Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Дипломна робота
бакалавра
з теми: **«РЕАЛІЗАЦІЯ СИСТЕМИ МОНІТОРИНГУ КОМП’ЮТЕРНИХ МЕРЕЖ НА ОСНОВІ GRAFANA»**

Виконала: студентка 4 курсу, групи KN1-B18
спеціальності: 122 Комп’ютерні науки
Островська Каріна Миколаївна

Керівник: Понеділок В.В.
старший викладач кафедри комп’ютерних наук, кандидат технічних наук

Кам’янець-Подільський – 2022

ЗМІСТ

ВСТУП

Актуальність дослідження. На сьогоднішній день хмарні та веб додатки займають істотну частину у суспільстві. Такі додатки охоплюють різні сфери діяльності від продажів до високотехнологічних підприємств. Деякі сфери діяльності, критичні до безперебійності та якості сервісу. Прикладом такої сфери є телемедицина – коли лікар стежить за станом пацієнта чи спеціальних пристроїв, які здійснюють лікування чи діагностику пацієнтів, у режимі реального часу. Однак відповідальність за збої програми лежить не на лікаря, так як він є лише користувачем системи, а на розробнику цієї програми. Щоб уникнути таких проблем, а відтак і фінансових втрат, слід застосовувати комплекс заходів для моніторингу програми.

Мета роботи

Система моніторингу дозволяє відстежувати доступність компонентів програми, що збирає показники в режимі реального часу. Крім цього, можливо генерувати різні звіти, наприклад, використання ресурсів: завантаження процесора, пам'яті, жорсткого диска та ін. В разі виявлення перевантаження якогось сервера, можна зробити перерозподіл потужносте.

Аналіз показників системи можливий завдяки наявності лог-записів у журналі програми, у разі хмарної програми з мікро сервісною архітектурою – у журналі окремих сервісів. Тому кожен додаток має мати систему логів. Більш того, лог-файли повинні бути легко доступні та читані. Як правило, звернення до журналу логів відбувається після несподіваної помилки в результаті виконання програмного коду, щоб з'ясувати проблему та констатувати несправність, локалізувати місце (клас, сервіс, інтерфейс), де код виконався некоректно. Відповідно, іноді необхідно перемикатися між різними журналами, щоб простежити шлях помилки від початку до кінця – для локалізації первісного походження. Варто зазначити, що файли логів це слабо структуровані дані. Тому, вивчати величезний масив таких файлів нелегке завдання, що слід враховувати при виборі інструментів для аналізу логів та моніторингу.

РОЗДІЛ 1. ХМАРНІ ПРОГРАМИ

Хмарні програми виробляють всі обчислення на віддалених ресурси, що надаються користувачеві через інтернет як онлайн сервіс. Відповідно до термінології IEEE, хмарні технології – це парадигма, що дозволяє постійно зберігати інформацію користувача на інтернет серверах і лише тимчасово кешуючи її на стороні користувача . Існує кілька типів хмарних додатків, розглянемо притаманний даної предметної області – SaaS (від англ. System as a service). Цей тип надає доступ кільком клієнтам до єдиного додаток з за допомогою браузера. Компанія розробник самостійно керує додатком та надає доступ до платформи через інтернет . Для обох сторін є свої позитивні моменти. Для клієнта вигода полягає в економії на обладнання - немає необхідності купувати дороге високопродуктивне обладнання та програмне забезпечення. Для Розробник модель SaaS дозволяє ефективно боротися з неліцензійним використанням програмного забезпечення, в силу того, що фактично софт не потрапляє до кінцевих замовників. Крім цього, ця концепція знижує витрати на розгортання та впровадження систем технічної та консультаційної підтримки продукту, хоч і не виключає їх повністю. Хоча для користувача це і виглядає як одна єдина програма, найчастіше його внутрішня архітектура, з таким типом, вигладить як набір окремих сервісів, що виконують строго конкретне завдання, у своєму процесі та оточенні. Взаємодія між елементами програми та зовнішніми компонентами здійснюється, як правило, за протоколом HTTP. Даний архітектурний підхід називається мікро сервісним. Як було сказано вище, ідея полягає в тому, щоб розділити програму на набір невеликих взаємопов'язаних сервісів, а не розробляти одне монолітний додаток. Замість спільного використання однієї схеми бази даних з іншими сервісами, кожен мікро сервіс має власну бази даних. З одного боку, цей підхід суперечить ідеї моделі даних для всієї програми. Крім того, це часто призводить до дублювання даних. Однак наявність власних окремих баз даних забезпечує слабкий зв'язок. На *Рис.1.* представлена узагальнена схема програм з мікро сервісною архітектурою.

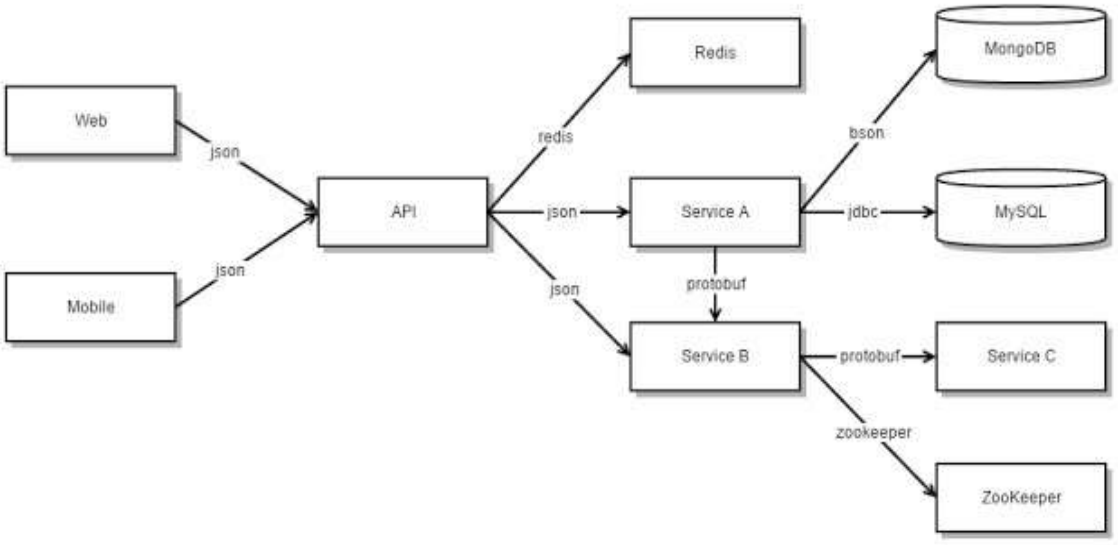

Рис.1. Узагальнена схема хмарної програми з мікро сервісною архітектурою

Кінцеві користувачі не мають прямого доступу до внутрішніх сервісів. Натомість зв'язок здійснюється через посередника, відомого як шлюз. API-шлюз відповідає за такі завдання, як балансування навантаження, кешування, контроль доступу та моніторинг.

1.1. Предметна область програми

Система моніторингу, що розробляється, призначається для хмарного застосування лікаря-сомнолога. З огляду на комерційну таємницю та аспекти безпеки користувачів, технічні деталі та назва не розкривається.

Додаток дозволяє лікареві стежити за стоянням пацієнтів з розладами сну дистанційно. Кожен пацієнт має спеціальний пристрій, який реєструє плісомнографічні сигнали під час сну. Потім сигнали надходять у систему через інтернет, піддаються аналізу та зберігаються, розподіляючись за різними мікро сервісами системи.

Крім складного технологічного потоку обробки медичних даних, додаток присутні стандартні елементи та функції такі як, пошук, пагінація, фільтри, авторизація тощо. Ці функції теж повинні піддаватися моніторингу, для виявлення затребуваності певних характеристик та оцінки навантаження на програму в цілому.

1.2. Система логів хмарної програми

Кожен мікро сервіс записує лог-інформацію в окремий файл, що у разі ручної обробки логів лише ускладнює процес.

Однак для швидкого реагування на надзвичайні помилки та моніторинг лог-файлів у додаток використовується ELK – стек технологій, що складається з Elasticsearch, Logstash та Kibana.

Менеджери з якості стежать за сплесками помилок у системі після чергового релізу та передають інформацію про це у відділ розробки. Ті, своєю чергою, знаходять кореляції з помилками у додатку, відразу бачать стеки викликів функцій та інші деталі, необхідні налагодження. На *Рис.2.* наведено приклад лог записів системи.

