будь-яким іншим способом, визначеним користувачем через модуль системи);
- можливість визначати обробники подій, що відбулися зі службами або хостами для проактивного вирішення проблем;

- автоматична ротація лог-файлів;
- можливість організації спільної роботи кількох систем моніторингу з метою підвищення надійності та створення розподіленої системи моніторингу;
- включає утиліту nagiosstats, яка виводить загальне зведення по всіх хостах, якими ведеться моніторинг.

Відмовою від використання системи стали такі причини:

- "загальний" характер моніторингу показників;
- проблема взаємодії із серверами під керуванням Windows;
- "мережева" спрямованість моніторингу;
- додатковою причиною стала відсутність досвіду з даними програмним продуктом в компанії.

1.5 Огляд Cacti

Cacti - безкоштовний додаток моніторингу, що дозволяє збирати статичні дані за певні часові інтервали та відображати їх у графічному вигляді за допомогою RRDtool утиліти, призначеної для роботи з круговими базами даних (Round Robin Database), які використовуються для зберігання інформації про зміну однієї або декількох величин за певний проміжок часу. Стандартні шаблони збору включають статистику завантаження процесора, виділення оперативної пам'яті, кількості запущених процесів, використання вхідного/вихідного трафіку.

Cacti написаний в інфраструктурі Apache-PHP-MySQL, дозволяє налаштовувати збір й відображення даних моніторингу на основі веб-інтерфейсу, представленого на рисунку 1.5, з користувачем-френдлі організацією. Є можливість дописування своїх агентів збору даних.

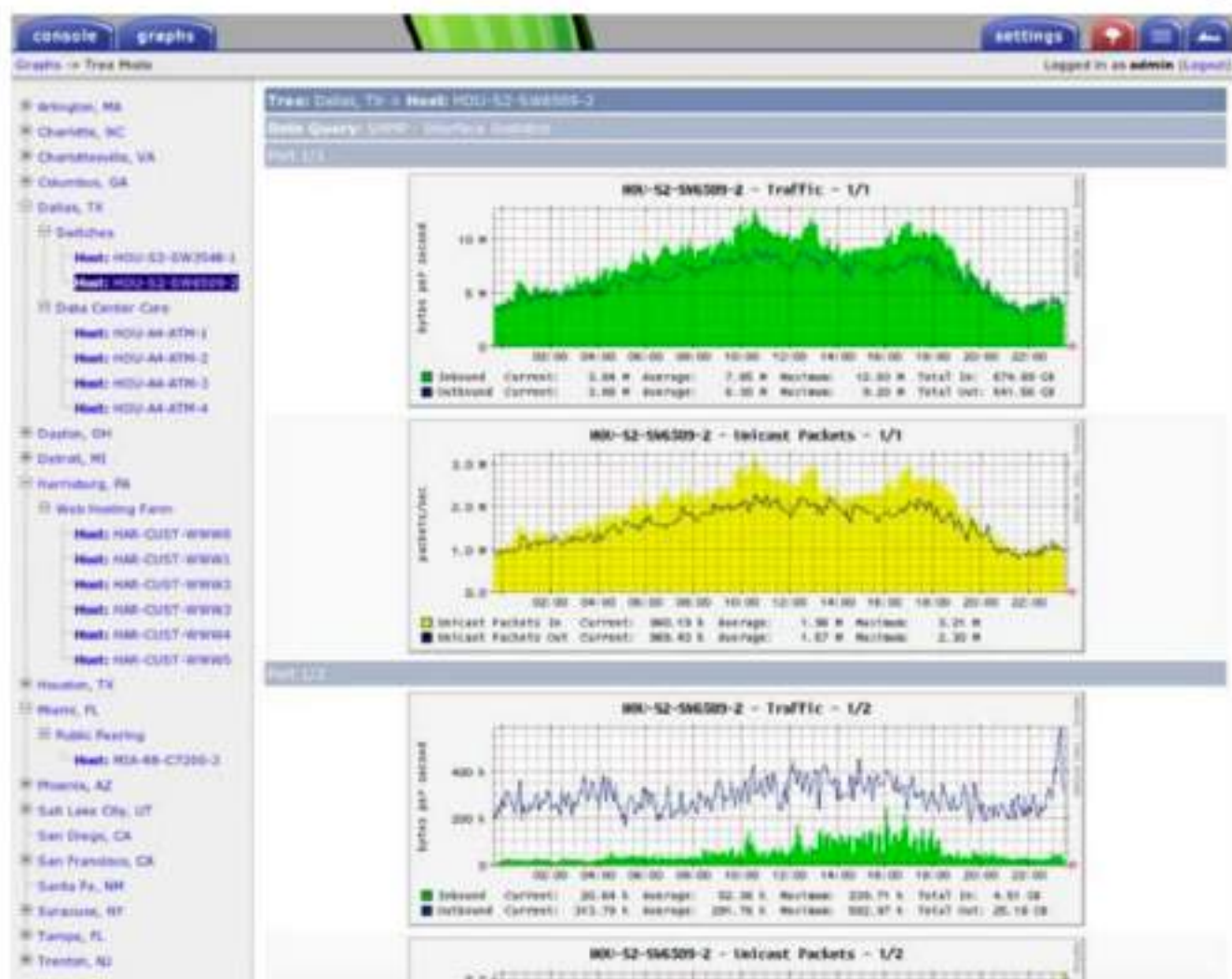


Рисунок 1.5 - Інтерфейс Cacti

Інтерфейс відображення статистики, зібраної з пристроїв, представлений у вигляді дерева, структура якого задається самим користувачем. Як правило, графіки групують за певними критеріями, причому той самий графік може бути присутнім у різних гілках дерева. Є варіант перегляду заздалегідь складеного набору графіків і є режим попереднього перегляду. Кожен із графіків можна розглянути окремо, при цьому він буде представлений за останній день, тиждень, місяць, рік. Можливо самому вибрати часовий проміжок, за який буде згенеровано графік.

Переваги Cacti:

- висока швидкість розгортання за мінімального додаткового кодування;
- простота та зручність інтерфейсу перегляду діаграм та їх налаштування.

Недоліки Cacti:

- досить швидке наростання кількості однотипних налаштувань у разі великої кількості середовищ та серверів;
- обмежена продуктивність "нерідких" JMX рішень для Cacti;
- відсутність можливості інвентаризації.

Логіка роботи системи заснована на обслуговуванні ряду пристроїв (Devices – у термінології Cacti). Кожен пристрій - це хост, до якого є доступ по мережі, тобто він характеризується IP-адресою або ім'ям DNS. З пристроєм асоційовані сховища даних (Data Sources). Кожне таке сховище обслуговує один графік (Graph), причому на цьому графіку може малюватись кілька змінних – сховище для них усіх буде одне. Сховище створюється на основі шаблону даних (Data Template), який визначає відповідність вхідних величин (отриманих з SNMP-запитів або зі скриптів) полям в базі даних і встановлює додаткові параметри зберігання цих величин. Самі ж вхідні величини виходять із методів збору даних (Data Input Methods) чи запитів (Data Queries). Перші призначені для величин, кількість яких заздалегідь відома (наприклад, кількість процесів - це одне ціле число), а другі - навпаки (наприклад, статистика з мережевих інтерфейсів, число яких може бути різним). Графік генерується з кругової бази даних (сховища) щоразу, коли завантажується сторінка. Алгоритм та параметри його створення задаються шаблоном графіка (Graph Template). Шаблони хостів (Host Templates) спрощують роботу з однотипними пристроями та дозволяють прив'язати певні шаблони графіків та запити до цього типу хоста. Наприклад, для маршрутизаторів Cisco – один набір графіків, а для UNIX-серверів - інший.

Cacti дозволяє завести кілька користувачів та розмежувати їхні права як на перегляд статистики, так і на керування системою. Логіка поділу доступу дозволяє для кожного користувача встановити спільну політику («Заборонити» або «Дозволити»), а потім зробити винятки.

У результаті Cacti дозволяє малювати графіки лише основних показників продуктивності, тоді як спроби моніторингу нестандартних показників сильно знижують продуктивність продукту. Також проекту дуже часто потрібна

інвентаризація (перерозподіл ресурсів та планування модернізації), а в даному пакеті вона відсутня.

Логіка роботи системи заснована на обслуговуванні ряду пристроїв (Devices – у термінології Cacti). Кожен пристрій - це хост, до якого є доступ по мережі, тобто він характеризується IP-адресою або ім'ям DNS. З пристроєм асоційовані сховища даних (Data Sources). Кожне таке сховище обслуговує один графік (Graph), причому на цьому графіку може малюватись кілька змінних – сховище для них усіх буде одне. Сховище створюється на основі шаблону даних (Data Template), який визначає відповідність вхідних величин (отриманих з SNMP-запитів або зі скриптів) полям в базі даних і встановлює додаткові параметри зберігання цих величин. Самі ж вхідні величини виходять із методів

1.6 Порівняльний аналіз вільно поширюваних систем

Таблиці 1.2,1.3,1.4 дозволяють оцінити функціонал існуючих систем комплексного моніторингу ІТ-інфраструктури, що вільно розповсюджуються.

Таблиця 1.2 – Функціонал вільно поширюваних систем моніторингу

Назва	Діаграми	Звіти SLA	Логічне угрупкування	Події	Прогнозування події
Cacti	Так	Так	Так	Так	Так
Nagios	Так	Через плагін	Так	Так	Ні
Zabbix	Так	Так	Так	Так	Так

Таблиця 1.3 - Функціонал вільно поширюваних систем моніторингу

Назва	Автоматичне виявлення	Агент	SNMP	Syslog	Зовнішні скрипти
Cacti	Через плагін	Ні	Так	Через плагін	Так
Nagios	Через плагін	Так	Через плагін	Через плагін	Так
Zabbix	Так	Підтримується	Так	Так	Так

Таблиця 1.4 - Функціонал вільно поширюваних систем моніторингу

Назва	Плагіни	Складність створення плагінів	Тригери/Тривоги	Інвентаризація	Карти
Cacti	Так	Середній	Так	Ні	Через плагін (Weathermap)
Nagios	Так	Легкий	Так	Через плагін	Динамічні
Zabbix	Так	Легкий	Так	Через плагін	Так

2 РОЗРОБКА СИСТЕМИ МОНІТОРІНГУ

2.1 Основні завдання

На початку розвитку інформаційних технологій навіть дуже великими компаніями використовувалися досить тривіальні рішення: всі системи були настільки спрощеними, що достатньо було команди фахівців, щоб підтримувати систему в робочому стані. Справді, мінілих складових було замало, тому й необхідність моніторингу була відсутня.

У міру ускладнення систем в ІТ з'явилися різні частини, які з комплексними внутрішніми складовими: системи передачі повідомлень, бази даних, інфраструктурні компоненти, службу каталогів та ін. використання яких доведеться зіткнутися з низкою проблем:

Моніторинг та управління частинами. Моніторинг кожної частини компонента здійснюється спеціалізованим програмним забезпеченням, налаштованим на специфічне завдання. Однак сервіс, який використовується для відстеження однієї складової, не може відстежувати іншу. Це найбільш фундаментальна проблема. Використання окремих строго специфічних інструментів є фундаментальною проблемою. Це призводить до розгляду кожної із систем в ізоляції – не враховуючи оточуючі системи, які також впливають на вихідну роботу. Висновок: спеціалізовані інструменти повинні бути другою лінією моніторингу для більш детального аналізу при необхідності. Перша лінія спостереження повинна збирати і надавати огляд звітів інформації комплексно по всіх системах в інфраструктурі «в одному вікні».

Відсутність зв'язку між користувачем, сервіс-дискон та ІТ-менеджментом. Взаємодія та зв'язок – це ключовий компонент, який робить працездатною будь-яку команду. У випадку ІТ зазвичай використовуються системи для організацій хелп-деска, маючи на увазі, що це розумний спосіб здійснення комунікацій, але цього не завжди буває достатньо. Системи хелп-деска, як правило, будуються на концепції реагування на проблему та подальше управління реакцією, і за визначенням, вони практично не проактивні.

