ІНФОКОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ ТА ЕЛЕКТРОННА ІНЖЕНЕРІЯ
INFORMATION AND COMMUNICATION TECHNOLOGIES, ELECTRONIC ENGINEERING

№ 4 (2), 2024

МІГРАЦІЯ СЕРВІСІВ В КЛАСТЕРІ KUBERNETES НА ОСНОВІ ПРОГНОЗУВАННЯ НАВАНТАЖЕННЯ

Б. Федоришин [ORCID: 0009-0005-3779-0186], **О. Красько** [ORCID: 0009-0007-3607-4916]

Національний університет «Львівська політехніка», вул. С. Бандери, 12, 79013, Львів, Україна

Відповідальний за рукопис: Богдан Федоришин (e-mail: bohdan.fedoryshyn.mitpa.2022@lpnu.ua).

(Подано 1 Березня 2024)

У статті розглядається проблема масштабування мікросервісів в кластері Kubernetes, розглянуто існуючі підходи масштабування мікросервісної архітектури та запропоновано підхід до масштабування шляхом міграції частини компонентів. На відміну від найбільш поширених підходів горизонтального та вертикального масштабування, в яких необхідне виділення додаткових ресурсів для їх здійснення, суть запропонованого підходу полягає в міграції частини компонентів, які не є критично важливими для кінцевого користувача системи, на інший Kubernetes кластер. Запропонований підхід дозволяє звільнити ресурси на кластері в якому збільшується навантаження без необхідності залучення додаткових ресурсів. Це, в свою чергу, зменшує вартість обслуговування системи. Також було проведено порівняння реактивного та проактивного підходу до прийняття рішення системою про масштабування. Було обрано проактивний підхід до прийняття рішення, оскільки на відміну від реактивного підходу, де масштабування відбувається як реакція на збільшення навантаження в системі, в проактивному підході рішення про масштабування приймається на основі прогнозованих даних, ще до того, як реальне навантаження на систему почне зростати, що дозволить зберегти показник якості обслуговування QOS на хорошому рівні. Для вибору найоптимальнішого способу прогнозування навантаження було розглянуто існуючі моделі прогнозування та проведено практичне порівняння моделей прогнозування ARIMA, Prophet, LSTM. В результаті порівняння було обрано ARIMA, як модель для реалізації запропонованого підходу, було реалізовано запропонований підхід у вигляді docker контейнера з python застосунком всередині. Даний застосунок отримує дані про систему з Prometheus бази даних та здійснює прогнозування, після чого змінює конфігураційні файли з описом розгортання для ArgoCD та зберігає зміни в Git репозиторії. Після того як змінені конфігураційні файли потрапляють в Git репозиторій - ArgoCD отримує оновленну конфігурацію та проводить порівняння її з поточним розгортанням, якщо оновлена конфігурація відрізняється від існуючого розгортання- ArgoCD автоматично проводить інфраструктуру до заданого стану.

Ключові слова: *Kubernetes, прогнозування навантаження, масштабування, ArgoCD, ARIMA*

1. Вступ

В сучасному світі інформаційні системи постійно перебувають в динамічному навантаженні, це спровоковано різною активністю користувачів в залежності від робочих годин, періоду дня, дня тижня, святкових днів та інших чинників, так наприклад інформаційна система, яку використовує компанія для роботи буде знаходитися під більшим навантаженням в робочі години працівників в

будні дні, а інформаційна система інтернет-магазину скоріше навпаки буде зазнавати більшого навантаження в неробочі години, в вихідні та святкові дні. Це означає, що інформаційні системи постійно будуть потребувати різної кількості ресурсів для роботи. З одного боку для забезпечення стабільної роботи системи в неї повинно бути завжди вдосталь ресурсів, проте виділення надлишкових ресурсів для системи збільшує вартість її роботи - тому потрібно дотримуватися балансу таким чином, щоб система мала вдосталь ресурсів для того, щоб стабільно працювати та не мала надлишкових ресурсів, щоб зменшити вартість її роботи. Ця потреба в балансі створює необхідність у можливості масштабування системи.