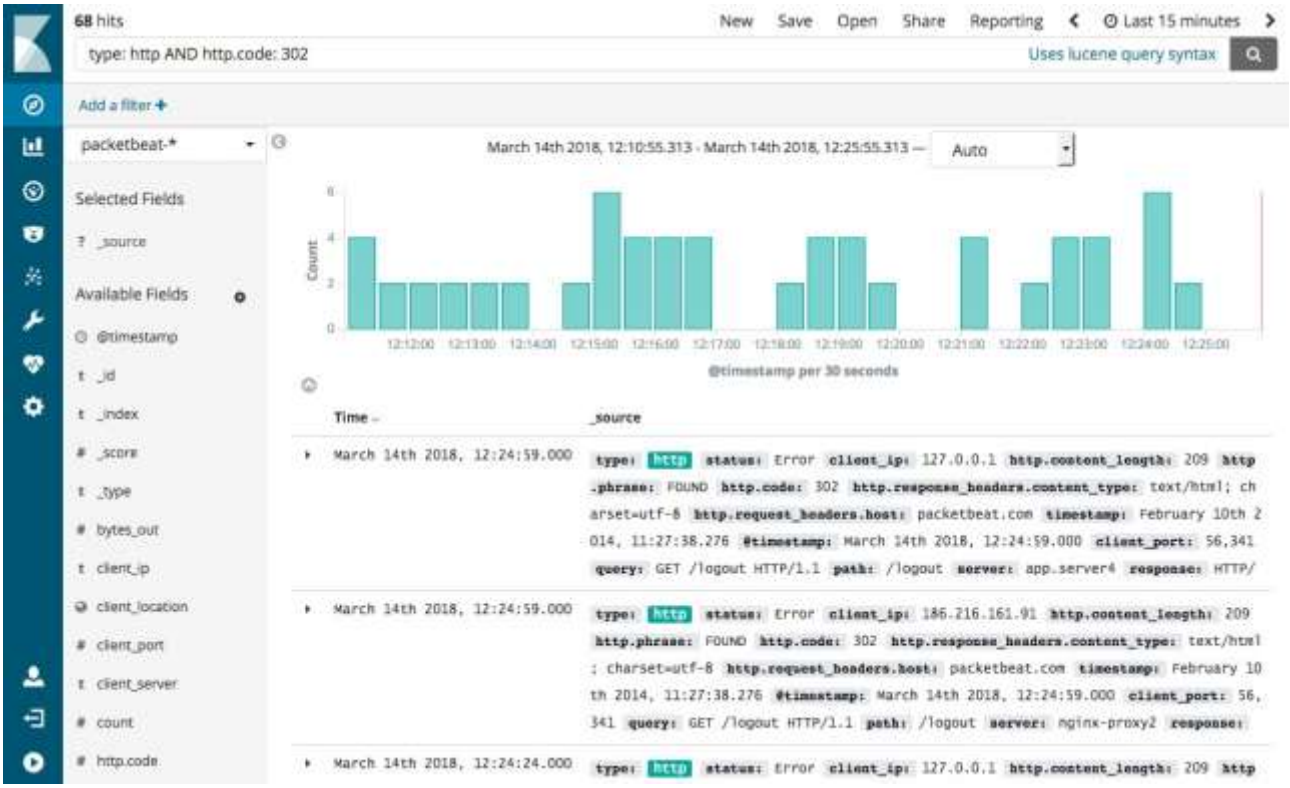

Рис.2. Приклад логів у сервісі Kibana

1.3. Бізнес-метрики

Цей блок призначений для відображення фактичної інформації програми хмари. Наприклад, кількість зареєстрованих користувачів, авторизації у системі, затребуваність тієї чи іншої фільтра під час пошуку, як і коли була отримана

статистика з приладів щодня та інші, специфічні для предметної області, показники.

Кінцева мета цього набору метрик необхідна для кращого розуміння користувачів системи, щоб у подальшому модернізувати той чи інший бізнес-процес. Тобто бізнес-метрики дають інформацію про використання конкретних функцій програми, наприклад, авторизація за допомогою другого фактора, генерація звітів тощо.

Якщо яка-небудь функція має успіх, то варто задуматися про те, як зробити її краще і легше в плані використання. І навпаки, якщо функція мало затребувана, то не варто продовжувати подальшу детальну розробку, а зупиниться на поточному стані.

Крім цього, бізнес-метрики говорять не лише про поведінку кінцевих користувачів, а й характеризують кількісні властивості програми. Наприклад, якщо ми знаємо, що у нас в системі зареєстровано велику кількість організацій, то при реалізації нової функції варто задуматися про архітектуру, чи це не додасть додаткового навантаження на згадку під час роботи програми і чи варто оптимізувати існуючі ділянки коду.

1.4. Метрики апаратних ресурсів

Моніторинг серверів, як фізичних, і віртуальних показує те, наскільки коректно і правильно додаток було реалізовано з погляду пам'яті, процесорного часу тощо. Якщо в додаток є вузькі місця, то ця група метрик дасть негайну відповідь про таку проблему. Наприклад, при пошуку пацієнтів може статися ситуація, коли код виконується не на боці СУБД, а весь масив даний завантажується на згадку і надалі здійснюється робота в оперативній пам'яті. Такий підхід миттєво поглинає всю доступну пам'ять на віртуальній або фізичній машині та програма уповільнює свою роботу, а іноді й зовсім завершується аварійно.

Тому щоб розуміти систему з погляду ресурсів, необхідно стежити за кожним сервісом окремо, скільки ресурсів кожного ресурсу він споживає і за перевищення певного порогового значення сповістити про це розробників.

1.5. Поточний стан моніторингу системи

Звичайно, без будь-якого моніторингу неможливо підтримувати хмарну програму. Тому програма, для якої проектується система моніторингу в даній роботі, мала панель доступності сервісів зображена на *Рис.3*.

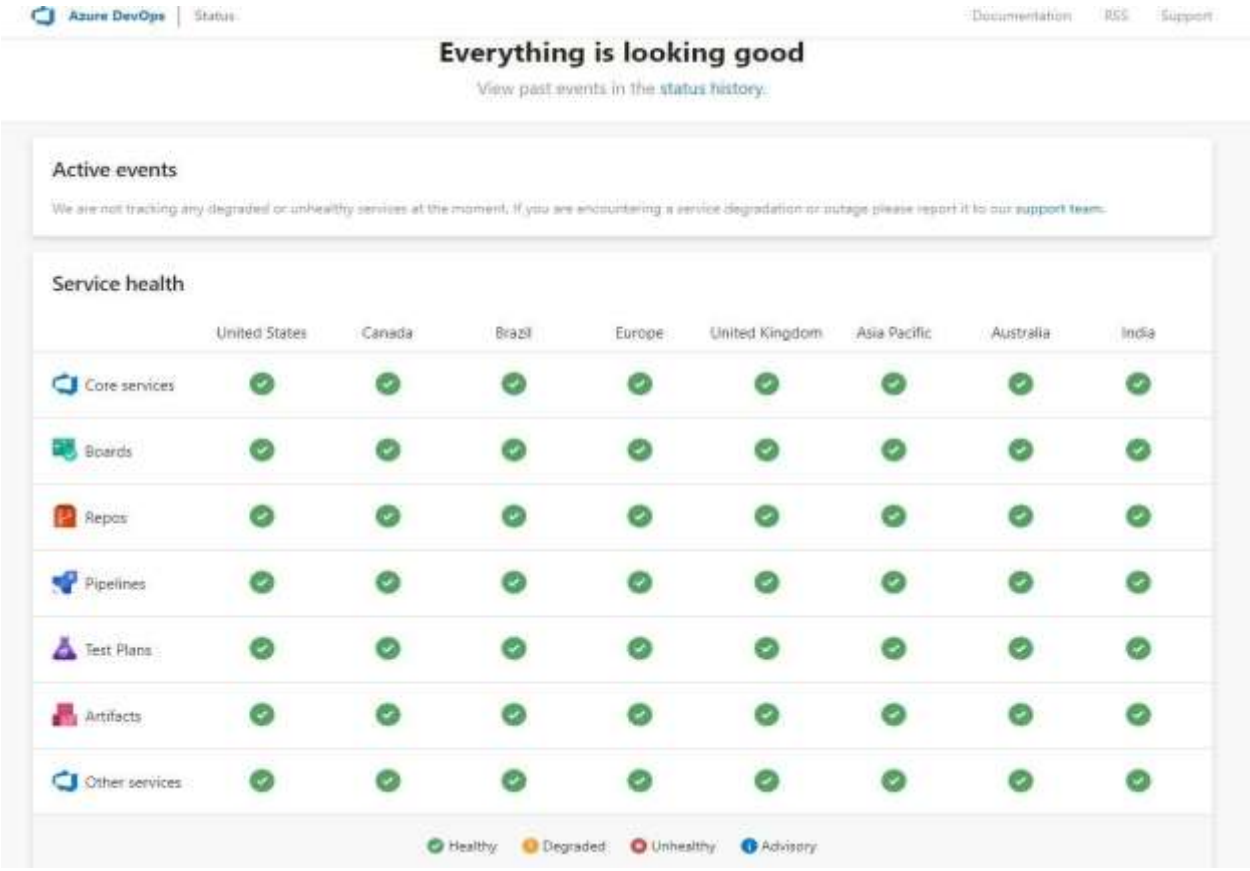

Рис.3. Панель доступності сервісів Azure DevOps

Що стосується бізнес-метрик та аналітики апаратних ресурсів – то моніторинг здійснювався системними адміністраторами на запит та за допомогою ручного доступу. Що говорить про незручність та небезпеку даного підходу. Більше того, монітор доступності мав проблему з переповненням кешу і зависанням, що виражалося відображенням неактуальної інформації.