Управлінські комунікації настільки ж важливі, як і непрості. Забезпечення чесних цифр у рівнях сервісного обслуговування, часу реакції, відсутності сервісів тощо – це все дуже важливо, якщо менеджменту необхідно приймати добре опрацьовані рішення щодо ІТ, але достовірну інформацію часто буває отримати зовсім не просто.

Моніторинг не те й не там. Зазвичай ІТ служба бачить багатокomпонентний, розподілений додаток, при цьому фахівці по кожному компоненту здійснюють моніторинг основних показників: вимірюють їхню продуктивність, використовуючи технічні метрики, такі як утилізація процесора, час відповіді і т.д. Коли один із компонентів виходить за допустимі значення, то відповідний інженер починає його детальну перевірку.

Користувачі та IT-служба вимірюють різні речі. У SLA, орієнтованих на IT, може вказуватися конкретний час відповіді запиту, надіслані на сервер СКБД, але це малокорисно, якщо додаток, з погляду користувача, працює «повільно». Що ще гірше, якщо починати мігрувати послуги та компоненти в «хмару», то втрачається більшість здібностей визначати продуктивність компонентів тим способом, яким звично було це робити в ЦОД до міграції. В результаті жодну із сторін не влаштують положення такого SLA.

Все це треба змінювати, причому з погляду користувача. Продуктивність індивідуальних компонентів важлива, але тільки в тій мірі, наскільки це впливає на загальну продуктивність, яка відчувається на конкретному робочому місці. Необхідно прописувати такі SLA, в яких користувач і служба IT знаходяться «на одній сторінці», потім керувати виконанням цих SLA тими способами та інструментарієм, які дозволяють це успішно робити. Деякі організації повідомляють, що вони переходять, або вже перейшли до роботи IT на основі надання сервісів — це, у широкому розумінні, означає, що компанія шукає спосіб реалізувати роботу IT служби у вигляді набору сервісів для різних департаментів організації та їх користувачів. Саме до такої моделі слід рухатись службам інформаційних технологій.

Втрата інформації. Знання про інфраструктуру компанії – і про те, як вирішувати проблеми має бути акумульовано та збережено. Ця вимога не тільки допоможе швидше вирішувати питання, що виникають у майбутньому, але також допоможе запобігати їх появі, дозволяючи приймати більш зважені рішення в галузі IT. У рамках компанії цю проблему зазвичай вирішують створенням централізованої системи бази знань, яка заповнюється фахівцями у міру виникнення та розв'язання ними різних інцидентів. Значення, зібрані монітором, також повинні акумулюватися, щоб створювати довгострокові звіти для з'ясування динаміки роботи того чи іншого вузла інфраструктури, щоб використовувати їх, наприклад, планування розширення пулу ресурсів конкретного сервісу, або виявлення вузького місця.

2.2 Вимоги до моніторингу

Вибір методів і об'єктів моніторингу залежить від багатьох чинників – конфігурації мережі, які у ній сервісів і служб, конфігурації серверів і встановленого ними ПЗ, можливостей ПЗ, що використовується для моніторингу і т.п.

- перевірка фізичної доступності устаткування;
- перевірка стану занедбаних служб і сервісів;
- детальна перевірка не критичних, але важливих параметрів функціонування: продуктивності, завантаження тощо;
- перевірка параметрів, специфічних сервісів і служб даного конкретного оточення (наявність деяких значень у таблицях БД, вміст лог-файлів).

Проектом, над яким здійснюватиме моніторинг, було висунуто такі конкретизуючі уточнення до загального переліку елементів моніторингу.

У першому пункту можна віднести перевірку доступності мережі хоста, віртуальної машини. Тут достатньо тестування наявності з'єднання та швидкості пригоди сигналу.

До другого пункту в конкретному проекті належить доступність IIS-сервісів, стан призначених завдань та служб.

До третього пункту відносяться перевірка довжини дискової черги, стану оперативної пам'яті, синхронізації серверних систем за часом, заповненості «кошика» тощо.

У четвертому, найважливішому пункту відносяться перевірки доступності сховища Apache Cassandra, наявності або відсутності в ній певного кіспейсу та значення показника його фактора реплікації. Для систем Windows також перевіряється, чи встановлено важливий хот-фікс на машині з сервісами, значення оперативної пам'яті, яку займає служба установки модулів і оновлень. Також необхідно було реалізувати очищення логів у Logstash-ElasticSearch комбінації за певними умовами.

2.3 Загальні відомості про побудову моніторингу

Враховуючи всі вищеописані вимоги до моніторингу, було вирішено використовувати систему, яка могла б працювати з будь-якими операційними системами, могла б виконувати скриптові сценарії на більшості поширених мов і не вимагала встановлення агентського програмного забезпечення на кожную машину, за станом якої здійснюватиметься стеження.

Було помічено, що система TeamCity, яка вже використовується в інфраструктурі проекту (за допомогою неї виконувалося розгортання проекту на бойові сервери), повністю відповідає необхідним умовам, а також не вимагає вкладення додаткових коштів та встановлення пакетів сервера та агента.

У загальному випадку моніторинг у TeamCity є окремим проектом (проектами) з однією або декількома конфігураціями всередині: розбиття на конфігурації краще здійснювати залежно від необхідності сповіщень тих чи інших користувачів про певні проблеми, тобто. оповіщення спрацьовує за будь-якої помилки всередині конфігурації, яка призводить до невдалої побудови. Також можна створювати окремий проект, а додати конфігурації (конфігурацію) до існуючого проекту розробки, на серверах якого реалізується моніторинг.

Перше зручно у разі великої компанії, коли розробкою та ремонтом поломок сервера займаються різні люди, другий випадок можна використовувати в маленькій компанії (у невеликому проекті), коли всі маніпуляції здійснюються силами розробників. Вид проекту представлений рисунку 2.1.

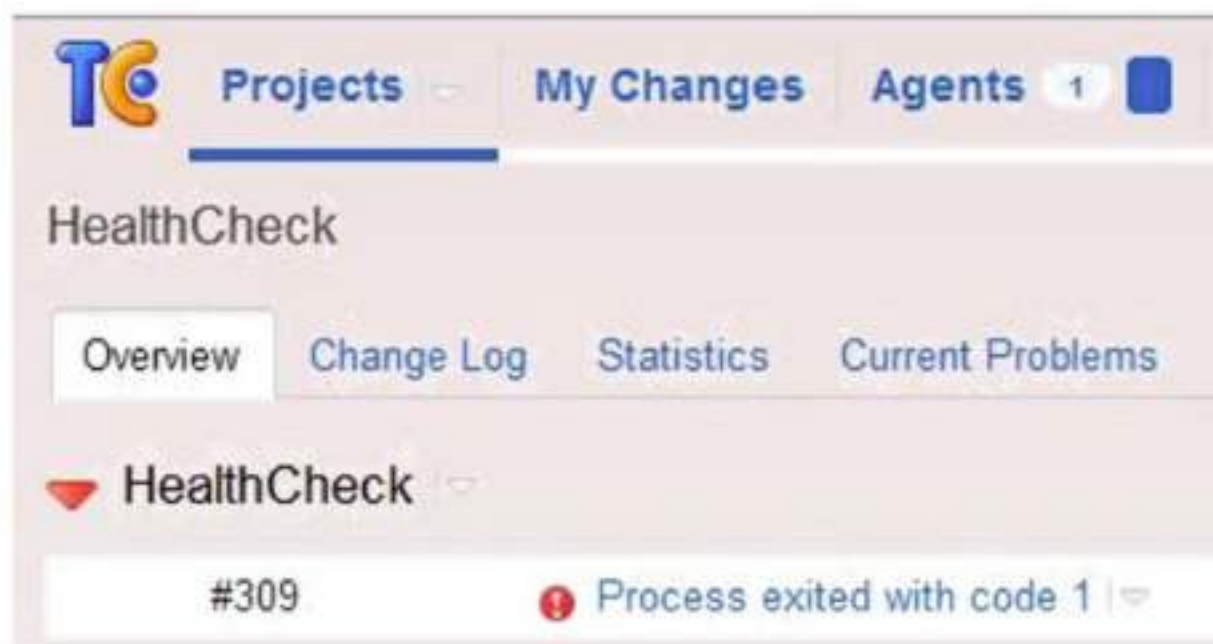


Рисунок 2.1 - Загальний вигляд проекту

Проект містить конфігурації для виконання, кожна з яких може складатися з одного або декількох кроків (рисунок 2.2). Крок у разі є скрипт (часто з параметрами). При виконанні кроку перевіряється значення показника, що відстежується.

Administration > HealthCheck Project > HealthCheck Configuration				
Build Steps				
Build Step	Description			
Powershell	Powershell x86 HealthTests\space.ps1<script> Execute: Only if all previous steps were successful			edit more
Scheduled tasks	Powershell Powershell x64 HealthTests\TASK.ps1<script> Execute: Even if some of previous steps failed			edit more
RAM 15%	Powershell Powershell x64 HealthTests\RAM.ps1<script> Execute: Even if some of previous steps failed			edit more
Purge old vm-log indices	Powershell Powershell x64 HealthTests\deleteINDEF no1<script>			edit more

Рисунок 2.2 - Покрокова стратегія конфігурації

Всі дії логуються (всередині лога також пишуться повідомлення користувача, які обумовлені всередині скрипта). Стану виконаних конфігурацій зберігаються історія TeamCity, де можна подивитися докладний лог виконання (рисунок 2.3).

#	Results
#309	 Process exited with code 1 ▾
#308	 Success ▾
#307	 Success ▾
#306	 Success ▾
#305	 Success ▾

Рисунок 2.3 - Історія збирання конфігурації

Вся інформація з історії збірок зберігається в базі даних (у нашому випадку це MySQL) і може бути використана аналітичним відділом для обробки даних про доступність інфраструктури протягом певного часу.

При збої користувачі, які налаштували для себе оповіщення даного проекту або конфігурації отримують повідомлення виду рисунок 2.4 або будь-якого іншого (є шаблони, що налаштовуються).

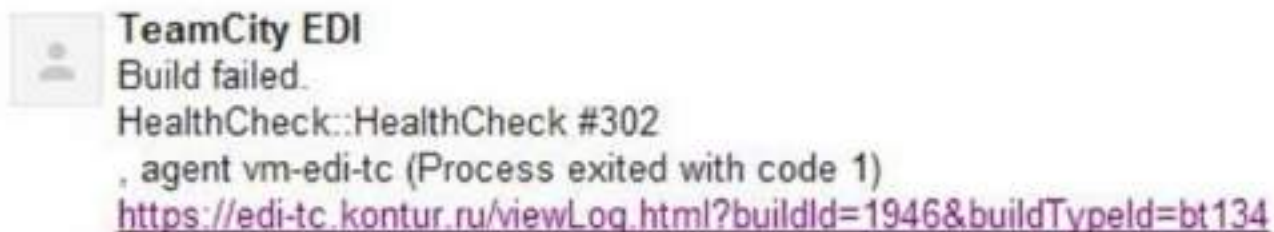


Рисунок 2.4 - Повідомлення про збій

Цей шаблон містить посилання на докладний лог конфігурації, при виконанні якої сталася помилка.

Складання конфігурації здійснюється за заданим адміністратором розкладом: визначення розкладу може відбуватися як у стандартній формі, прийнятій, наприклад, у планувальнику завдань, так і у вигляді стандартного cron-виразу.

Тригерів збірки може бути задано необмежену кількість. Також можна накласти різні умови, за яких той чи інший тригер буде або не буде активним у певний момент часу. Загальний вигляд конфігурованого тригера представлений рисунку 2.5.