Традиційно масштабування за типом поділяється на горизонтальне та вертикальне, а також за напрямом вверх та вниз. Горизонтальне масштабування - це зміна кількості копій компонентів системи, вертикальне масштабування - це збільшення, або ж зменшення ресурсів системи таких як CPU та RAM. Обидва типи можуть масштабуватися вверх, тобто збільшувати ресурси чи кількість копій, так і вниз, тобто їх зменшувати. Проте існують такі ситуації, коли на систему зростає навантаження, але вона має надлишкові ресурси в певному іншому місці, які не можуть бути застосовані для масштабування. Таке явище може спостерігатися, коли інформаційна система працює в декількох кластерах, проте навантаження зростає лише в одному з кластерів. Для такого випадку я пропоную новий підхід, який дозволить забезпечити працездатність інформаційної системи при зростанні навантаження та без необхідності в залученні додаткових ресурсів за рахунок міграції певних компонентів інформаційної системи з одного кластеру на інший.

Розглянемо інформаційну систему, яка працює одночасно на декількох кластерах Kubernetes незалежних один від одного, кожен з кластерів знаходиться в окремій географічній зоні. Це забезпечує швидкий доступ користувачів за рахунок використання кластеру, який географічно знаходиться в їх регіоні, та у випадку технічних неполадок в кластері, вони не впливають на роботу інформаційної системи в інших кластерах. Така інформаційна система складається з компонентів, які використовують користувачі - веб сервер, база даних та інші, та додаткових компонентів таких як система моніторингу. Швидкий доступ до клієнтських компонентів є важливим, оскільки це впливає на час відповіді сервера клієнту, тому вони повинні розташовуватися на кожному кластері, проте додаткові компоненти системи не використовуються користувачем та не впливають на час відповіді сервера, це дозволяє розмістити ці компоненти на одному кластерів для обслуговування всіх інших кластерів, що зменшує кількість ресурсів, які споживають додаткові компоненти.

Отже в даній інформаційній системі додаткові компоненти розміщені на одному з кластерів та використовують ресурси лише цього кластера. Проте виникають ситуації коли навантаження на користувацькі компоненти на цьому кластері зростає, а на інших кластерах ні - в таких ситуаціях доцільно буде використовувати запропонований мною підхід масштабування - масштабування вбік, або ж "міграція". Суть даного підходу полягає в міграції додаткових компонентів на найменш завантажений кластер. При цьому підході кількість ресурсів, які використовує система в загальному не змінюється, проте змінюється кількість доступних ресурсів на кожному з кластерів окремо. Приведемо порівняння використання підходу масштабування вверх, та міграції у випадку коли на кластер №2 зростає навантаження.