Проте система логів налаштована та функціонує стабільно з використанням ELK – стека.

ELK включає такі компоненти як:

• Elastic Search. NoSQL-сховище та пошукова система, заснована на мові Lucene;

• Logstash. Місце імпорту, з безліччю змін, якою дані потрапляють у сховище Elastic Search;

• Kibana. Веб-інтерфейс візуалізації даних із Elastic Search.

Додавання логів Logstash відбувається через додаткову утиліту з набору ELK - logstash-forwarder (beats). Цей інструмент є окремою службою, яка стежить за змінами файлу на диску і додає їх до Logstash. Після цього кожен рядок файлу розпізнається та відправляється у вигляді полів для індексування та тегів в Elastic Search. Результат такого перетворення можна побачити *Рис. 4.*

Рис.4. Панель доступності сервісів Azure DevOps

РОЗДІЛ 2. ПРОЕКТУВАННЯ

По-перше, визначимося з критеріями моніторингу, щоб виділити характерні риси, які б визначити набір інструментів.

Система моніторингу та візуалізація даних має відповідати наступним вимогам:

- працювати у режимі реального часу;
- бути гнучким у налаштуванні нових метрик;
- використовувати поточні лог – файли (можливість
- підключення/імпорту даних з Logstash, ElasticSearch, filebeat, kibana);
- масштабуватись на кілька серверів;
- стабільно працювати;
- повідомляти про аномалії в месенджер компанії;
- мати можливість моніторингу Docker оточення та системних ресурсів;
- мати Time Series базу даних;
- мати можливість написання власних плагінів однією зі знайомих мов (javascript, C#, F#);
- бути безкоштовною та з відкритим вихідним кодом.
- Метрики повинні відображати:
- моніторинг апаратних ресурсів;
- кількість користувачів у системі;
- кількість пацієнтів у системі;
- кількість організацій у системі;
- кількість пристроїв (можливий поділ на терапевтичні, вентиляційні);
- спроби авторизації;
- загальну статистику звернення до системи;
- відображати працездатність сервісів;
- актуальний стан сервера безперервної інтеграції та результатів розгортання сервісів.

2.1. Вибір інструментів моніторингу

Були розглянуті такі інструменти для засобів моніторингу: Cacti, Ganglia, Collectd, Graphite, Zabbix, Nagios, Icinga.

Вищезазначені компоненти гідні поваги завдяки легкій установці та підтримці windows і linux – контейнерів, величезному відсотку використання в реальних додатках на багатьох серверах і компаніях. Але всі вони застарілі, хоча все ще підтримуються, але дуже слабо розвиваються і мають застарілий інтерфейс.

У більшості з них використовуються sql-бази даних, що не є оптимальним для зберігання історичних даних (метрик). Це здається універсальним, але з іншого боку — бази даних створюють велике навантаження на диски. У цьому розмірність даних велика. Для таких завдань більше підходять сучасні бази даних часових рядів, такі як

ClickHouse. Системи моніторингу нового покоління використовують бази даних тимчасових рядів, одні з них включають їх у свій склад як невіддільну частину, інші використовують як окремий компонент, а треті можуть працювати без баз даних

До останньої групи, сучасних засобів моніторингу слід віднести такі продукти як NetData, Prometheus, InfluxDb, Grafana, Telegraph. Незважаючи на те, що в компанії вже використовується ELK - стек, який дозволяє налаштувати візуалізацію в сервісі Kibana, вибір був зроблений на зв'язку nodex_exporter, Prometheus, Grafana.

Kibana, що входить у стек ELK, добре дозволяє шукати детальну інформацію по лог-файлах, візуалізувати дані за останні кілька днів, але при реалізації невеликої панелі метрик, з агрегацією – система стала відчутно повільніше працювати. Тому вирішили виділити окрему інфраструктуру під візуальний моніторинг у вигляді Grafana та Prometheus.

2.2. Prometheus

Prometheus – це комплексне рішення, що включає фреймворк для

моніторингу і власну темпоральну базу даних. Це досить молодий продукт, пер–ший реліз якого відбувся у 2016 році.

Архітектура. Prometheus включає набір наступних компонентів:

- Сервер. Функція якого це отримання та збереження метрик у темпоральній (time series) базі даних;
- Набір клієнтських бібліотек для різних мов
- програмування (Haskell, Java, Python, Go, Ruby, Node.js, .NET/C#);
- Pushgateway – компонент для прийому метрик короткочасних процесів;
- PROMDASH – дашборд для метрик;
- інструменти для експорту даних із сторонніх додатків (Statsd, Ganglia, HAProxy та інших);
- менеджер повідомлень;
- клієнт командного рядка для виконання запитів до даних.

Всі компоненти Prometheus взаємодіють між собою за протоколом HTTP. Схема комунікації зображено *Рис. 5*.

Рис. 5. Схема компонентів Prometheus

Ядром системи є сервер Prometheus. Його робота здійснюється автономно. Є локальна база даних для зберігання. Виявлення інших послуг відбувається автоматично, що значно полегшує процедуру конфігурування і розгортання: для спостереження однією сервісом досить встановити лише сервер і необхідні

компоненти збору та експорту метрик. Існує набір готових компонентів під моніторинг існуючих сервісів таких, як: Haproxy, Docker, PostrgreSQL і т.д.

Є два способи отримання метрик у Prometheus – push та pull. Назви кажуть самі за себе. Pull – сервер Pometheus робить запит до сервісу, тобто підтягує дані. Проте, передбачено й іншу можливість отримання механізм push. Для такого підходу потрібен спеціально призначений компонент – pushgateway. Це необхідно у випадках, коли отримання метрик методом pull із будь-яких причин неможливо. Наприклад, коли сервіс захищений фаєрволом або знаходиться у приватній закритій мережі. Також механізм push може бути корисним при спостереженні за сервісами, що підключаються до мережі періодично та на нетривалий час, дані надходять у міру надходження на сервер автоматично. Модель даних Prometheus вимагає від метрик, що надходять. уявлення у вигляді часових рядів (time series). Тому всі метрики зберігаються у своїй темпоральній БД. Для зберігання індексів використовується LevelDB.

2.3. Grafana

Grafana є відкритим (Apache 2.0) веб-інтерфейс до різним темпоральним СУБД, таким, як Graphite, InfluxDB, і, само собою Зрозуміло, Prometheus. Загалом, Grafana будує графіки, використовуючи інформацію з Prometheus або Elastic Search. Незважаючи на те, що у Prometheus є власний веб-інтерфейс, який дуже мінімальний і Досить незручний, використовується візуалізація за допомогою Grafana. Терміни Grafana. Панель – базовий елемент візуалізації вибраних показників. Grafana підтримує панелі з графіками, одиничними статусами, таблицями, тепловими картами кліків та довільним текстом, а також інтеграцію з офіційними та створеними спільнотою плагінами (наприклад, карта світу або годинник) та програмами, які також можна візуалізувати. Можна налаштувати стиль та формат кожної панелі; всі панелі можна перетягувати на нове місце, перебудовувати та змінювати їх розмір. Дашборд - набір окремих панелей, розміщених у сітці з набором змінних (наприклад, ім'я сервера, програми та

датчика). Змінюючи змінні, можна перемикати дані, що відображаються на дашборді (наприклад, дані із двох окремих серверів). Усі дашборди можна налаштовувати, а також секціонувати та фрагментувати представлені в них дані відповідно до потреб користувача. У проекті Grafana бере участь велика спільнота розробників коду та користувачів, тому існує великий вибір готових дашбордів для різних типів даних та джерел. У дашбордах можна використовувати інструкції для відображення певних подій різних панелях. Анотації додаються настроюванні запити в Elasticsearch; на графіку інструкція відображається вертикальну червону лінію. При наведенні курсору на інструкцію можна отримати опис події та теги, наприклад, для відстеження відповіді сервера з кодом помилки 5xx або перезапуск системи. Завдяки цьому можна легко зіставити час, конкретну подію та її наслідки у додатку та досліджувати поведінку системи.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ МОНІТОРИНГУ ЗА ДОПОМОГОЮ GRAFANA LIVE

У світі робототехніки та автоматизації однією з найбільш постійних потреб є збирання та візуалізація даних у реальному часі з апаратних компонентів, таких як датчики та приводи, які дають уявлення про те, як система поводиться в цілому, і допомагає діагностувати будь-які потенційні проблеми, які можуть виникають понаднормово.

Історично склалося так, що для того, щоб передавати такі дані на інформаційні панелі в рамках типових інструментів аналітики та візуалізації, включаючи Grafana, вимагалося чимало зусиль для спеціальної розробки зі складними інструментами. Але з випуском Grafana 8.0 стало набагато простіше передавати та візуалізувати дані в реальному часі за допомогою нового потокового API, який був представлений як частина функції Grafana Live.

(Примітка: під «реальним часом» ми маємо на увазі «м’який режим реального часу», оскільки через затримки в мережі, цикли збирання сміття, обмеження протоколу тощо затримки доставки повідомлень можуть становити кілька сотень мілі секунд.)