Build Triggering

Build triggers are used to add builds to the queue either when an event occurs (like a VCS check-in) or periodically with some configurable interval.

Build Trigger	Parameters Description
Schedule Trigger	Cron expression: 0 0 * * * ? *, next scheduled time: 31 May 14 19:00 +0400 (server timezone)

Рисунок 2.5 - Повідомлення про збій

Крім того, можна використовувати вбудовані засоби для моніторингу стану, збору статистики, побудови звітів. Кінцевий вигляд статистики можна спостерігати на рисунку 2.6.

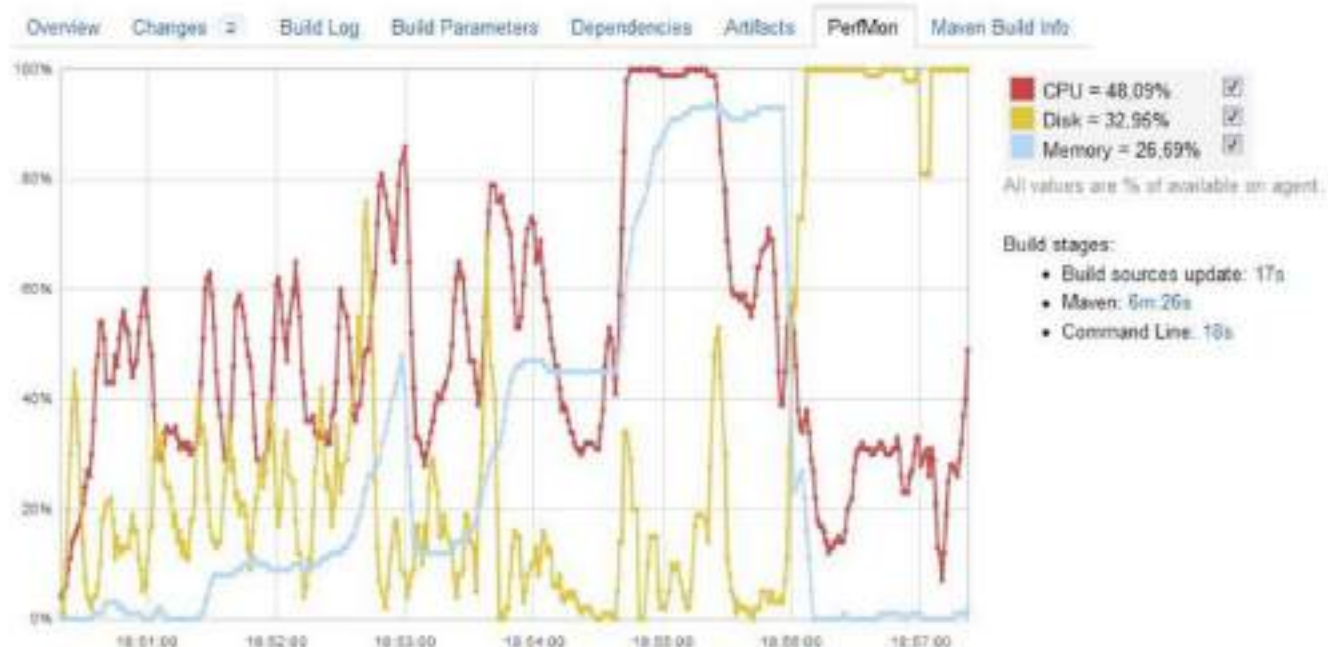


Рисунок 2.6 - Статистика

Дуже зручна можливість - є клікнути в точку на графіку - буде вивести лог, що відповідає цьому моменту часу - таким чином, можна подивитися, що саме створювало навантаження на BuildAgent.

Складання конфігурації має здійснюватися на машині з встановленим агентським програмним забезпеченням, проте звернення всім відстежуваним машин можна здійснювати з єдиного розгорнутого агента, тобто. встановлення ПЗ на всі сервери не потрібно.

Однак, за рахунок налаштування кількох агентів можна здійснити масштабування моніторингу. Розподіл збирань здійснюється за рахунок спеціального параметра збирання - вимоги до агента (рисунок 2.7).

Agents compatibility

In this section you can see which agents are compatible with requirements and which are not.

There are 3 agents authorized.



Compatible agents (1)

EDi pool (1)

vm-ed-4c

Incompatible agents (1)

RosAko pool (1)

vm-ed-4c2

Unmet requirements:

env.COMPUTERNAME equals VM-ED-4C

Рисунок 2.7 – Вимоги до агента

Існує стандартний набір параметрів, яким можна розподілити агенти на пули, але також можна задавати і власні умови поділу.

Крім вимог до агентів, можна використовувати параметри складання. Зазвичай туди розміщуються спеціалізовані змінні. Наприклад, автентифікаційні дані (є спеціальний вид параметрів, які шифруються після створення, що дозволяє зберігати паролі в закритому вигляді і використовувати їх при виконанні конфігурації). Також у моніторингу проекту параметри складання вказуються для скасування моніторингу сервера (наприклад, сервер вимкнено у зв'язку з ремонтними роботами і ви не бажаєте отримувати постійні оповіщення про його недоступність). Параметри можуть застосовуватись як для конкретного складання, так і для проекту загалом.

Скрипти для збирання не обов'язково зберігати прямо на агенті. Для чекауту скриптів та змін у систему TeamCity вбудована підтримка різних VCS. У моєму проекті моніторингу я використовую Git. До проекту можна прив'язати кілька систем контролю версій одночасно. Також можна вказати вид чекауту: чекають на сервері, якщо ви не хочете навантажувати агент під час збирання, або чекають на агенті, якщо ви вважаєте сам сервер TeamCity слабким місцем у вашій системі.

У результаті ми отримуємо багатогранну систему з широкими можливостями інтеграції, налаштування та масштабування. Однак існує єдиний мінус – моніторингові скрипти все ще залишаються на совісті системного адміністратора.

Отже, застосування такої системи для моніторингу серверів та служб проекту дозволить:

- підвищити експлуатаційну готовність серверного майданчика;
- отримати оперативну інформацію про стан систем (також контролювати значення критично важливих метриків);
- швидше шукати першопричину збою і навіть прогнозувати його заздалегідь;
- контролювати конфігурацію устаткування;

- надавати інформацію для клієнтів (у тому числі для аналітичного відділу), на основі якої можна зробити аналіз ступеня негативного впливу збою на послуги проекту.

2.4 Загальна структура скриптів. Спеціальні параметри

Загальна структура скрипта виглядає так:

- підключення додаткових модулів бібліотек;
- визначення початкових значень змінних (наприклад, значення змінної виходу);
- вибір набору серверів (сервісів) для перевірки стану;
- цикл збору інформації та перевірки стану;
- показ сповіщень про помилки;
- виведення підсумкової інформації щодо стану систем;
- вихід із скрипта за допомогою значення змінної виходу;
- надсилання оповіщення про завершення збирання.

Також використовуються спеціальні параметри:

- змінна для пропуску серверів (наприклад, щоб не отримувати сповіщення про відсутність сервера/сервісу під час ремонтних робіт);
- шифровані змінні, що містять у собі паролі для інтеграції з різними сторонніми сервісами;
- тимчасові змінні для побудови конфігурації. Наприклад, адреса електронної пошти, на яку потрібно надіслати звіт інвентаризації.

2.5 Відомості про додаткові модулі

Для універсальності (сумісності зі старими версіями Apache Cassandra) запити моніторингу використовують Thrift протокол. Thrift - мова опису інтерфейсів та двійковий протокол зв'язку. Включає готовий програмний конвеєр, що складається з шести рівнів, для роботи з клієнтською та серверною частиною. Верхній рівень складає згенерований код опису Thrift. Служби генерують із нього клієнтський та серверний код. На відміну від вбудованих типів, створена структура даних повертається як результат у згенерованому коді. Рівні протоколу та транспортування є частиною runtime-бібліотеки. У Thrift можна вибрати служби та змінити протокол та транспортування без перекомпіляції коду. Крім клієнтської частини Thrift включає серверну інфраструктуру для зв'язку протоколу і транспортування в blocking, non-blocking і multi-threaded серверах. Основа рівня Введення/Виведення по-різному реалізована для різних мов.

Thrift підтримує безліч протоколів:

- TBinaryProtocol – простий бінарний, що не оптимізований під оптимальне використання місця. Найшвидший для обробки, ніж текстовий, але складний для налагодження;
- TCompactProtocol – двійковий, компактніший, а також ефективніший для обробки;
- TDebugProtocol – текстовий, легкий для сприйняття, використовується для налагодження;
- TDenseProtocol – аналогічний TCompactProtocol, для зачистки мета-інформації;
- TJSONProtocol – використовує JSON для кодування даних;
- TSimpleJSONProtocol – використовується тільки для запису, підходить для парсингу за допомогою скриптових мов.

Підтримувані транспорти:

- TFileTransport – транспорт для запису до файлу;

- `TFramedTransport` – транспорт для використання сервера, що не блокує. Надсилає дані у кадрах, де кожен кадр передусім інформації про довжину;
- `TMemoryTransport` – введення/виведення пам'яті, використовує `ByteArrayOutputStream`;
- `TSocket` – використовує блокуючий введення-виведення для транспортування;
- `TZlibTransport` – виконує стиск за допомогою `Zlib`. Використовується у поєднанні з іншим транспортом;
Thrift також надає ряд серверів:
- `TNonblockingServer` - багатопотоковий сервер, використовує асинхронне введення/виведення;
- `TSimpleServer` - однопотоковий сервер, використовує стандартний блокуючий введення/виведення. Корисний для тестування;
- `TThreadPoolServer` - багатопотоковий сервер, використовує стандартний блокуючий введення/виведення;
- Thrift - це компонент для виклику двійкових віддалених процедур, багато в чому схожий на DCOM або CORBA, або .NET Remoting. Це особливо низькорівневий підхід до взаємодії з Cassandra, і хоча Thrift підтримує C#, підготувати все це ще й запустити — завдання далеко не тривіальне.

У моніторингу використовується підключення до Cassandra з Microsoft .NET Framework. Для цього може застосовуватися один із двох способів: некерований Apache Thrift API, архітектура якого представлена на рисунку 2.8 або стороння оболонка некерованого Thrift API.