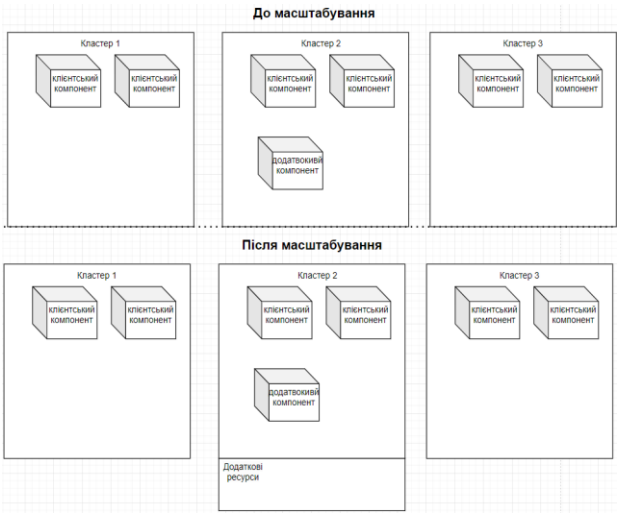

Рис. 1. Масштабування вверх

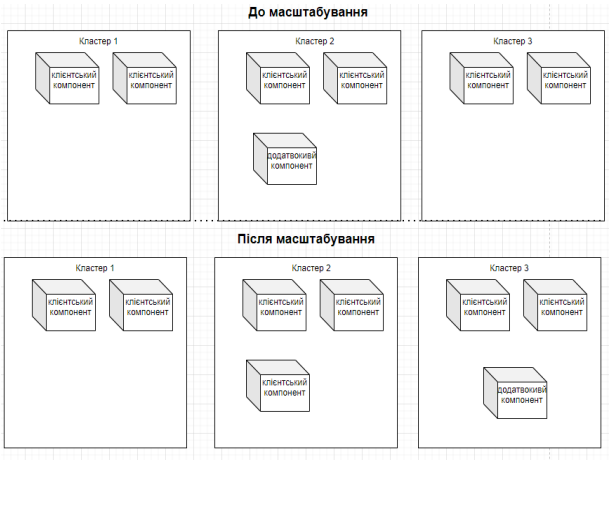

Рис. 2. Міграція

Як видно з першого підходу нам потрібно було виділити додаткові ресурси для кластера №2, а при застосуванні другого підходу додаткові компоненти з кластеру №2 були змігровані в кластер №3, що дозволило звільнити додаткові ресурси на кластері де навантаження зростає за рахунок використання іншого кластера з вільними ресурсами для додаткових компонентів.

Існує два підходи до масштабування системи - реактивний та проактивний, в першому варіанті система реагує на збільшення навантаження та починає масштабуватися, це дозволяє системі продовжити працювати, проте цей підхід має недолік у вигляді часу, який потрібне на масштабування, оскільки система починає масштабуватися вже тоді, коли не може справлятися з навантаженням - падає показник якості обслуговування, середній час відгуку (QOS). У випадку проактивного підходу - система прогнозує майбутнє зростання навантаження та завчасно починає масштабування ще до того, як це навантаження зросло до критичної точки, це дозволяє підтримувати показник QOS на хорошому рівні постійно. Зважаючи на економічні витрати та параметр QOS - ефективніше буде обрати проактивний метод масштабування системи.

2. Огляд математичних моделей для прогнозування часових рядів

Для прогнозування навантаження системи можна використовувати різні метрики системи, зазвичай використовують показники, які мають кореляційний зв’язок з навантаженням, такі як показники спожитих ресурсів - CPU, Memory, та показники кількості запитів до системи та час їх обробки. Ці показники збираються за допомогою системи моніторингу та представляють собою набір даних про значення показника в різні періоди часу, отже цей набір даних є часовим рядом.

Існують різні підходи до прогнозування часових рядів такі як:

- Прогноз експоненціально зваженим ковзаючим середнім — найпростіша модель прогнозування часового ряду. Застосовна в багатьох випадках. У тому числі, охоплює модель ціноутворення на основі випадкових блукань
- Моделі авторегресії і ковзного середнього (ARMA) — моделі орієнтовані на опис процесів, що виявляють однорідні коливання, порушувані випадковими впливами. Дають змогу передбачати майбутні значення ряду.
- Багатоканальні моделі авторегресії і ковзного середнього — моделі застосовуються в тих випадках, коли є кілька корельованих між собою часових рядів. У них є коливання, порушувані однією причиною. Дозволяють передбачати майбутні значення ряду.
- Сезонна модель Бокса-Дженкінса (ARIMA) — застосовується, коли часовий ряд містить явно виражений лінійний тренд і сезонні складові. Дає змогу передбачати майбутні значення ряду.

– Модель Довга короткочасна пам'ять (LSTM) - універсальна архітектура рекурентних нейронних мереж

Розглянемо детальніше кожний з способів в контексті застосування для нашої інформаційної системи.

Exponential Smoothing - далі (ES). Експоненційний метод згладжування - це метод, який використовує історичні середні значення змінної за певний період для того щоб зробити прогнозування щодо майбутньої поведінки змінної.