Тут ми розглянемо одне з таких застосувань абсолютно нових потокових можливостей Grafana в контексті сенсорної системи, яка спілкується через протокол обміну повідомленнями MQTT, використовуючи API потокової передачі Grafana Live у внутрішньому плагіні джерела даних.

3.1. Налаштування

MQTT — це легкий мережевий протокол публікації-підписки, який зазвичай використовується для обміну повідомленнями між пристроями через TCP/IP. Спочатку розроблений IBM, протокол MQTT набув широкого поширення для між машинного зв’язку в середовищах з обмеженою пропускною здатністю, де компактне транспортування даних є суворою вимогою.

Grafana Live — це інтегрований механізм обміну повідомленнями в реальному часі, вбудований у Grafana, який був представлений як частина випуску v8.0. Він заснований на структурі публікації-підписки, яка спілкується за протоколом WebSocket і дозволяє передавати будь-які дані про події до інтерфейсних клієнтів, як тільки вони виникають.

Для збирання даних використовується сенсорна плата заснована на дуже широко використовуваному, універсальному і готовому в продажу IMU, BMI160 від Bosch Sensortec. Це компактний, малопотужний і малошумний 6DOF IMU, який зазвичай зустрічається в смартфонах і носимих пристроях.

Щоб отримати дані датчика потокової передачі в екземпляр Grafana на окремій машині, ми встановили та використали плагін Grafana MQTT Datasource на цій машині.

Плагін використовує інтегрований клієнт MQTT для підписки на теми MQTT та API потокової передачі Grafana Live для публікації їх у потоці подій у Grafana. На *Рис.6*. зображено підключення до пристрою через MQTT протокол.

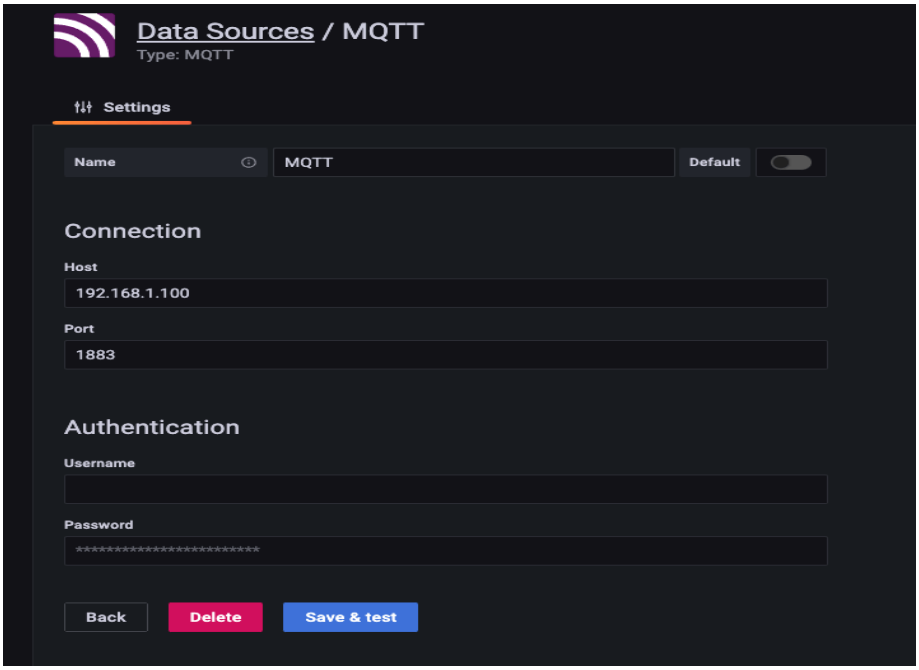

Рис.6. підключення через MQTT протокол

Для візуалізації інформації необхідно добавити дашборд. Є декілька варіантів як це можна зробити: створивши самому, використати готовий з бібліотеки дашбордів. На *Рис. 7.* зображено меню дашбордів.

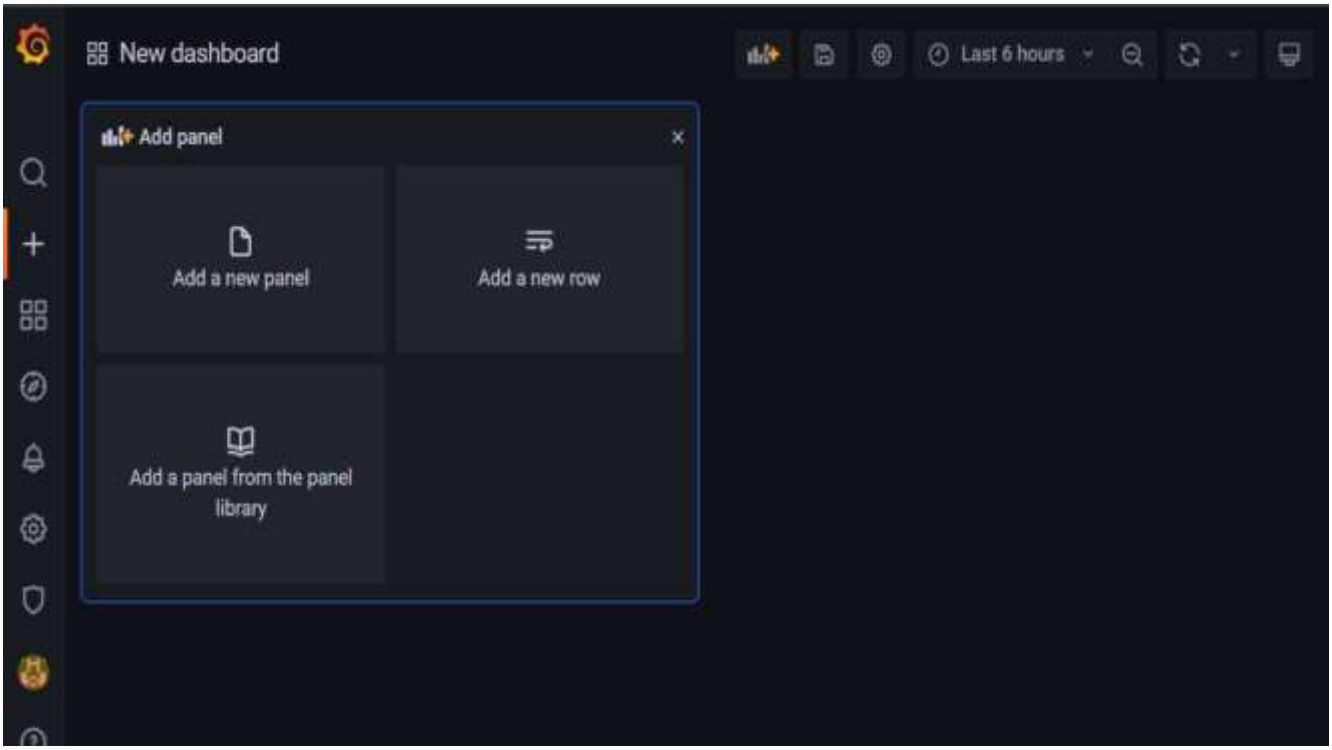

Рис. 7. Меню дашбордів

Розглянемо приклад реалізації моніторингу 6 датчиків струму які умовно позначені L1-L6. Перевіримо як змінювалася сила струму впродовж 24 годин. Моніторинг зображено на *Рис. 8.*

Рис. 8. Моніторинг датчиків струму

Також скориставшись іншими дашбордами ми можемо з легкість побачити момент навантаження на датчик. Розглянемо подібний приклад з 6 датчиками струму які зображені на *Рис. 9.*

Рис.9. Моніторинг датчиків струму з навантаженням.

На даному рисунку ми бачимо що датчик L1 та L2 потребують уваги. Вони знаходяться уже в жовтому рівні навантаження. Це надає нам змогу заздалегідь передбачити збій в роботі пристрою.

Панелі моніторингу можуть бути більш розширеними якщо для прикладу нам потрібно переглядати не лише силу струму а і температуру або ж навіть тиск на пристрій. Розглянемо моніторинг кондиціонеру який зображений на *Рис. 10*.

Рис.10. Моніторинг кондиціонеру.

Як бачимо все розміщено поряд для зручного перегляду. Моніторинг відбувається в реальному часі можна обирати різні проміжки часу від одного місяця до однієї години. Використання джерела даних MQTT є простим і полегшує створення діаграм у реальному часі.