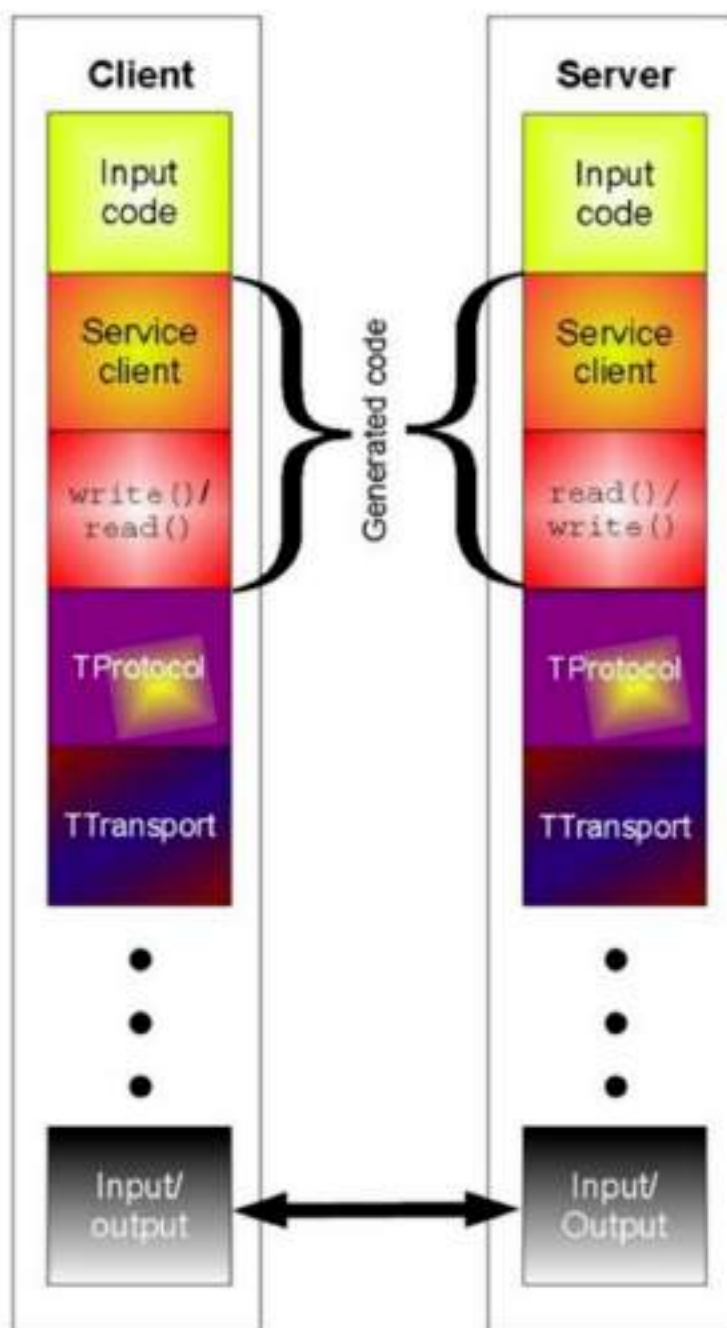


Рисунок 2.8 – Apache Thrift API клієнт-серверна архітектура

До альтернатив Thrift відносяться FluentCassandra, cassandra-sharp, Cassandraemon і Aquiles. Все це є проектами з відкритим кодом і пропонує деякі непогані абстракції Cassandra API. У моніторингу було вирішено скористатися FluentCassandra, але будь-який з цих проектів має право на життя. FluentCassandra доступний як NuGet-пакета, отже трудомісткість вбудовування їх у проект – мінімальна.

FluentCassandra - це бібліотека .NET для здійснення доступу до Apache Cassandra (проект з відкритим вихідним кодом, noSQL розподілена СКБД, призначена для обробки великих обсягів даних між кількома центрами обробки даних, без жодної точки відмови). Вона повністю сумісна з усіма командами, реалізованими інтерфейсом Apache Cassandra. Підтримується NET 4.0, в тому числі динамічні ключові слова та вирази LINQ для запитів до бази даних. Мета цієї бібліотеки - спрощення взаємодії з усіма версіями сховища з використанням методів .NET розробки.

Запити можуть здійснювати використання Thrift (API інтерфейс для всіх версій Apache Cassandra, на даний момент вважається застарілим), а також через Contextual Query Language (CQL) різних версій, починаючи з модифікації Cassandra 1.2.0. CQL3 – остання версія API, на відміну від Thrift, забезпечує тонкий шар абстракції над внутрішньою структурою, проте основні принципи моделювання залишаються тими самими.

Active Directory модуль для Windows PowerShell – модуль додаткових командлетів для керування та адміністрування служб каталогів та доменних імен, який підтримується всіма операційними системами, починаючи з Windows Server 2019 R2, тобто. Щоб використовувати командлети Active Directory Powershell, у домені повинен бути хоча б один контролер на основі Windows Server 2019 R2 або вище.

Модуль може бути використаний для керування доменами, для легкого доступу до каталогів наборів конфігурацій (AD LDS), а також як пакет для роботи з Active Directory Database Mounting Tool просторами.

У операційних системах Microsoft Windows 2016 Server, Windows Server 2019, і Windows Server 2012 , адміністратори використовували різні інструменти командного рядка та консолі керування Microsoft (MMC) оснастками для підключення до доменів Active Directory та контролю конфігурації LDS AD. Модуль Active Directory у Windows Server 2019 R2 в даний час забезпечує централізоване керування для ведення екземплярів служби каталогів. Існує можливість використання активного постачальника модуля для спрощення

SMS-сервіс - служба для відправки смс-повідомлень, розроблена в компанії. Ставить у чергу, а потім здійснює відправлення повідомлень від імені зареєстрованого раніше сервісу-відправника. Для кожного відправника формуються черги повідомлень. Це забезпечує пріоритезацію трафіку, а також підвищення надійності роботи системи.

Під час реєстрації відправника визначається:

- альфолітерний номер відправника;
- значення за промовчанням «Час життя повідомлення»;
- значення за замовчуванням «Прапор контролю доставки»;
- правила маршрутизації шлюзами (як правило, балансування по всіх зареєстрованих SMPP-шлюзах, але у разі вимоги захищених каналів може відрізнятися);
- пріоритет;
- пароль (опціональний параметр).

Як механізм авторизації використовується http basic access authentication, де ім'ям користувача виступає ім'я сервісу-відправника.

Реєстрація повідомлення на відправлення здійснюється POST запитом на адресу (листинг 2.1) .

Лістинг 2.1 – Адреса для POST-запиту у смс-сервіс

```
http://{кінцева_точка}/{ім'я_відправника}/sms.
```

POST запиту необхідно вказати параметри SMS повідомлення формат JSON – листинг 2.2

Лістинг 2.2 – Приклад JSON для надсилання повідомлення

```
{
  "text": "message text",
  "destinationAddress": {
    "address": "+723423423",
    "npi": 1,
```

```

"ton": 0
},
"sourceAddress": {
"address": "+75678656",
"npi": 1,
"ton": 0
},
"deliveryControl": true,
"scheduleDate": "2012-10-20 03:04:05Z",
"validityPeriod": "2012-10-21 03:04:05Z",
"deliverySchedules": [
{
"beginTime": "08:00:00",
"endTime": "18:00:00",
"dayOfWeek": "Friday",
"utcOffset": "00:00:00"
},
{
"beginTime": "08:00:00",
"endTime": "14:10:00",
"dayOfWeek": "Monday, Tuesday, Wednesday",
"utcOffset": "04:00:00"
}
]
}

```

Параметри SMS повідомлення наведено в таблиці 5.

При успішній реєстрації повідомлення на відправлення сервер поверне відповідь, що містить ідентифікатор відправленого повідомлення.

Таблиця 2.1 – Параметри повідомлення

Назва	Значення
DayOfWeek	Дні відправлення
DeliveryControl	Прапор контролю доставки
DeliverySchedules	Масив розкладів відправки повідомлень
deliverySchedules.beginTime	Час початку відправлення
deliverySchedules.endTime	Час кінця відправлення
DestinationAddress	Адреса одержувача
destinationAddress.address	Номер отримувача
destinationAddress.npi	Формат номера одержувача
destinationAddress.ton	Тип номера одержувача
ScheduleDate	Час, коли повідомлення необхідно виправити
SourceAddress	Адреса відправника
sourceAddress.npi	Формат номера відправника
sourceAddress.ton	Тип номера відправника
Text	Текст повідомлення
UtcOffset	Зміщення часу від Грінвіча
ValidityPeriod	Час життя повідомлення

Відповідь сервера на операцію надсилання повідомлення у форматі JSON, представленому на лістингу 2.3.

Лістинг 2.3 – JSON відповідь сервера

```
{
  "status": "Ok",
  "result": {
    "smsId": "123abc"
  }
}
```

Параметри відповіді півночі відображені у таблиці 2.2.

Таблиця 2.2 – Параметри відповіді сервера

Назва	Значення
Status	Статус виконання операції
result.smsId	Ідентифікатор зареєстрованого повідомлення

Для визначення статусу раніше відправленого повідомлення необхідно відправити GET запит на адресу, представлену на лістингу 2.4.

Лістинг 2.4 – Адреса GET-запиту статусу повідомлення

```
http://{кінцева_точка}/{ім'я_відправника}/sms/{код_повідомлення}
)
```

Відповідь сервера на операцію запиту статусу надісланого повідомлення у форматі JSON – лістинг 2.5.

Лістинг 2.5 – Відповідь сервера на запит статусу повідомлення

```
{
  "status": "Ok",
  "result": {
    "smsId": "524d62656b576b0f4000c780",
    "createDate": "2013-10-03 12:26:13Z",
    "lastChangeDate": "2013-10-03 12:33:13Z",
    "smsState": "Delivered",
    "sentInfos": [
      {
        "gatewayId": "megafon",
        "sentDate": "2013-10-03 12:33:13Z"
      }
    ]
  }
}
```

Параметри відповіді сервера наведено в таблиці 2.3.

Назва	Значення
Status	Статус виконання операції
result.smsId	Ідентифікатор зареєстрованого повідомлення
result.smsState	Статус повідомлення
result.createDate	Дата надходження повідомлення у сервіс
result.sentInfos	Набір елементів виду шлюз/дата.
result.lastChangeDate	Дата останньої зміни стану повідомлення

Структура елемента sentInfos виглядає чином, представленим у таблиці 2.4.

Повідомлення може бути в одному з наступних статусів, перелічених у таблиці 2.5.

Таблиця 2.4 – Статуси повідомлення

Назва	Значення
GatewayId	ідентифікатор шлюзу,
SentDate	дата надсилання повідомлення

Таблиця 2.5 – Структура sentInfos

Статус	Опис
New	Повідомлення розміщено в чергу на відправку
Captured	Повідомлення у стані відправки
Sent	Повідомлення успішно відправлено
Delivered	Повідомлення успішно доставлено
Obsolete	Закінчився час надсилання повідомлення
Fault	Помилка відправлення
Expired	Закінчився час життя повідомлення
Undelivered	Повідомлення неможливо доставити
Error	Помилка (кінцева)

Перемикання статусів відбувається згідно з графом на рисунку 2.10

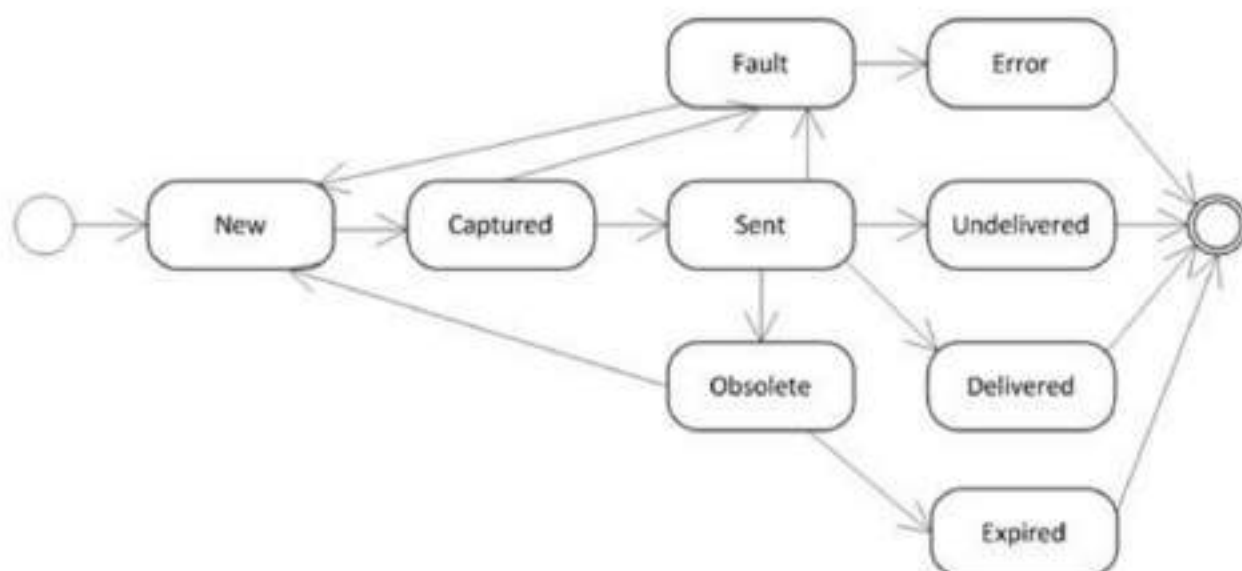


Рисунок 2.10 - Граф переходів статусів повідомлення

TON (Type of Number) визначає тип номера, відповідність номера та опису прокоментовано у таблиці 2.6.