Визначається за формулою:

$$P_1 = P_0 + (D_0 - P_1) \tag{1}$$

$$S_t = \left\{ \frac{c_1}{S_t + a(C_t - S_{t-1})} \begin{matrix} t = 1 \\ t > 1 \end{matrix} \right\} \tag{2}$$

де: S_t - згладжений ряд; C_t - первинний ряд; a - коефіцієнт згладжування, який обирається апріорі з діапазону (0 < a < 1)

Double Exponential Smoothing, далі - (DES). На відміну від звичайного експоненційного згладжування (ES) враховує такий параметр як тренд.

$$L_t = \alpha \cdot (\frac{S_{tm}}{Y_t}) + (1-a) \cdot (L_{t-1} + T_{t-1}) \tag{3}$$

$$T_t = \beta \cdot (L_t - L_{t-1}) + (1-\beta) \cdot T_{t-1} \tag{4}$$

$$S_t = y \cdot (\frac{Y_t}{L_t} + (1-y) \cdot S_{tm}) F_{t+1} = + L_t + T_t \cdot S_{t-m+1} \tag{5}$$

де L_t - оцінка середнього значення часового ряду на момент часу t. Вона враховує поточні спостереження, історичні дані та тренди для надання згладженого значення, яке відображає основну тенденцію даних без сезонних коливань; α - коефіцієнт згладжування; S_{tm} - сезонний компонент на момент часу m; Y_t - спостережене значення на момент часу t; L_{t-1}: рівень середнього значення часового ряду на момент часу $t-1$ (враховує поточні спостереження, історичні дані та тренди для надання згладженого значення, яке відображає основну тенденцію даних без сезонних коливань); T_{t-1} - тренд на момент часу $t-1$; T_t -тренд на момент часу t; β - коефіцієнт згладжування тренду; S_t -сезонний компонент на момент часу t; y - коефіцієнт сезонності; F_{t+1} - прогноз значень ряду на момент часу $t+1$; S_{t-m+1} - сезонний компонент на момент часу $t-m+1$.

Дані два методи експоненціально зваженим ковзаючим середнім є доволі простими впроваджені, вони надають більшу вагу коефіцієнтам нещодавніх даних, це дозволє зменшити вплив на прогноз даних, які дальше в часі. Проте цей спосіб погано підходить для даних в яких є складні закономірності, такі як раптова зміна сезонності чи тренду, а отже використання даного підходу для прогнозування навантаження в інформаційній системі не буде доцільним.

Авторегресійне інтегроване ковзне середнє - ARIMA

Є узагальненням моделі авторегресійної ковзної середньої (ARIMA). Обидві ці моделі адаптуються до даних часових рядів або для кращого розуміння даних, або для прогнозування. Моделі ARIMA застосовуються в деяких випадках, коли дані демонструють докази нестаціонарності Коли сезонність відображається в часовому ряді, можна застосувати сезонну різницю щоб усунути сезонний компонент.

Ця модель базується на двох основних показниках - минулі значення та минулі помилки. В загальному ARIMA має три параметри - p, d, q.

- p: кількість затриманих спостережень у моделі; також відома як порядок затримки.
- d: кількість разів, які виходові спостереження диференціюються; також відома як ступінь диференціювання.
- q: розмір вікна ковзного середнього; також відомий як порядок ковзного середнього.

Прогнозування відбувається наступним чином, нехай y позначає d-те відмінне Y, що означає:

$$d = 0, y_t = Y_t \tag{6}$$

$$d = 1, y_t = Y_t - Y_{t-1} \tag{7}$$

$$d = 2, y_t = \left(Y_t - Y_{t-1}\right) - \left(Y_{-1} - Y_{t-2}\right) = Y_t - 2Y_{t-1} + Y_{t-2} \tag{8}$$

Для Y рівняння виглядає наступним чмином:

$$\hat{y}_t = \mu + \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} - \theta_1 e_{t-1} - \ldots - \theta_q e_{t-q} \tag{9}$$

де y_t - фактичне значення ряду на момент часу t; μ - константа; ϕ_1, ϕ_2, ⋯, ϕ_p - коефіцієнти авторегресії; y_{t-1}, y_{t-2}, ⋯, y_{t-p} - попередні значення часового ряду; θ_1, θ_2, ⋯, θ_p - коефіцієнти ковзного середнього; e_{t-1}, e_{t-2}, ⋯, e_{t-q} - попередні помилки прогнозування; e_t -поточна помилка прогнозування.

LSTM

Long short-term memory (LSTM). Модель, що працює з послідовністю даних, та на відміну від Prophet та ARIMA - може працювати не лише з часовими рядами - відноситься до рекурентних нейронних мереж.