РОЗДІЛ 4. АРХІТЕКТУРА

Архітектура системи моніторингу завдяки підтримці контейнерів дозволяє розміститися на одній віртуальній машині. Це є величезним плюсом та економить ресурси, як фізичні, так і матеріальні. Більш того при необхідності та можливості можна розмістити на одній віртуальній машині як сама програма, так і систему моніторингу. Бо це тільки етап проектування, то зв'язки є умовними. Цілком ймовірно, що в ході реалізації додаються нові компоненти або види зв'язків заміняться необхідності. Із раніше описаних елементів системи моніторингу новим є компонент cadvisor. Він необхідний для опису стану docker – контейнерів. На *Рис. 11.* представлена схема взаємодії компонентів системи моніторингу між один одним та зовнішніми елементами, такими як сервер безперервної інтеграції та хмарний додаток. Архітектура програми вийшла досить економічною та компактною, незважаючи на величезну кількість компонентів. Однак кожен з них необхідний та виконує конкретне завдання. Оскільки програма містить персональні дані, то доступ обмежений авторизацією. Тип авторизації basic auth, тобто авторизація відбувається за логіном та паролем. Справедливо було і закрити доступ до моніторингу за допомогою авторизації, незважаючи на те, що система буде перебуває в приватній закритій мережі – авторизація за паролем не буде зайвим, а навіть необхідним заходом захисту комерційних та персональних даних. Ще один аспект, що стосується проектування архітектури – це виділення системи моніторингу окрему персональну машину. Це є важливим критерієм для цієї програми. Вся річ у тому, що додаток є медичним продуктом, тобто має вплив на процес лікування пацієнта. Тому за регламентом хостинг додатків такого виду здійснюється на спеціальних майданчиках, доступ яких обмежений.

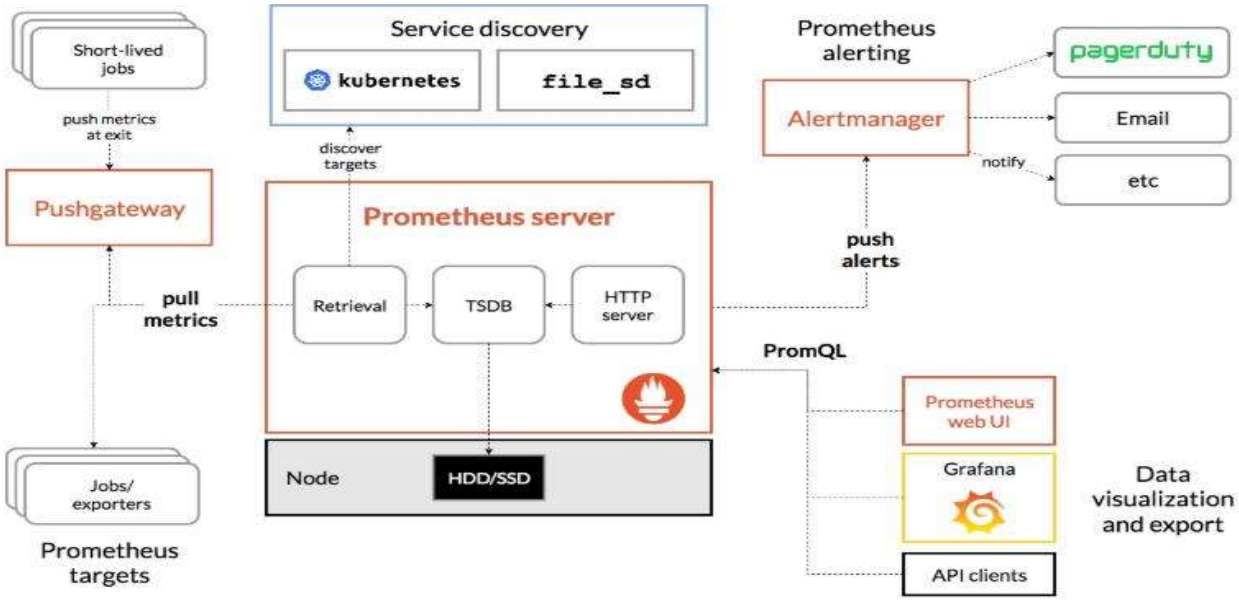

Рис. 11. Схема системи моніторингу та компонентів

Звідси випливає проблема моніторингу більшої частини програми - сервісів, що знаходяться за фаєрволом. Таку проблему дозволить вирішити комбінація методів доставки метрик у Prometheus – push та pull.

4.1. Конфігурація Prometheus

Скористаємося засобами Docker та запустимо, як і планували, згідно з архітектурою, Prometheus у контейнері. Для цього скористаємося базовим чином та зберемо власний контейнер на основі yml – файл конфігурації. Неповний вміст yml – файл представлений у додатку

Після цього ми можемо зібрати власний образ за допомогою команди:

docker run –t <image name> .

Потім запустити контейнер за допомогою docker-compose.yml.

Щоб перевірити результат, достатньо відкрити веб-інтерфейс Prometheus. Якщо в поспіху сервісів відображаються налагоджені в yml файлі послуги, то все зроблено коректно. Приклад проілюстрований на *Рис. 12.*

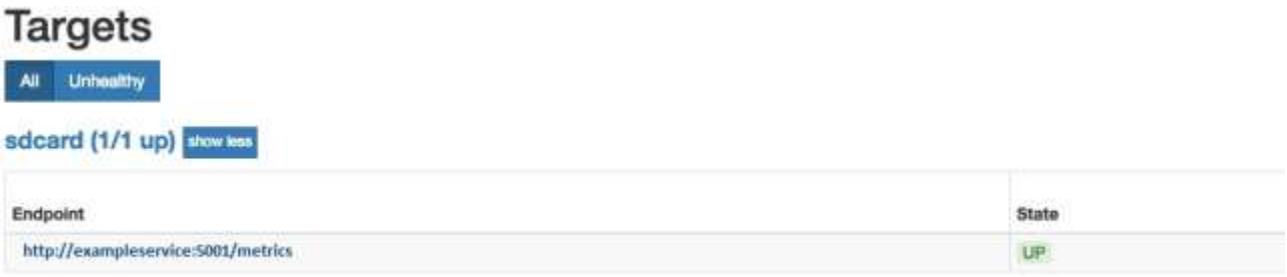

Рис. 12. веб-інтерфейс Prometheus

Інші компоненти додаються шляхом редагування файлу конфігурації. Їх конфігурація нічого очікувати відображено у цій роботі, у вигляді те, що містить багато персональної і комерційної інформації.

4.2. Налаштування Grafana

Знову скористаємося публічним чином для контейнера Docker і запустимо сервіс візуалізації метрик Grafana. Для цього до вже наявного конфігураційного файлу додамо ще одну секцію наступного змісту:

graphana: environment:

- GF_INSTALL_PLUGINS=grafana-piechart-panel image:

grafana/grafana ports:

- "3000:3000"

Вперше система попросить авторизуватися з дефолтним обліковим записом admin/admin. Після першого успішного входу слід змінити пароль та додати нових користувачів.

Все, що необхідно для налаштування Grafana – це лише задати джерело даних. Подальша робота полягає у вибірці значень, аналогічно SQL запитам, тільки використовуючи іншу мову - lucene. Це можна зробити через web-інтерфейс як адміністрування.

4.3. Додавання панелей візуалізації

Якщо оперувати термінами grafana, то візуалізації необхідно додати дашборди. Більше того, можна скористатися вже готовими рішеннями лише трохи під коригувавши їх під свої потреби. На панель візуалізації можна додати різні види

графічної інформації: діаграми, індикатори навантаження, текст, таблиці, аркуш метрик, аркуш попереджень. На *Рис. 13.* зображено меню налаштувань Grafana.

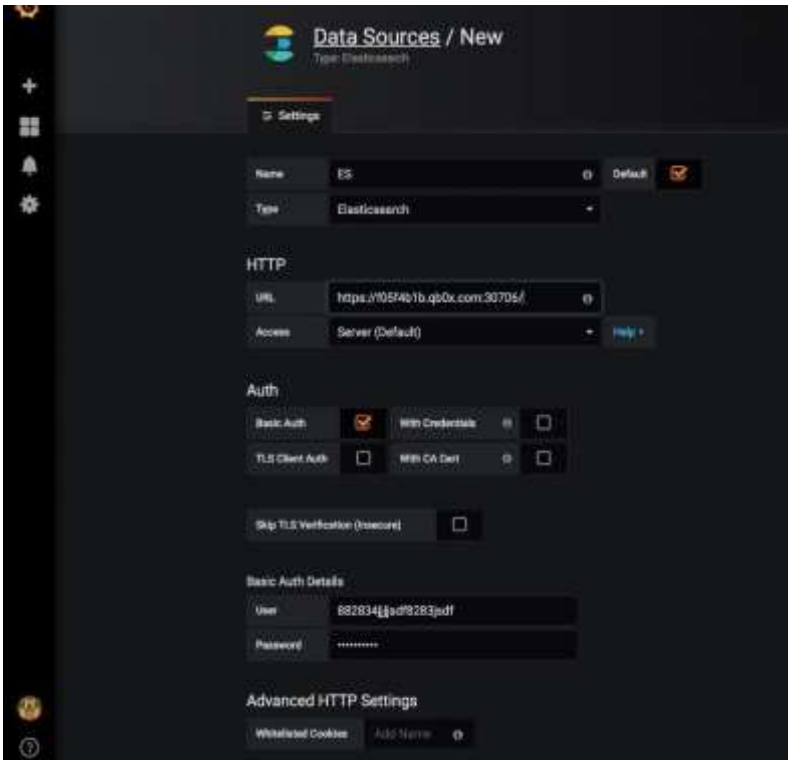

Рис. 13. Додавання джерела даних до grafana

Крім того, можна написати власний плагін. Підтримувані мови – javascript, причому навіть із підтримкою typescript, і html, css. Доступні типи візуалізації показано на *Рис.14*.