Таблиця 2.6 – Значення TON

TON	Опис
0	Визначити автоматично (виходячи із номера)
Лікар	Здійснює авторизацію в системі, переглядає інформацію про розклад, статистику прийомів, пацієнтів
1	Міжнародний (починається з +7)
2	Національний (починається з 8)
3	Короткий номер (оператора)
4	Короткий номер (виділений і швидше за все платний)
5	Альфоітерний номер

Наприклад, якщо TON = 1, то SMS-центр відправить повідомлення лише в тому випадку, якщо номер починається з +7 (відсікає можливість надсилання на платні номери). Якщо гарантується, що номери будуть лише в міжнародному або національному форматі, можна встановити в 0 (автоматично).

NPI (Number Plan Indicator) визначає план нумерації. Для надсилання на абонентські номери достатньо встановити його в 0 (невідомий) або 1 (телефонні мережі).

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Основні показники

Цей скрипт призначений для стеження за вільним дисковим простором: кожен сервер опитується WMI для генерування набору дисків. Після цього, знову з використанням WMI, запитується інформація про вільний і зайнятий простір. Скрипт включає метод `Get-WmiCustom` (листинг 3.1), який розширює стандартний командлет `Get-WMIObject` додатковим параметром - тайм-аут запиту. Якщо за час, обмежений цим параметром, відповідь від сервера не була отримана, то генерується виняток, що містить повідомлення про недоступність машини і код виходу зі скрипту змінюється помилковим. Таке рішення дозволяє уникнути зависання скрипту при різних проблемах мережі, а також одночасно перевіряти доступність хостів та працездатність на них служби WMI.

Метод створюється `ManagementObjectSearcher`, який витягує колекцію керуючих об'єктів відповідно до заданого запиту. Цей клас є однією з найчастіше використовуваних вихідних точок входу для отримання відомостей про управління. Наприклад, його можна використовувати для перерахування всіх дисководів, мережевих адаптерів, процесів та інших керуючих об'єктів в системі, або для запиту всіх працюючих мережевих підключень, призупинених служб і т. д. Після створення екземпляра цього класу приймає в якості вихідних даних запит WMI об'єктом `ObjectQuery` або його похідними, і при необхідності об'єктом `ManagementScope`, що представляє простір імен WMI, у якому має бути виконаний запит. Примірник також може прийняти додаткові параметри, які містяться в об'єкті `EnumerationOptions`. При виклику методу `Get()` для цього об'єкта

ManagementObjectSearcher виконує даний запит заданої області і повертає колекцію об'єктів, що управляють, які задовольняють запиту, в ManagementObjectCollection.

Лістинг 3.1 - МетодGet-WmiCustom

```
Function Get-
WmiCustom([string]$computername,[string]$namespace,[string]$class,[int]$timeout,[int]$tim
eout = 15) {
    $ConnectionOptions = новий об'єкт
System.Management.ConnectionOptions
    $EnumerationOptions = новий об'єкт
System.Management.EnumerationOptions
    $timeoutseconds = New-timespan -seconds $timeout
    $EnumerationOptions.set_timeout($timeoutseconds)
    $assembledpath = "\" + $computername + "\" + $namespace
    $$Scope = новий об'єкт System.Management.ManagementScope
$assembledpath, $ConnectionOptions
    $$Scope.Connect()
    $querystring = "SELECT * FROM" + $class
    $query = new-object System.Management.ObjectQuery $querystring
    $searcher = новий об'єкт
System.Management.ManagementObjectSearcher
    $searcher.set_options($EnumerationOptions) $searcher.Query =
$querystring $searcher.Scope = $$Scope trap { $exitcode = 1; $_}
    $result = $searcher.get()
    return $result
}
```

Основною частиною скрипта є стандартна схема, описана раніше: спочатку задаються значення основних змінних – змінної завершення та змінної пропуску серверів. Далі, з використанням модуля ActiveDirectory створюється вибірка машин для моніторингу та визначаються їх імена для виконання запитів

(листинг 3.2). Якщо вибраний хост є серед колекції машин у спеціальній змінній SkipHosts, сервер пропускається.

Лістинг 3.2 - Створення вибірки для перевірки

```
Get-ADComputer -Filter * -Properties name | ForEach-Object {
    [String] $C = $_["name"]
}
```

Описаним вище методом Get-WmiCustom проводиться збір показників: набір дисків з їх ідентифікатором, потім доступне вільне місце кожному з них і загальний розмір диска (листинг 3.3).

Лістинг 3.3 – WMI запит дискових показників

```
$disks = get-wmicustom -class Win32_LogicalDisk -namespace
"root\cimv2" -computername $C -timeout 5 | where {$_.DriveType -eq 3}
```

Якщо розмір вільного дискового простору менше 10% від загального доступного місця, скрипт сигналізує про це за допомогою функції Write-Error і встановлює змінну виходу в значення 1. Подібний цикл операцій проводиться послідовно для кожної машини з вибірки.

Скрипт призначений для перевірки налаштувань та стану завдань опеределеного типу у планувальнику завдань Windows. Також, якщо необхідна задача не виконується, то скрипт примусово запускає її.

Скрипт створює об'єкт типу Schedule.Service, який надає доступ до планувальника завдань для керування зареєстрованими завданнями. Спочатку використовується метод Connect() для підключення до дистанційної машини та асоціації всіх наступних викликів на цьому інтерфейсі з цією віддаленою сесією. Потім застосовується метод GetFolder(), який отримує весь вміст зареєстрованих завдань.

Для потреб проекту потрібно, щоб постійно були запущені завдання, у яких тригер спрацьовує при включенні сервера і зареєстрований статус завдання не дорівнює Disabled.

Необхідно перевірити, що завдання, яке підходить за цими параметрами, запущено і її відсутнє налаштування автоматичної зупинки через три доби після запуску. І тому необхідно порівняти різні властивості об'єкта завдання (Task.Definition) з цими значеннями (листинг 3.4).

Лістинг 3.4 – Порівняння властивостей завдання

```
If ($tsk.Enabled) {
    $triggers = $tsk.Definition.Triggers
    foreach ($t in $triggers) {
        If ($ t.Type -eq 8) {
            If ($tsk.State -ne 4) {
                Write-Error "Computer $C, task $N
                enabled, scheduled to run at startup, but not running"
                $exitcode = 1
                Write-Host "starting it anyway..."
                $tsk.Run($null)
            }
        }
        If
        ($tsk.Definition.Settings.ExecutionTimeLimit -ne "PT0S") {
            Write-Error "Computer $C, task $N
            enabled, scheduled to run at startup, but ExecutionTimeLimit = 3
            days"
            $exitcode = 1
        }
    }
}
```

Якщо ж завдання зупинено, потрібно змінити код виходу на помилковий і спробувати запустити завдання методом Run().

Після перевірки кожного хоста необхідно обнулити набір отриманих завдань (лістинг 3.5).

Лістинг 3.5 – Скидання набору завдань.

```
[System.Runtime.InteropServices.Marshal]::ReleaseComObject($sched) | Out-Null
Remove-Variable sched;
```

Перевірка вільної оперативної пам'яті проводиться подібно до вищеописаного скрипту з контролю вільного місця на диску: по вибірці комп'ютерів з Active Directory проводиться збір відомостей за допомогою WMI запиту. Результати обробляються за певними умовами.

Однак, крім пропуску машин із спеціальної змінної SkipHosts, необхідно також не перевіряти машини із встановленою службою віртуалізації Hyper-V, т.к. Наявність оперативної пам'яті на хості для віртуалізації - не важливий показник, інтерес представляють тільки значення на самих віртуальних машинах.

Визначення наявності служби віртуалізації здійснюється через wmi-запит, представлений на лістингу 3.6.

Лістинг 11 – WMI-запит до служб

```
$feature = Get-WmiObject -Class Win32_OptionalFeature -cn $CI
Where {$_.Name -eq "Microsoft-Hyper-V"}
```

Потім, для остаточної вибірки машин визначається повний обсяг оперативної пам'яті та зайнятий відсоток способом, описаним у лістингу 3.7.

Лістинг 3.7 – Визначення показників оперативної пам'яті

```
$Committed = Get-WmiObject -Query "SELECT * FROM
Win32_PerfRawData_PerfOS_Memory " -ComputerName $C | select -
ExpandProperty
CommittedBytes
```

```
$Total = Get-WmiObject -Query "SELECT * FROM Win32_OperatingSystem" -ComputerName $C | select -ExpandProperty TotalVisibleMemorySize
```

Якщо обсяг доступної оперативної пам'яті на сервері менше 10%, то про це надходить повідомлення в лог з використанням функції Write-Error, а потім змінна виходу зі скрипта приймає значення 1, щоб при закінченні виконання повідомити про помилку при складанні цього кроку конфігурації.

Для перегляду та аналізу різноманітних логів у проекті використовується зв'язка Logstash-Elasticsearch-Kibana.

Logstash – безкоштовна open-source програма на java для збирання та нормалізації логів. Може приймати логи або з локальних файлів, або через порти tcp/udp/ Різних вхідних (input) фільтрів налічується 26. Також є вхідний модуль, для збору повідомлень з twitter'a або irc.

ElasticSearch - безкоштовний open-source пошуковий сервер, заснований на Apache Lucene. Швидкий, що легко налаштовується і дуже масштабується.

Kibana - веб-інтерфейс написаний на ruby для відображення даних з ElasticSearch. Просте налаштування, але безліч функцій - пошук, графіки, stream.

Моніторинг потрібно переглядати датування логів, і, якщо існують застарілі, то видаляти їх. Управління логами здійснюється через HTTP API. Для видачі статистики логів достатньо надсилання HTTP-GET запиту для видалення – HTTP-DELETE. Для кожного із запитів у скрипті реалізовано окремі методи.

Перший метод Execute-HTTPGetCommand(), у якому створюється об'єкт класу System.Net.WebRequest, який виконує запит до заданого URI (Uniform Resource Identifier). Потім запитується відповідь на запит, результат якого зчитується символами потоку байтів у певному кодуванні (System.IO.StreamReader). Цей потік буде повернутий з методу у вигляді набору рядків, які містять інформацію виду, представленого на лістингу 3.8.

Лістинг 3.8 – Відповідь на запит списку логів

```
"logstash-2014.06.02-20.00" : {
```

```

"primaries" : {
  "docs" : {
    "count" : 146058,
    "deleted" : 0
  }
}

```

За цією інформацією можна визначити дату створення індексу за допомогою регулярного виразу на лістингу 3.9.

Лістинг 3.9 - Регулярний вираз для визначення дати створення індексу

```
[regex]$ex = "logstash-\d{4}`.\d{2}`.\d{2}-\d{2}`.\d{2}"
```

Необхідно видалити логи, які були зібрані понад добу тому. Для видалення потрібно надіслати запит HTTP-DELETE. Для цього створено метод `Execute-HTTPDeleteCommand()`. Його структура аналогічна структурі методу `Execute-HTTPGetCommand()`. Відмінність полягає у зміні об'єкта `System.Net.HttpWebRequest`, зокрема його властивості `Method`, яка набула значення "DELETE".

Основна частина скрипта запитує всю інформацію про поточні логи на сервері за допомогою методу `Execute-HTTPGetCommand()`, потім визначає дату для порівняння і формує набір логів для видалення. Потім, за допомогою методу `HTTPDeleteCommand()` виконується видалення застарілих логів.

Часто для вирішення проблем, виявлених вже після релізу операційної системи або її компонентів, компанія-розробник випускає патчі, установка яких необхідна тільки при попаданні під певні умови.