Для прогнозування використовується наступне обчислення:

Основна ідея за клітинами LSTM полягає в тому, щоб вивчити важливі частини послідовності, побачені досі, і забути менш важливі. Це досягається за допомогою так званих воріт, тобто функцій, які мають різні цілі навчання, такі як:

- компактне представлення часового ряду;
- як поєднувати новий вхід з минулим представленням ряду;
- що забути про ряд;
- що вивести як прогноз на наступний крок часу.

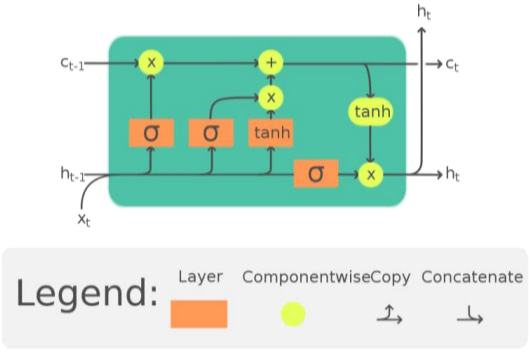

Рис. 3. Архітектура LSTM

PROPHET

Prophet - це бібліотека для прогнозування часових рядів на основі адитивної моделі, де нелінійні тренди адаптуються щорічними, тижневими та щоденними сезонностями, а також ефектами свят. Вона працює найкраще з часовими рядами, які мають виражені сезонні ефекти та

кілька сезонів історичних даних. Prophet стійка до відсутності даних та змін в тренді, і зазвичай добре впорядковується з викидами

Прогноз здійснюється за формулою:

$$y_t = g_t + s_t + h_t + e_t, \tag{10}$$

де g_t - лінійна або логістична крива часткового розрізання для моделювання не періодичних змін у часових рядів; s_t - сезонна компонента на момент часу *t*, яка відображає повторювані цикли або сезонні коливання в даних (тиждень/рік/сезон); h_t - вплив святкових днів (задає користувач) з ненормованим графіком; e_t - помилка прогнозування, враховує будь-які незвичайні зміни, які не враховані моделлю.

Тренд визначається за допомогою "моделі лінійної регресії з частковими відрізками". Математично тренд виглядає наступним чином:

$$y_t = g_t + e_t, \tag{11}$$

де y_t - значення часового ряду в момент часу t; g_t - тренд; e_t - похибка.

$$g_t = k_t \cdot t + m_t, \tag{12}$$

де k_t - це кут нахилу тенденції у часі t, а m_t - це перетин тенденції у часі t.

Модель сезонності як додаткового компонента - використовує підхід експоненціального згладжування в Holt-Winters прогнозуванні. Сезонність в Prophet рахується за формулою:

$$s_t = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi n t}{P} \right) + b_n \sin\left(\frac{2\pi n t}{P} \right) \right), \tag{13}$$

де s_t - сезонна компонента на момент часу t; a_n, b_n - коефіцієнти для сезонної компоненти; P - період сезонності (наприклад, 365.25 для річних даних).

Враховують певні свята та події, коли поведінка змінної нетипова. Prophet дозволяє задати минулі та майбутні події, дані під час яких будуть розглядатися як окремі події.

3. Вибір математичної моделі для прогнозування

Для порівняння було обрано ARIMA, LSTM, Prophet. Ці три підходи добре підходять для прогнозування навантаження на основі історичних даних у вигляді часових рядів для інформаційної системи, навантаження якої має певний тренд та сезонність. Для експерименту було взято часовий ряд для метрики яка відображає загальну кількість запитів в систему протягом 5 секунд за період 1 рік. Для прогнозування використовувався період перших 11 місяців, прогнозування здійснювалося на період тривалістю в 1 місяць, це дозволило визначити точність прогнозування шляхом порівняння спрогнозованих даних з реальними даними за 12-тий місяць. Результати прогнозування наведені на рисунках 4-6.