Рис. 14 Додавання нової панелі

Створимо простий графік, який відображатиме кількість авторизацій з використанням другого фактору.

Запити можна налагоджувати у вбудованому редакторі запитів. Результат такого налагодження показаний *Рис. 15*.

Рис. 15. Побудова метрики у Grafana

Скористайтеся готовим шаблоном для відображення апаратних ресурсів сервера з бібліотеки dashboard'ів із сайту Grafana Labs.

Результат представлений *Рис. 16*.

Рис. 16. Моніторинг апаратних ресурсів

Додамо деякі кількісні метрики, такі як кількість користувачів системи, кількість пацієнтів, організацій, середній час запитів для кожного сервісу та сервера, щоб оцінити процес додавання кількісних метрик. А також додамо критичні значення, такі як кількість довгих запитів, щоб знати про ймовірні затримки системи та загальну кількість досконалих запитів до системи. Результат представлений *Рис. 17.*

Рис. 17. кількісні метрики системи

РОЗДІЛ 5. РЕАЛІЗАЦІЯ ПРОГРАМНОЇ ЧАСТИНИ

Конфігурування Prometheus та підготовка мікро сервісів до моніторингу займе деякий час. Тому для інтеграції сервера безперервної інтеграції та тимчасового рішення для health – моніторингу необхідно реалізувати модуль експорту даних до Grafana. Аналіз варіантів реалізації Спочатку визначимо перелік завдань.

1. Необхідно, якимось чином, доставити дані до Grafana з

TeamCity та стан веб-сервісів, використовуючи http-запит service.baseAddress.domain/api/version.

2. Після цього дані, при необхідності, перетворюють в timeseries об'єкт з цифровим значенням.

3. Візуалізуйте дані за допомогою вбудованих компонентів візуалізації Grafana.

Проаналізуємо готові рішення для імпорту та експорту даних у Grafana. Плагін для TeamCity teamcity-graphite Даний плагін дозволяє відправляти дані у graphite у момент виконання збірок у TeamCity.

5.1. Встановлюємо та конфігуруємо graphite.

Для цього скористаємося docker-контейнером та розгорнемо його однією командою:

```
docker run -d\ --name
graphite\
--restart=always\
-p 80:80\
-p 2003-2004:2003-2004\
-p 2023-2024:2023-2024\
-p 8125:8125/udp\ -p
8126:8126\
graphiteapp/graphite-statsd
```

Встановлюємо teamcity-graphite плагін у TeamCity.

Результат *Рис. 18.*

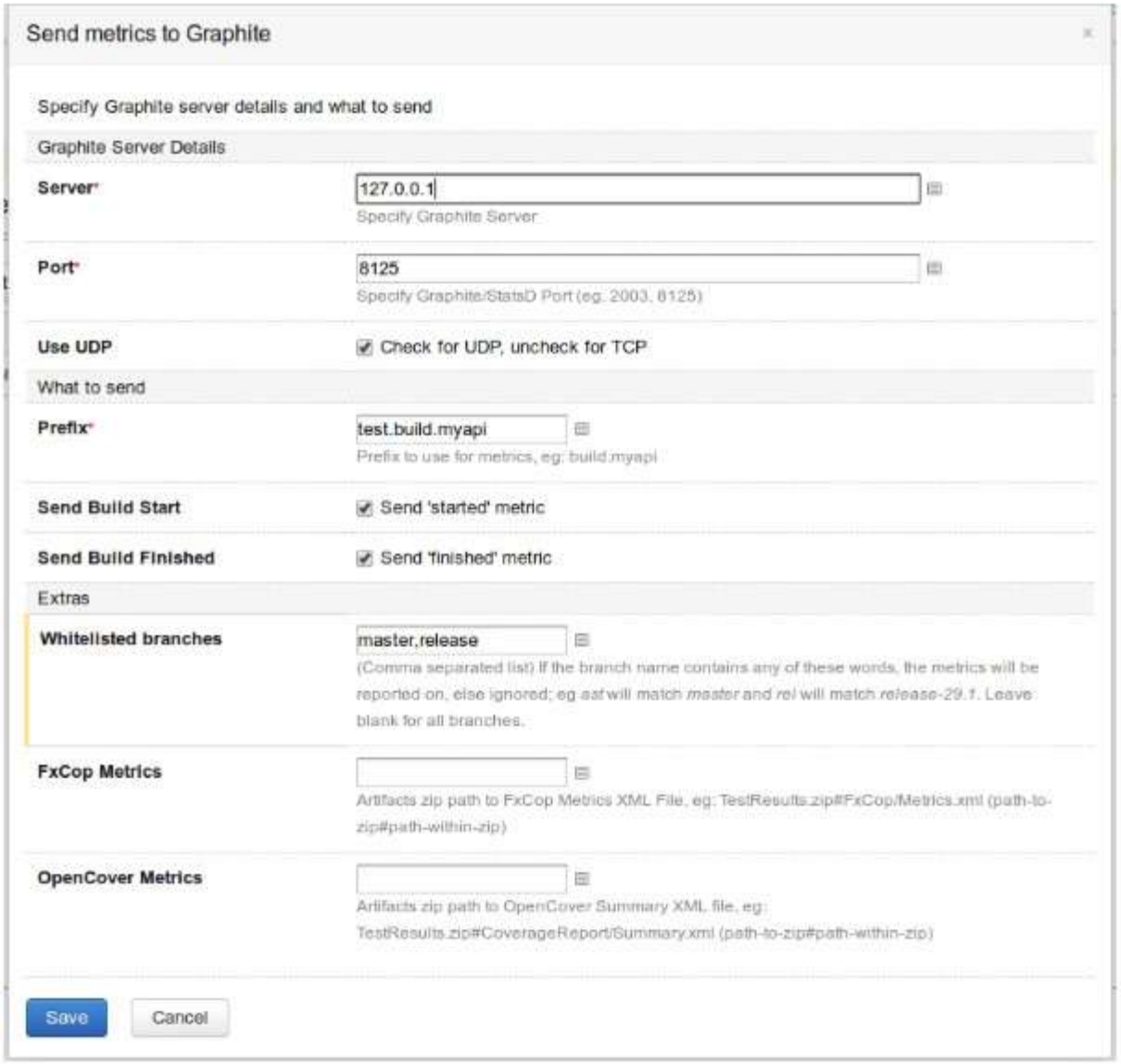

Рис. 18. Конфігурація плагіна у TeamCity

Налаштовуємо в Grafana джерело даних для сервера Graphite.