Скрипт моніторингу перевіряє наявність хотфіксу KB2480994. Хотфікс вирішує проблему з витоком пам'яті під час використання у додатку методу `EnvelopedCms.Decode` із Microsoft .NET Framework. Серед додатків, що входять до проекту, є `CryptoService`, в якому використовується цей метод. Моніторинг спочатку перевіряє наявність цього сервісу у списку активних завдань планувальника, а потім, якщо перевірка виявилася успішною, то через клас

win32_quickfixengineering за допомогою wmi-запиту в отриманому списку встановлених хотфіксів проводиться пошук KB2480994 (листинг 3.10).

Лістинг 3.10 – WMI-запит про наявність KB2480994

```
$hf = gwmi -cl win32_quickfixengineering -cn $c | where
{$_Hotfixid -eq "KB2480994"}
```

Якщо патч з таким значенням властивості Hotfixid відсутній у списку, то скрипт рапортує про помилку на сервері, що перевіряється, і встановлює значення змінної виходу в одиницю.

Для більшості сервісів проекту, а також для баз даних, критичним фактором є точна синхронізація часу всередині ферми. Стандартним засобом синхронізації є Служба часу Windows. Однак іноді у ній можуть відбуватися збої. Додатковим засобом відстеження синхронізації є моніторинг.

З допомогою Net.Sockets.Socket, який реалізує інтерфейс сокетів Berkeley, скрипт звертається до публічного NTP-сервера pool.ntp.org і запитує поточний час (листинг 3.11).

Лістинг 3.11 – звернення до NTP за допомогою Net.Sockets.Socket

```
$Socket = New-Object
Net.Sockets.Socket([Net.Sockets.AddressFamily]::InterNetwork,
[Net.Sockets.SocketType]::Dgram,
[Net.Sockets.ProtocolType]::Udp)
$Socket.ReceiveTimeout = 120000
$Socket.SendTimeout = 120000
$Socket.Connect($NTPServer,123)
$t1 = Get-Date # Start of transaction... the clock is
ticking...
[Void]$Socket.Send($NtpData)
[Void]$Socket.Receive($NtpData)
$t4 = Get-Date # End of transaction time $Socket.Close()
```

Відповідь конвертується у формат, подібний до застосовуваного на агенті TeamCity. За допомогою WMI запиту до класу win32_operatingsystem визначаємо поточний час на агенті (листинг 3.12).

Лістинг 3.12 – Визначення поточного часу

```
$TCQueryTime = gwmi win32_operatingsystem -computer "vm-edi-tc"
$TCTime =
$TCQueryTime.converttodatetime($TCQueryTime.localdatetime)
```

Після цього поточні значення порівнюються. Якщо розбіжність склала більше хвилини, то Скрипт відразу формує повідомлення про помилку, а також змінює код виходу на 1 значення.

Якщо ж розбіжність становила менше хвилини, перевірка вважається пройденою успішно.

Тоді слідом, з використанням Powershell модуля ActiveDirectory, формується список хостів для перевірки часу. Значення часу для порівняння також виходить за допомогою WMI-запиту до сервера для перевірки, а еталонний час для порівняння витягується з відповіді агента TeamCity (листинг 3.13).

Лістинг 3.13 – Визначення часу агента та поточного сервера

```
$TCQueryTime = gwmi Win32_LocalTime -computer "vm-edi-tc"
$TCTime = (Get-Date -Day $TCQueryTime.Day -Month
$TCQueryTime.Month -Year $TCQueryTime.Year -Minute
$TCQueryTime.Minute -Hour $TCQueryTime.Hour -Second
$TCQueryTime.Second)
$QueryTime = gwmi win32_operatingsystem -computer $server $DT =
gwmi Win32_LocalTime -computer $server
$LocalTime = (Get-Date -Day $DT.Day -Month $DT.Month -Year
$DT.Year -Minute $DT.Minute -Hour $DT.Hour -Second $DT.Second)
```

Знову перевіряється, чи становить величина розбіжності хвилину і більше. Якщо розбіжність мінімальна, то хост вважається синхронізованим із фермою. В іншому випадку, моніторинг подає сигнал повідомленням про помилку.

Зрештою, якщо хоча б один сервер відрізняється від ферми за часом більш ніж на хвилину, то виконання кроку складання неуспішно.

Паралельно порівнюється встановлене налаштування часового поясу, яке також виходить за допомогою WMI-запиту, представленого на лістингу 3.14.

Лістинг 3.14 – Запит статусу головної сторінки

```
$QueryTimeZone = (gwmi Win32_TimeZone -Comp $server).Description
if ($QueryTimeZone -ne "(UTC+04:00) Moscow, St. Petersburg,
Volgograd")
{
```

Для перевірки доступності достатньо проаналізувати код відповіді, як описано в лістингу 3.15.

Лістинг 3.15 - Аналіз коду відповіді

```
if ($status -ne "OK")
{
Write-Error "$front is not responding"
$exitcode = 1
}
```

Перевіряємо код відповіді на відповідність успішному. Якщо код не дорівнює "OK", тоді відправляємо повідомлення про помилку і задаємо нове значення коду виходу.

Крім описаної вище перевірки доступної вільної оперативної пам'яті, також необхідно стежити, чим саме зайнята зарезервована частина. Так як сервери, на яких встановлені послуги програми, вимагають високого показника доступності і тому перезавантажуються дуже рідко, то було помічено «витікання» пам'яті під процеси, ініціалізовані власником службою установки модулів Windows

(TrustedInstaller). Особливо це помітно при рідкісному встановленні оновлень операційної системи, що потребують перезавантаження. Тому необхідно стежити за кількістю пам'яті, яка відведена під подібні процеси, щоб при високих значеннях виділеної оперативної пам'яті здійснити встановлення оновлень та перезавантаження проблемного сервера.

Перевірка відбувається за списком машин, сформованого за допомогою вибірки командлет Get-ADComputer з Powershell модуля ActiveDirectory.

Значення пам'яті визначається WMI-запитом на лістингу 3.16.

Лістинг 3.16 - WMI-запит займаної пам'яті

```
Get-WmiObject -ComputerName $server -Class Win32_Process -Filter
"Name='TrustedInstaller.exe'" | Select -ExpandProperty WorkingSetSize
```

Отримана величина зберігає величину байтах. Для зручності оцінки значення перекладається гігабайтами, після чого скрипт виводить інформаційне повідомлення по кожному хосту.

Якщо повернена величина більша за встановлену «критичну», то в лог збірки буде виведено попередження, а виконання кроку завершено з відповідною помилкою, яка і послужить сигналом системному адміністратору.

Крім відстеження відсотка місця, що використовується на диску, також необхідно знати, чим диск зайнятий. Часто, наприклад, у процесі видалення логів сервісів або резервних копій баз даних, файли складуються в «Кошику» Windows, тому все ще використовують дисковий простір. Щоб відстежувати подібне явище, у збірці використовується скрипт нижче.

Перевірка відбувається за списком машин, сформованого за допомогою вибірки командлет Get-ADComputer з Powershell модуля ActiveDirectory. За допомогою змінної задається стандартний шлях до папки «Кошки» - лістинг 3.17.

Лістинг 3.17 – Стандартний шлях «Кошки»

```
$devicePath = ("\"+$server+"\\C$\"+$Recycler2008)
```

Потім командлетом `Get-ChildItem` рекурсивно пізнаються всі вкладені файли, а потім визначається місце, яке займає на диску (листинг 3.18). Опитування зроблено у вигляді `Job` операції з таймаутом. Подібний захід необхідний, щоб не створювати навантаження моніторинговим скриптом на диск: якщо дискова черга велика через працюючих сервісів, то додаткове завантаження лише збільшує її, що негативно позначається на продуктивності.

Лістинг 3.18 – Опитування розміру файлів із таймаутом

```
$FileSystem = new-object -com "Scripting.FileSystemObject";
$getFolderAttr = $FileSystem.GetFolder($recyclepath);
$getSubFolderAttr = $getFolderAttr.SubFolders
$colItems = start-job {param ($getSubFolderAttr)
$getSubFolderAttr | measure-object -properties size -sum } -
ArgumentList $getSubFolderAttr | wait-job -Timeout 15 | receive-job
```

Отримане значення для зручності сприйняття перекладається в мегабайти, після чого виводиться інформаційне повідомлення. Якщо отриманий розмір перевищує заданий «критичний», виконання скрипта завершується з вихідним кодом, рівним одиниці, що сигналізує системному адміністратору про необхідність провести обслуговування даної машини.

3.2 Спеціалізовані показники. Підключення бібліотек

Усі попередні показники можна відстежувати більшістю стандартних моніторингів з коробки. Однак найчастіше для серверів кожної галузі є показники, які є специфічними і не відстежуються жодним програмним забезпеченням.

Розглянемо скрипт для спостереження за значенням фактора реплікації кіспейсу в `noSQL`-системі БД `Cassandra`. Для цього будемо використовувати підключається `.Net` бібліотеку `FluentCassandra`.

Спочатку необхідно підключити бібліотеку і завантажити типи даних, що містяться в ній так, як це показано на лістингу 3.19.

Лістинг 3.19 – Підключення бібліотеки FluentCassandra

```
[System.IO.Path]::GetDirectoryName($myInvocation.MyCommand.Definition) + "\FluentCassandra101.dll"
Add-Type -Path $dllpath
[reflection.assembly]::LoadWithPartialName(
    "FluentCassandra.Connections")
[reflection.assembly]::LoadWithPartialName("FluentCassandra.Types")
[reflection.assembly]::LoadWithPartialName(
    "FluentCassandra.Linq")
```

Потім створюємо новий об'єкт, використовуючи тип `Connections.Server` з підключеної бібліотеки, за допомогою якого підключаємося до одного з існуючих серверів Cassandra, а потім визначаємо змінну, всередині якої будуть імена існуючих кіспейсів (лістинг 3.20).

Лістинг 3.20 – Створення опису кіспейсів

```
$system = New-Object
FluentCassandra.CassandraContext($KeyspaceName, $Server, '', '');
$test = $system.DescribeKeyspaces();
```

Для кожного отриманого кіспейсу створюємо новий об'єкт типу `CassandraContext` із підключеної бібліотеки (всередині будуть міститися властивості кіспейсу) та визначаємо фактор реплікації (лістинг 3.21).

Лістинг 3.21 – визначення фактора реплікації

```
$t = $KeyspaceName.KeyspaceName
$db = New-Object FluentCassandra.CassandraContext($t,
$Server, '', '');
```

```
$KeySp = $db.Keyspace
[int]$rf = $KeySp.GetDescription().Replication_factor;
```

Порівнюємо отримане значення фактора реплікації кіспейсу з необхідним. Якщо воно відмінне від необхідного, то змінюємо вихідний код на 1. Також, якщо визначені невідомі кіспейси, які не описані в скрипті раніше, то повідомляємо про це в повідомленні і змінюємо вихідний код.

Далі йде відправка SMS-повідомлення через веб-сервіс, робота якого описана вище. Надсилання здійснюється тільки якщо вихідний код не дорівнює успішному.

Створюємо функцію надсилання смс-повідомлення (листинг 3.22), параметром якої буде номер телефону абонента, що передається.

Необхідно здійснити запит POST для завершення дії.

Спочатку вкажемо облікові дані: пароль передається як параметр (зберігається у вигляді зашифрованих даних усередині TeamCity), аналогічно можна передавати ім'я користувача, але в даному випадку воно просто вказано у вигляді рядка.

Лістинг 3.22 – функція надсилання смс-оповіщення

```
[string]$password = (Get-Item env:smsspassword).Value
$webRequest =
[System.Net.WebRequest]::Create("https://smsserviceaddress")
$webRequest.PreAuthenticate = $true
$webRequest.Method = "POST"
$webRequest.Credentials = new-object
system.net.networkcredential("Username", $password)
```

Далі визначимо дані запиту: метод (POST), облікові дані (створюємо новий об'єкт .Net типу networkcredential і записуємо до нього раніше певні параметри) та тіло запиту. Формат тіла запиту було визначено вище, на чолі опису смс-сервісу. У нашому випадку він виглядає як на лістингу 3.23.