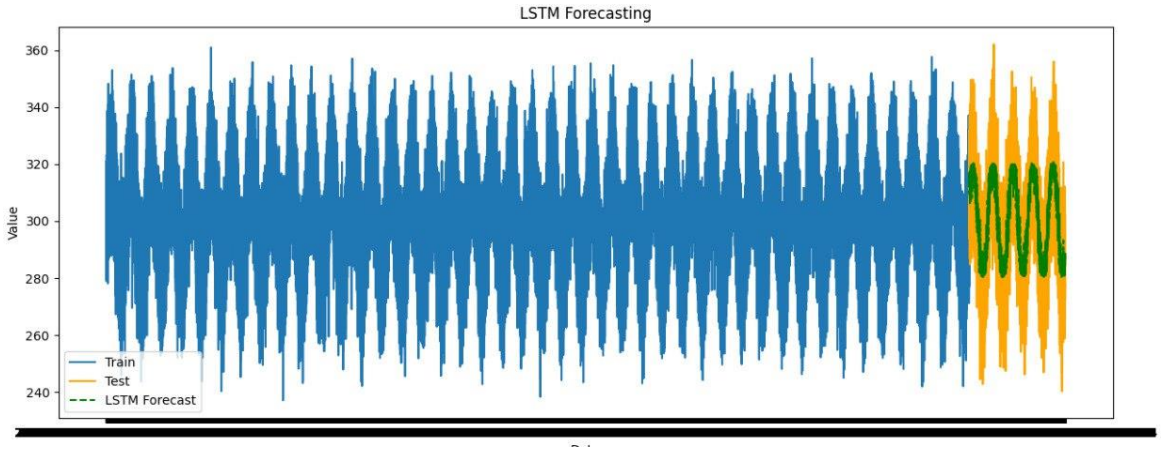

Рис. 4. Прогнозування LSTM

Середня абсолютна похибка: 19.373637533085441
Час прогнозування: 597.5793950557709

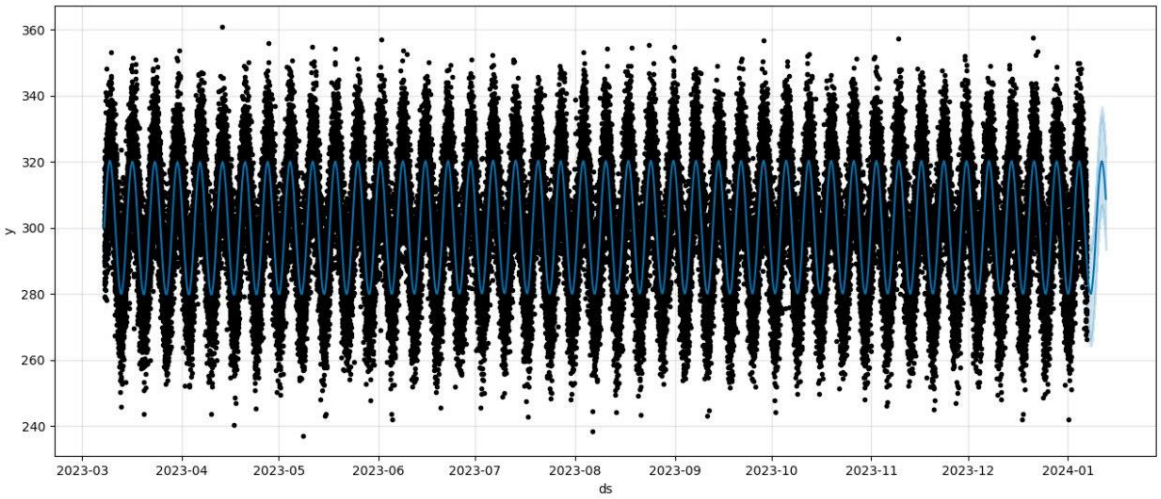

Рис. 5. Прогнозування Prophet

Середня абсолютна похибка: 19.224081440831526
Час прогнозування: 13.427870035171509 сек.