Результат представлений *Рис. 19.*

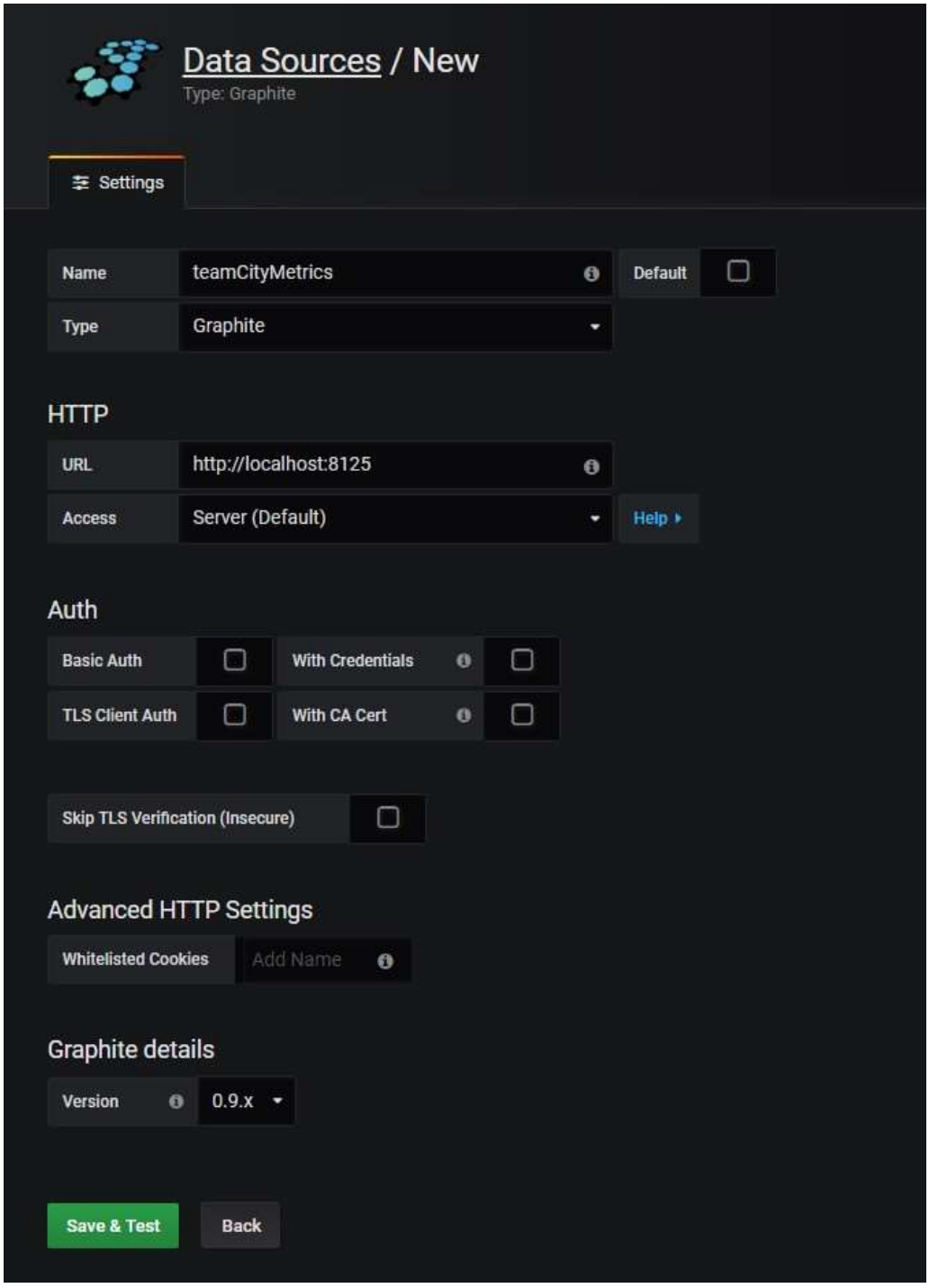

Рис. 19. Конфігурація джерела даних у Grafana

Візуалізуємо отримані метрики.

На жаль, це рішення не працює, з невідомої причини дані з Teamcity не надсилаються в сервер Graphite. Варіанти з доставкою UDP теж не увінчалися успіхом. Результат: рішення не працює.

Подивимося дані з іншого боку. Якщо не вдалося їх відправити, можливо, вийде їх запросити.

5.2. Використання плагіна teamcity-datasource для Grafana.

Опрацювання рішення:

- Встановлення плагіна у Grafana.
- Конфігурація плагіна, підключення teamcity-сервера.
- Візуалізація отриманих даних.

На *Рис. 20.* можна побачити, що успішно отримано список діючих збірок.

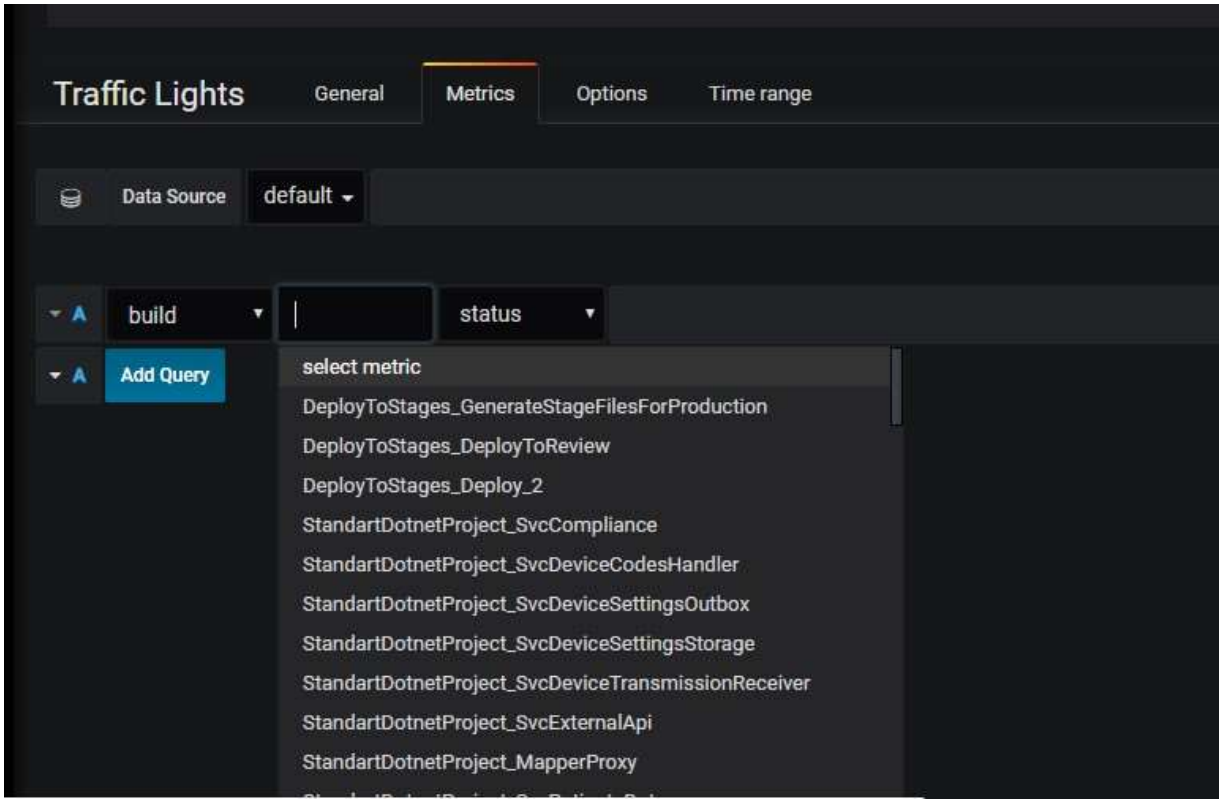

Рис.20. Демонстрація роботи плагіна

Результат: рішення працює, але є проблема. Цей плагін працює за API з використанням часу. Тобто якщо у нас у Grafana обраний період – останні 6 годин, а складання пройшло 2 дні тому, то ми не отримаємо жодних даних. При цьому відповідь з плагіна приходить рядком, що не дозволяє легко візуалізувати будь-що. Плагін не відповідає умовам системи.

5.3. Аналіз використаних рішень

Варіант з Graphite був хороший, тому що Graphite може використовувати кілька сторонніх джерел даних, зокрема один із продуктів, який так само використовується - AppMetrics. Для відображення внутрішнього стану сервісів.

Варіант із плагіном для Grafana сподобався більше, тому що були видні результати працездатності даного підходу, але відображення збірок за тимчасовим інтервалом абсолютно марно, тому що реліз у додатку відбувається раз на 2-3 місяці і отримати інформацію коректно для основної програми не вдасться.

5.4. Генерація нового рішення

Залишається єдине правильне рішення – написати свій плагін.

Тільки існує одне питання – для якої системи писати плагін, TeamCity з використанням Graphite, або для Grafana плагін — джерело даних?

Після аналізу та оцінки власних навичок було прийнято рішення писати плагін - джерело даних для Grafana, зважаючи на те, що:

1. Мова реалізації JavaScript, на відміну від Java. (Навичок та досвіду більше).

2. Можливо, повторне використання частини коду для написання плагіна – джерела даних для опитування веб-сервісів через Http.

3. Через те, що навичок у js більше, це істотно прискорить процес розробки.

4. Документація для Grafana чітко та структуровано описана.

5. Величезна кількість прикладів таких плагінів.

Після завантаження шаблону garafana-datasource-typescript був написаний плагін, що використовує TeamCity API.

Можливості плагіна:

- Запит списку всіх білдів, доступних на сервері авторизованого користувача.
- Відображення інформації щодо кожного білду, такої як: статус (числове значення від 0 до 100), строкове представлення статусу, номер білда, ім'я, ім'я проекту та дата виконання.

Результат побудови запиту – представлений *Рис. 21*.

Рис. 21. Побудова запиту

Статус є набором констант:

```
export const BuildStates: any = {
    SUCCESS: 100,
    PENDING_AND_SUCCESS: 65,
    PENDING: 50,
    PENDING_AND_FAILED: 35,
    QUEUED: 25,
    FAILED: 0
}
```

Опис роботи плагіну. Спочатку необхідно налаштувати джерело даних, передавши шлях до сервера та ім'я облікового запису користувача з паролем.

Вся комунікація між плагіном та кінцевим сервером відбувається через внутрішні функції Grafana, тому передача конфіденційних даних захищена.

Коли користувач налаштував джерело даних і намагається побудувати запит, завантажується список усіх доступних збірок. При виборі конкретного складання зі списку стають доступними метрики. Результат роботи плагіна це значення певної метрики для конкретних збірок.

Отримання метрик у числових константах дозволяє легко налаштувати панелі візуалізації в залежності від значення.

Частина коду представлена у додатку. Весь проект доступний у вільному доступі у github-репозиторії https://github.com/mav10/grafanateamcity-datasource.

Аналогічно було написано плагін для опитування сервісів за протоколом HTTP.