Лістинг 3.23 – POST-запит для sms-service

```
$PostStr = @"
{
  "text": " Cassandra replication_factor check FAILED!",
  "destinationAddress": {
    "address": "$number",
    "npi": 1,
    "ton": 0
  }
}
"@
```

Після виконання запити необхідно отримати відповідь від sms-сервісу, щоб дізнатися, чи було успішно надіслано sms-повідомлення. Відповідь можна повернути як результат виконання HTTP-GET запити, цим дізнатися поточний статус відправки повідомлення.

Для стеження за статусом доступності серверів з базою даних Apache Cassandra застосовується збір даних з веб-оболонки, надається спеціалізованою бібліотекою MX4J. Вбудовування в Apache Cassandra відбувається автоматично, при копіюванні .jar файлу в папку з серверною частиною СКБД.

Після перезапуску сервера Cassandra з'являється доступ до сторінки з різними показниками, пов'язаними як із самою java-машиною, так і з внутрішніми параметрами СУБД. Для моніторингу використовується компонент FailureDetector, в якому можна спостерігати розгорнуту карту станів кільця Cassandra (рисунк 2.11).

Key	Key class	Value
/192.168.191.68	java.lang.String	UP
/192.168.191.49	java.lang.String	UP
/192.168.191.76	java.lang.String	UP
/192.168.191.74	java.lang.String	UP
/192.168.191.29	java.lang.String	DOWN
/192.168.191.78	java.lang.String	UP

Рисунок 2.11 – Карта станів кільця Apache Cassandra

Скрипт запитує цю сторінку, виконує парсинг HTML-сторінки та рапортує про стан вузлів. При недоступності хоча б одного з вузлів відправляється смс оповіщення команді проекту.

Список серверів, на яких розташовані вузли, заздалегідь задається у скрипті у вигляді списку імен. Далі, за допомогою [System.Net.Dns]::GetHostAddresses обчислюються їх IP-адреси.

Перед запитом веб-сторінок визначається доступність порту на кожному сервері. Створюється об'єкт System.Net.Sockets.TCPClient і з допомогою методу Connect() перевіряємо доступність порту, якого будемо здійснювати звернення методом, представленим на лістингу 3.24.

Лістинг 31 - З'єднання для перевірки доступності порту

```
$tcpClient.Connect($server,8081);
```

Про успішність з'єднання можна судити за якістю об'єкта Connected (тип Boolean).

Якщо значення властивості встановлено \$true, тоді виконується стандартний HTTP-GET запит наступного виду – лістинг 3.25.

Лістинг 3.25 – HTTP-GET запит на карту станів

```
$r = [System.Net.WebRequest]::Create("http://" + $server
+":8081/getattribute?objectname=org.apache.cassandra.net:type=FailureDetect or&attribute=SimpleStates&format=map&template=viewmap ")
```

Відповідь надається вигляд вихідного коду повної коду запитаної веб-сторінки, парсинг якого здійснюється в сценарії. За наявності сервера в стані «DOWN» його IP адреса додається до списку недоступних серверів. Після закінчення перевірки зверненням до смс-сервісу відправляється повідомлення, що містить відомості про недоступні сервери.

Щоразу після успішного завершення перевірки виконується скрипт, який сигналізує, що все кільце знаходиться в робочому стані.

У складі скрипта функцією Test-TeamCityBuildStatus генерується звернення до веб-оболонки TeamCity з використанням API – лістинг 3.26.

Лістинг 3.26 – Звернення до API TeamCity

```
$client = New-Object System.Net.WebClient
$client.Credentials = New-Object System.Net.NetworkCredential
$UserName, $Password $url =
"$ServerUrl/httpAuth/app/rest/builds/?locator=buildType:$BuildT
ypeId&count=1"
$status = $client.DownloadString($url)
```

Звернення повертає статус останнього складання, що перевіряє стан Apache Cassandra. Якщо попередня збірка була невдалою, то кільце кассандри змінило свій стан з недоступного на робоче (рисунок 2.12). Тоді скрипт звертається до смс-сервісу та рапортує команді проекту про вихід Cassandra зі стану DOWN, що забезпечує максимально швидку реакцію на ситуацію з боку команди проекту.



Рисунок 2.12 –Налаштування кроку у конфігурації

3.3 Інвентаризація обладнання

Коли проект має розгорнуту структуру та задіює безліч різноманітного обладнання, а також використовує засоби віртуалізації, слідкувати за розташуванням сервісів та планувати розширення – досить складне завдання.

Додаткова функція, яка реалізована в моніторингу проекту на TeamCity – створення актуального звіту інвентаризації, що містить відомості про розташування ресурсів та найважливіші показники, такі як параметри віртуальних машин, кількість вільного місця на дисках, версії операційних систем, БІОС та іншу необхідну інформацію

Скрипт інвентаризації оперативно надає цю інформацію у вигляді html звіту, який доставляється на пошту, задану як параметр до початку складання.

Скрипт створює набір машин для збору інформації за допомогою модуля Active Directory, а потім збирає інформацію зсередини кожного сервера зі списку.

Для зберігання результатів створюється об'єкт користувача System.Management.Automation.PSObject, властивості якого дописуються по ходу виконання скрипту за допомогою командлета Add-Member, із зазначенням типу додається члена, його імені та значення – лістинг 3.26.

Лістинг 3.26 – Структура об'єкта результату

```
$ReportObject | Add-Member -type noteproperty -name Name -Value
([string]$server)
```

Значення властивостей виходять за допомогою стандартних wmi-опитувань серверів до наступних wmi-класів:

Інформація про роль домену – клас Win32_ComputerSystem, властивість об'єкта DomainRole. Інформація про оперативну пам'ять на хості або віртуальній машині – клас Win32_ComputerSystem, властивість об'єкта TotalPhysicalMemory. Інформація про природу сервера (віртуальний чи фізичний) – клас Win32_ComputerSystem, властивість об'єкта Model. Інформація про модель BIOS – клас Win32_BIOS, властивість об'єкта Name. Інформація про версію прошивки BIOS – клас Win32_BIOS, властивість об'єкту SMBIOSBIOSVersion. Інформація про модель процесора – клас Win32_Processor, властивість об'єкту Name. Інформація про кількість процесорів - клас Win32_Processor, властивість об'єкта NumberOfLogicalProcessors. Інформація про диски та доступний дисковий простір – клас Win32_LogicalDisk, властивості DeviceID, FreeSpace, Size. Інформація про версію встановленої операційної системи – клас Win32_OperatingSystem, властивість Caption. Інформація про тимчасову зону, налаштовану на машині – клас Win32_TimeZone, властивість Caption.

Також збір інформації ведеться рахунок віддалених запитів до реєстру – подібним чином визначається хост віртуальної машини – лістинг 3.27.

Лістинг 3.27 – Віддалений запит до реєстру

```
if ([string]$QueryComputerSystem.Model -eq "Virtual Machine")
{
    [string]$HW_type = ""
    $reg =
        [Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey('LocalMachine'
, $server)
```

```

$regKey=          $reg.OpenSubKey("SOFTWARE\Microsoft\Virtual
Machine\Guest\Parameters")
$PhysHost = $regKey.GetValue("PhysicalHostName")
if ($PhysHost -eq $null){$HW_type = "Virtual Machine:Undefined"}
else {$HW_type = [string]$PhysHost + "'s Virtual Machine"}
)

```

Після чого з одержаних об'єктів оформляється html-документ, зручність сприйняття якого досягається за допомогою стандартної html-розмітки, представленій на лістингу 3.28.

Лістинг 3.28 – HTML-розмітка

```

$style = "<style>"
$style += "BODY{background-color:#121212; text-align:center;}"
$style += "TABLE{border-width: 1px;border-style: solid;border-
color: black; border-collapse: collapse; width=100%;}"
$style += "TH{border-width: 1px;padding: 0px;border-style:
solid;border-color: black;background-color:#A19D72}"
$style += "TD{border-width: 1px;padding: 0px;border-style:
solid;border-color: black;background-color:PaleGoldenrod}"
$style += "</style>"
$Report = $ReportObjects | Sort-Property HW_Type | ConvertTo-
HTML - Head $style

```

Документ зберігається в спеціальну папку з тимчасовими файлами, після чого повністю готовий до відправки. За допомогою командлета Send-MailMessage інформація про інвентаризацію надсилається на вказану в параметрі (Get-Item env:to). Value електронну пошту. Після успішного надсилання згенерований файл видаляється з диска. Звіт повністю представлений у листі, але для зручності роботи його також можна завантажити як вкладення виду, представленого рисунку 2.13.

EDI Hardware

Operating System: Windows

BIOS: Supramicro X86/IT4+P

Name	Role	HW_Type	TotalPhysicalMemory_GB	BIOSName	BIOSVersion	DiskInfo	OperatingSystem
ICAT01	Member Server	ICAT01(Physical Machine)	128	Supramicro X86/IT4+P BIOS Date: 08/03/11 16:04:51 Ver: 2.1a	090016	C: DiskSize: 1,115,650B FreeSpace: 362,250B (32 %)	Microsoft Windows Server 2012 Standard
ICAT02	Member Server	ICAT02(Physical Machine)	128	Supramicro X86/IT4+P BIOS Date: 10/08/2011 Ver: 2.1a	2.1a	C: DiskSize: 1,115,650B FreeSpace: 609,530B (55 %)	Microsoft Windows Server 2012 Standard
ICAT03	Member Server	ICAT03(Physical Machine)	128	Supramicro X86/IT4+P BIOS Date: 10/08/2011 Ver: 2.1a	2.1a	C: DiskSize: 1,115,650B FreeSpace: 397,430B (36 %)	Microsoft Windows Server 2008 R2 Datacenter
EDI-FTP-01	Member Server	ICAT03's Virtual Machine	8	BIOS Date: 03/19/09 22:53:32 Ver: 09.00.04	090004	C: DiskSize: 126,900B FreeSpace: 20,670B (16 %)	Microsoft Windows Server 2008 R2 Standard
ICAT04	Member Server	ICAT04(Physical Machine)	128	BIOS Date: 07/09/13 20:54:38 Ver: 04.06.05	3.0	C: DiskSize: 1,115,650B FreeSpace: 287,230B (26 %)	Microsoft Windows Server 2012 Standard
VM-ICAT-0302	Member Server	ICAT04's Virtual Machine	3	BIOS Date: 09/23/12 17:15:53 Ver: 09.00.06	090006	C: DiskSize: 39,900B FreeSpace: 19,390B (49 %)	Microsoft Windows Server 2008 R2 Standard
VM-ALCO-	Member	ICAT04's Virtual Machine	32	BIOS Date: 05/23/12	090006	C: DiskSize: 320,900B	Microsoft Windows

Рисунок 2.13 – Звіт про інвентаризацію

3.4 Результат впровадження моніторингу

Спочатку, цей моніторинг являв собою кілька простих невзаємопов'язаних скриптів для швидкої перевірки статусу 1-2 важливих показників. На даний момент проект TeamCity зі збірками, що належать до моніторингу, є повноцінною частиною всіх розробок команди. Більше того, деякі сервіси значно знижують або зовсім втрачають свою стійкість до відмов без певних збірок: наприклад, зв'язка для агрегування, сортування та перегляду логів Logstash-Elasticsearch-Kibana, яка переповнюється при відсутності в моніторингу кроку очищення - при певному рівні переповнення спочатку значно збільшується на HTTP-запити для перегляду, а потім, після заповнення диска (як правило, протягом буквально декількох днів), зв'язування взагалі перестає виконувати свої функції.

Для основних фронтів і сервера FTP також важливим є скрипт щодо відстеження дискової черги – це дозволяє вчасно помітити проблему значного підвищення навантаження та заздалегідь запланувати заходи щодо модернізації обладнання або перерозподілу ресурсів.

Розроблена система дозволяє дізнаватися про проблеми в інфраструктурі раніше чи одночасно з користувачами. Це, по суті, комплекс швидкої діагностики, який дає своєчасне оповіщення про труднощі у роботі сервісів та обладнання, надає точну інформацію, конкретні відомості про проблему та її характер.

Якщо спочатку інформація про показники просто збиралася і давала лише загальне уявлення, після детального аналізу системним адміністратором якого можна було припустити наявність проблеми певного характеру, то тепер спеціалізоване підстроювання моніторингу під потреби проекту дає можливість зробити точну оцінку максимально швидко незалежно від досвіду системного адміністратора, у роботі з цим проектом.

Моніторинг використовує конфігураційну базу даних, де кожен показник - це одиниця, що зберігається в централізованій базі. Така вистава з використанням TeamCity дозволила «з коробки» інтегрувати систему моніторингу з сервіс-диском і, за потреби, можна розширювати функціонал далі. Використовуючи візуальне уявлення, що включає вбудовані графіки та статистику, аналітичний відділ може оцінити стійкість до відмови сервісів.

Збільшився рівень автоматизації завдань: для перезапуску сервісних служб-завдань, що зупинилися з якоїсь причини, більше не потрібна участь системного адміністратора. Вище також описано автоматичне очищення логів та відправлення оповіщень. Адміністратори звільнилися від рутинної роботи і змогли сконцентруватися на найважливіших завданнях. Помітно мінімізований час простою, що сприяло підписанню SLA. Моніторинг дозволяє організації "вести рахунок" своєї благополучної діяльності з надання ключових послуг користувачам та клієнтам та через це отримувати впевненість та напрацьовувати авторитет.

Моніторинг ведеться незалежно від рівня віртуалізації: раніше інфраструктура була фізична, тобто, всі сервери були реальне устаткування. Зараз інфраструктура часто складається із віртуальних машин, коли сервер фізично один, але на ньому працює кілька віртуальних. Зазвичай для моніторингів це було проблемою та вимагало тонкощів у налаштуванні, але розроблена система дозволяє

знімати всі значення лічильників незалежно від рівня віртуалізації обладнання проекту.

Впровадження даної системи, і навіть сама розробка її функціональності відбувалася з виникнення проблем у спостережуваних проектах, тобто поетапно, у міру інтеграції систем немає ніяких неузгодженостей.

У зв'язку з тим, що моніторинг побудований на основі продукту, з яким аналітики та розробники вже мали величезний досвід роботи, витрати часу на їх взаємодію із засобами системи були мінімальними. Внаслідок інтеграції з уже існуючою та налаштованою системою, делегування прав на редагування збірок моніторингу, перегляд відомостей про стан системи та інші операції мало мінімальну трудомісткість.

Своєю чергою виявилось кілька побічних позитивних ефектів. Розробникам проекту відомості, наприклад, про періодичну зупинку сервісів, допомагають дізнатися про проблеми – «баги» в коді програми або про виникнення необроблених винятків під час роботи сервісу. Адміністратори ж, з історії показників, що спостерігаються, можуть знайти «пляшкові шийки» системи, планувати поліпшення пропускнуєї спроможності системи, і необхідні модернізації для відповідності заявленому рівню обслуговування клієнтів. Одночасно можна точно визначити де модернізація не потрібна, що дозволяє заощадити кошти проекту за допомогою планування потужностей.

Сила пристрою моніторингу у вигляді рукописних модулів, інтеграція зі сторонніми бібліотеками та сервісами (наприклад, для відправки смс-повідомлень) виявилася легким завданням. За рахунок наявності рукописних скриптів немає прив'язки системи конкретної ОС – скрипт легко модернізується адміністратором та підлаштовується під необхідну операційну систему. Модульність дозволяє легко і швидко змінювати конфігурацію відстеження: у бік збільшення функціональності за рахунок додавання кроку конфігурації, і позбавлятися непотрібних функцій одним клацанням миші завдяки можливостям TeamCity.

Структура модулів, а точніше те, що вони повністю розробляються вручну, хоч і з використанням загальнодоступного функціоналу, надає можливість збору

відомостей абсолютно будь-якого характеру, ідеально вибудовуючи моніторинг під потреби замовника. Це дозволяє миттєво визначати характер проблеми, що виникла, а отже значно знизити час простою і миттєво визначити відповідального за вирішення завдання спеціаліста.

Ідеальна відповідність моніторингу завданням дозволяє простежувати вплив збоїв в ІТ-ресурсі на надання сервісів, що дозволяє оптимально встановлювати пріоритети у роботах з усунення збоїв.

Позитивним моментом виявилася практична «безагентність» (єдиний необхідний агент для збирання та моніторингу всіх систем може стояти прямо на сервері TeamCity) моніторингу, а також простота розширення системи – додаткову службу агента можна встановити на будь-якому устаткуванні і вона автоматично інтегрується з усім сервісом, що збільшить продуктивність рахунок одночасного виконання збірок різних агентів.

ВИСНОВКИ

Бізнес у будь-якій сфері сильно зав'язаний на доступність та працездатність його IT-інфраструктури 24/7/365. Щоб забезпечити цю працездатність, необхідно заздалегідь виявляти вузькі місця у конфігурації систем та мереж, а також швидко впізнавати про наявності поломки та її причини. Для цих потреб у компаніях, де подібне стеження неможливе за рахунок лише фахівців, прийнято використовувати системи моніторингу.

Системи моніторингу бувають платні і вільно розповсюджуються, а також розрізняються за своїм наповненням «з коробки», масштабованістю, необхідними ресурсами та рівнем знань, необхідним для прийнятного налаштування.

При виборі, розробці, впровадженні систем моніторингу спочатку потрібно визначитися з об'єктами, які будуть підлягати спостереженню, а також критичні події та показники, які визначають кількість оповіщень при поломці, частоту сканування та інші параметри та наслідки. Причому оцінювання показників насамперед потрібно здійснювати не з погляду технічного інженера, і з погляду кінцевого користувача.

Хід виконання роботи повністю відповідав поставленим у запровадженні завданням. Для початку були висунуті загальні вимоги до моніторингу проектів, що підтримуються, володіючи списком яких можна було приступити до пошуку відповідного готового рішення.

Потім докладно розглянуто різні існуючі системи моніторингу серверів від широкого різноманіття фірм-розробників, проаналізовано їх позитивні та негативні сторони, відповідність сформульованим раніше вимогам.

У результаті, зроблено висновок про неможливість чи неоптимальності використання вищерозглянутих існуючих систем. Після чого критерії відповідності моніторингу поставленим завданням у проекті були максимально

уточнені, завдяки чому став можливий вибір оптимальної основи для його створення – вже використовуваного для інших потреб розробки пакету TeamCity.

У міру виникнення проблем та нових завдань у проекті розроблялися різноманітні модулі, вівся пошук додаткових бібліотек та способів інтеграції. Існуючими сервісами як максимальної зручності написання скриптів, так відстеження специфічних показників. Паралельно здійснювалося поетапне тестування створених скриптів та його впровадження у експлуатацію. У результаті розроблений моніторинг став невідривною частиною проектів, що підтримуються, що забезпечує високу доступність, відмовостійкість та інформацію про стан систем.

Під час використання створеного моніторингу відділами адміністрування та підтримки серверів, розробки програмного забезпечення та підрозділом аналітиків було відзначено низку позитивних аспектів впливу на проект, а також несподівані плюси продукту:

- підвищення стійкості до відмов сервісів, у тому числі шляхом інтеграції з ними;
- система дозволяє чітко визначати і планувати шляхи модернізації проекту та її інфраструктури, шукати «пляшкові шийки»;
- максимально швидко і точно інформує про можливу або проблему, що виникла;
- використання бази даних моніторингу для різних зовнішніх потреб (наприклад, звіти аналітичного відділу);
- делегування прав "з коробки";
- простота інтеграції з різними зовнішніми бібліотеками та сервісами;
- автоматизація рутинних завдань системного адміністратора;
- мінімізація часу простою як наслідок, підписання SLA, тобто. вихід на новий рівень обслуговування клієнтів;
- незалежність від віртуалізації, програмного забезпечення та інших факторів, можливість моніторингу специфічних показників;

- мінімальний час навчання роботи із системою;
- простота делегування прав доступу та дозволів;
- виявлення «багів» та необроблених винятків у коді завдяки звіту про стан сервісів;
- простота налаштування під конкретний стан інфраструктури за допомогою вбудованих функцій TeamCity та спеціальних параметрів;
- повна історія доступності сервісів та зміни показників;
- простота застосування та масштабованості.

Загалом пропонований продукт для моніторингу серверів і служб, а також автоматизації завдань та інвентаризації допоможе значно знизити час простою сервісів, підвищити якість послуг, що надаються клієнту компанією, а також значно покращити продуктивність, уникнувши при цьому невиправданих вкладень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. D. Jones Creating Unified IT Monitoring and Management in Your Environment/ D. Jones: Realtime Publishers, 2012 – 92с
2. Alistair Croll Complete Web Monitoring/Alistair Croll, Sean Power: O'Reilly Media, 2009 – 672 с.
3. Andrea Dalle Vacche Zabbix Mastering. Andrea Dalle Vacche, Stefano Kewan Lee: Packt Publishing Ltd, 2013 – 358 с.
4. Nagios Nagios Documentation. URL: <http://www.nagios.org/> (Last accessed: 30.10.2023).
5. Zabbix URL: What is Zabbix – <http://www.zabbix.com> (Last accessed: 30.10.2023).
6. TechNet Magazine, System Center Operations Manager URL: <https://www.technetmagazine.com/> (Last accessed: 30.10.2023).
7. 2018: Простота розширення можливостей моніторингу – URL: <http://technet.microsoft.com> (Last accessed: 30.10.2023).
8. N. Berardi. Your First Fluent Cassandra Application- URL: <http://coderjournal.com/2010/06/your-first-fluent-cassandra-application/>(Last accessed: 30.10.2023).
9. CQL Scepification. CQL BINARY PROTOCOL v1 specification URL: https://github.com/apache/cassandra/blob/cassandra-1.2/doc/native_protocol.spec (Last accessed: 30.10.2023).
10. Cassandra Wiki: The Cassandra Thrift API Overview URL: <http://wiki.apache.org/cassandra/API10> (Last accessed: 30.10.2023).
11. Википедия: Apache Thrift URL: http://en.wikipedia.org/wiki/Apache_Thrift (Last accessed: 30.10.2023).
12. Elastic Search: Elastic Search Guide URL: <http://www.elasticsearch.org/guide/> (Last accessed: 30.10.2023).
13. Solarwinds: Server & Application Monitor URL:

<http://www.solarwinds.com/server-application-monitor.aspx> (Last accessed: 30.10.2023).

14. Zabbix Documentation: Zabbix documentation for version 2.0. URL: <https://www.zabbix.com/documentation/2.0> (Last accessed: 30.10.2023).

15. ITFB: Моніторинг серверів, сервісів і сайтів URL: <http://www.itfb.com.ua/monitoring.html> (Last accessed: 30.10.2023).

16. Jet Brains: TeamCity User Guide URL: <http://www.jetbrains.com/teamcity/documentation/userguide.jsp> (Last accessed: 30.10.2023).

17. microsoft.com: Introduction to the Deployment Guide for Monitoring Server - URL: [http://technet.microsoft.com/library/dd630725\(v=office.12\).aspx](http://technet.microsoft.com/library/dd630725(v=office.12).aspx) (Last accessed: 30.10.2023).

18. Oreilly: Could They Do It?: Real User Monitoring URL: <http://oreilly.com/web-development/excerpts/9780596155131/chapter-10.html> (Last accessed: 30.10.2023).