Опис роботи плагіна: користувач конфігурує джерело даних, додаючи базову адресу сервера, без додаткових шляхів та сабдоменів. Приклад представлений *Рис. 22.*

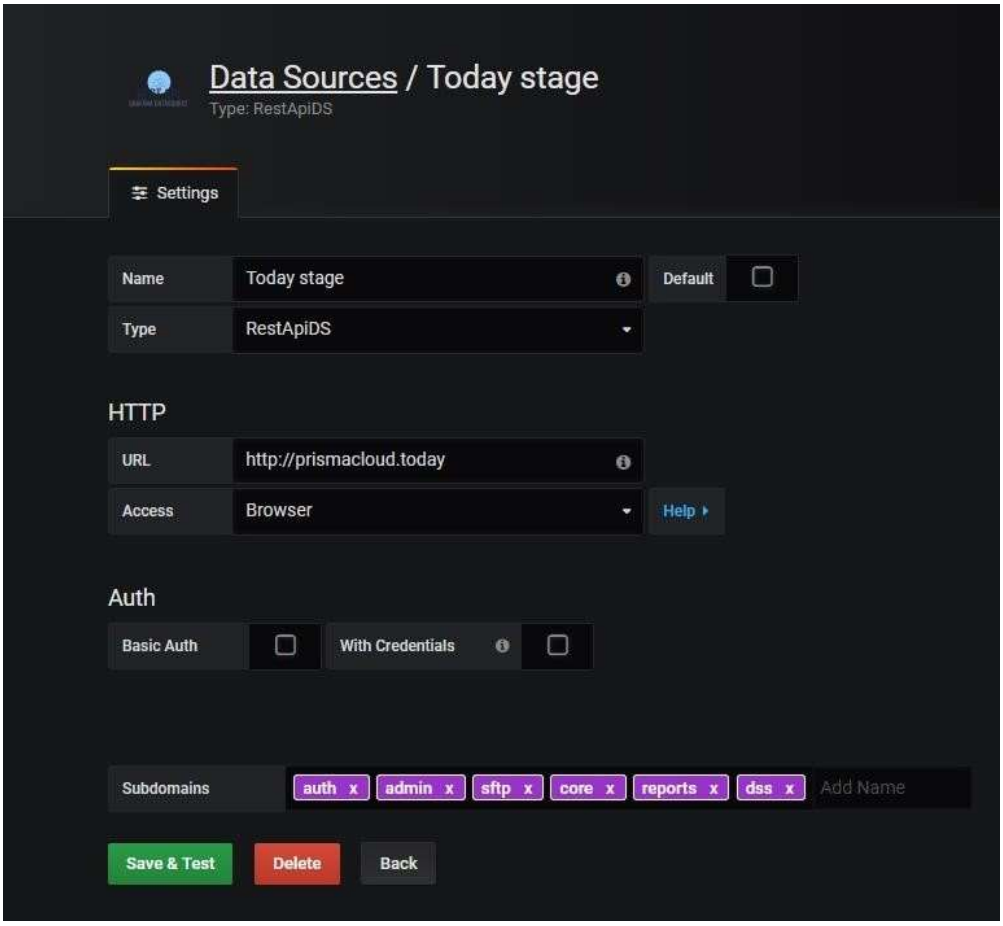

Рис. 22. Конфігурація джерела даних REST-API

Плагін був написаний таким чином, щоб не створювати багато джерел даних під кожен сервіс окремо. Для цього була реалізована підтримка сабдоменів та конфігурація шляхів.

Після конфігурації користувач вибирає необхідне джерело даних, як у прикладі на Рис. 10 – це Today Stage, потім сабдомен і при необхідності додає додатковий шлях, наприклад, /api/version.

Результат виконання плагіна - це числова відповідь, що відповідає HTTP-статус коду (200, 400, 404, 503 ...), а вже ці значення конвертуються в кінцевий результат.

Частина написаного коду доступна. Весь проект доступний у вільному доступі в github-репозиторії https://github.com/mav10/grafanarestapi-datasource.

5.5. Додавання панелей візуалізації

У відповідність до роботи плагінів була налаштована панель health мониторинга і утиліт. Як можна помітити на одному екрані, відразу доступна статистика з 3 різних серверів.

У Верхній частині дашборда відображені важливі послуги та результат останнього розгортання. Так, наприклад, на *Рис. 23.* для третього стейджу результат невдалий, чому відповідає напис та яскравий колір.

Рис. 23. Панель візуалізації з поганим результатом розгортання

У середній частині екрана відображено утиліти. На конкретному прикладі для кожної з реплік показані UI, інтеграційні тести, тести на безпеку.

Внизу розташовуються результати останнього складання кожного з мікро сервісів.

Як елемент візуалізації обраний графічний елемент світлофор, що відображає 3 різних стани:

1. зелений колір - складання завершено і пройшла успішна;
2. жовтий колір – зараз відбувається складання сервісу чи виконання операції;
3. червоний колір - складання завершилася з помилкою.

Кожен з таких елементів клікабельний і можна перейти в конкретний розділ сервера безперервної інтеграції.

ВИСНОВКИ

Результатом написання дипломної роботи є спроектована та розроблена система моніторингу для хмарної програми, що має мікро сервісну архітектуру, складні бізнес-процеси, медичну специфіку та зовнішню телеметрію.

Система наочно відображає актуальний стан програми у візуальній формі, за допомогою графіків, діаграм та різних графічних індикаторів. Процес візуалізації бізнес-показників, таких як: кількість зареєстрованих користувачів, організацій, полісомнографічних пристроїв, сила струму датчиків і т.д. складає основі лог – файлів окремих сервісів докладання. Розмір денного індексу інформації, що надходить, варіюється від гігабайта до десяти, що робить завдання візуалізації більш трудомісткою, але в той же час ще більш цікавою. Оскільки лог – файли є слабо структурованими даними, і в таких щоденних обсягах, рішення зводиться до обробки великих даних.

Інструменти для аналітики та візуалізації були підібрані відміно. За час тестування система жодного разу не відхилялася від фактичних значень. При високих навантаженнях не було ні єдиного натяку на зависання і повільну роботу.

Обравши хороший інструмент ми можемо просто та ефективно працювати з обладнанням.

Прості приклади показали нам наскільки потужним є новий API потокової передачі у Grafana, особливо в поєднанні з усіма різними візуалізаціями.

Це відкриває двері для різних протоколів зв’язку та апаратних платформ для потокової передачі даних через гнучку архітектуру Grafana.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Володимирський, Лебедєв: Телемедицина. Посібник. ГЕОТАРМедіа, 2018 р. ISBN: 978-5-9704-4195-4, 576 с.

2. Блажіс А.К., Дюк В.А. Телемедицина. - СПб: "СпецЛіт", 2000. - 154 с.

3. Гусіос Георгіос, Спінелліс Діомідіс. Ідеальна архітектура. Провідні спеціалісти про красу програмних архітектур. - Символ-Плюс,

2010 р. 580 с.

4. Michael J. Kavis. Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS). - Wiley; 1 edition (January 28, 2014), 224 pages.

5. Сем Ньюмен. Створення мікросервісів. – Бестселери O'Reilly (Пітер), 2016, 304 с.

6. Open Source Search & Analytics · Elasticsearch, [Електронний ресурс] - Kibana User Guide https://www.elastic.co/guide/en/kibana/6.2/index.html

(дата звернення: 12.03.2022)

7. Ігор Сухоруков, Матеріали конференції SECR-2014, “Збір та аналіз логів та метрик розподіленої програми за допомогою Elasticsearch,

Logstash, Kibana”

8. Betsy Beyer, Niall Richard Murphy, David K. Rensin, Kent Kawahara, Stephen Thorne. The Site Reliability Workbook: Практичні заходи до Implement SRE.

– O'Reilly Media, Inc., 2018 512 с.

9. Betsy Beyer, Chris Jones, Jennifer Petoff, Niall Richard Murphy. Site Reliability Engineering: How Google Runs Production Systems. – O'Reilly Media, Incorporated, 2016 524 с.

10. Gigi Sayfan. Mastering Kubernetes: Магістратура з художнього менеджменту, використовуючи владу Kubernetes. - Packt Publishing; 2nd Revised edition edition (April 27, 2018), 468 pages.

11. habrahabr.ru, [Електронний ресурс] - Блог компанії 2ГІС, стаття "Ще одна система логування, тепер на ElasticSearch, Logstash, Kibana та Prometheus", https://habr.com/company/2gis/blog/329128/ (дата звернення:

22.03.2022)

12. На відкритому source-Monitoring system with dimensional data model, flexible query language, efficient time series database and modern alerting approach,

[Електронний ресурс] – Documentation | Prometheus https://prometheus.io/docs/

(дата звернення 22.03.2022)

13. Grafana Labs, [Електронний ресурс] – docs, http://docs.grafana.org/

(дата звернення: 22.03.2022)

14. TeamCity Graphite Integration, [Електронний ресурс] - .Net logging https://code.mendhak.com/teamcity-graphite/ (дата звернення: 22.03.2022)

15. App Metrics [Електронний ресурс] – docs, https://www.appmetrics.io/ (дата звернення: 22.03.2022)

16. github.com, [Електронний ресурс] - найбільший веб-сервіс для хостингу IT-проектів та їхньої спільної розробки, TypeScript Template Data Source for Grafana, https://github.com/grafana/typescript-template-datasource (дата звернення : 22.03.2022)