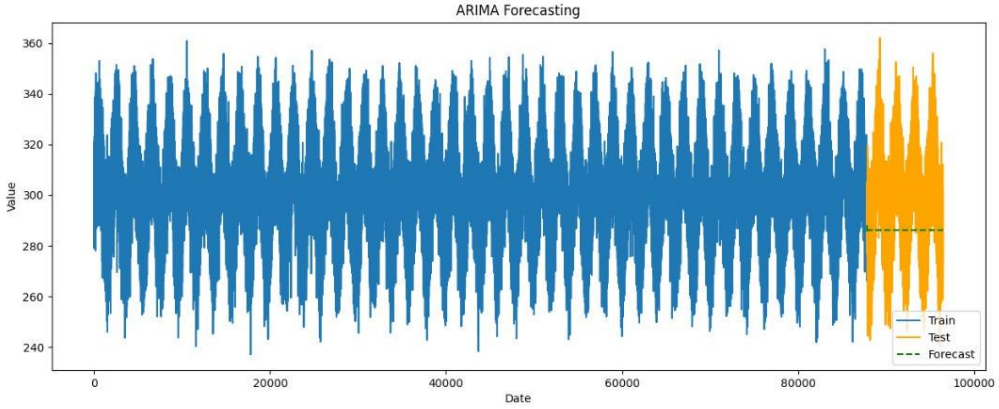

Рис. 6. Прогнозування ARIMA

Середня абсолютна похибка: 18.070020756855644
Час прогнозування: 2.265770673751831

Таблиця 1.

Результати прогнозування

Назва моделі	LSTM	Prophet	ARIMA
Середня абсолютна похибка	19.373	19.224	18.070
Час (сек)	597.579	13.42	2.265

Отже моделлю з найкращими результатами є ARIMA. Середня абсолютна похибка є приблизно однаковою у всіх трьох випадках, проте ARIMA потребує значно менше часу на проведення обчислення, що є важливим параметром у динамічному прогнозування навантаження на короткі періоди часу, оскільки це дозволить зменшити споживання ресурсів для проведення розрахунків. Також модель ARIMA є доволі простою в використанні на відміну від LSTM.

4. Практична реалізація запропонованого підходу

Отже, в даній роботі було розроблено програмне забезпечення для практичної реалізації запропонованого підходу міграції компонентів між кластерами.

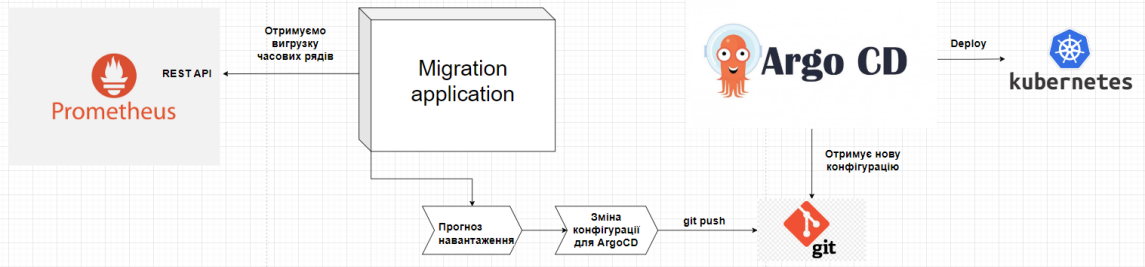

Рис. 7. Архітектура застосунку для міграції

Розгортання інформаційної системи в Kubernetes кластерах здійснюється з використанням gitOps підходу, який реалізує argoCD. argoCD - це continuous Delivery інструмент, який отримує конфігурацію з git репозиторію та приводить інфраструктуру в Kubernetes кластерах відповідно до заданої конфігурації. Prometheus використовується у якості бази даних для числових рядів. Опис розгортання інфраструктури являє собою Helm Chart, розглянемо детально файли values.yaml [додаток 1].

Даний файл відповідає за змінні чарту та містить змінну deploy_not_necessary_components яка вказує чи слід розгортати додаткові компоненти на кластері, чи лише основні.

Розглянемо файл Deployment.yaml [додаток 2] в даному файлі такі сервіси як alertmanager та prometheus будуть розгорнуті лише в тому випадку, якщо змінна deploy_not_necessary_components встановлена як True.

Розроблений застосунок на діаграмі підписаний "Migration application", написаний на Python з використанням моделі прогнозування ARIMA. Спершу застосунок отримує дані з Prometheus використовуючи rest API, для прийняття рішення використовується кількісний показник часу відповіді веб-сервера > 0.75 секунд. Слід задати наступні константи:

- percent_of_permissible_limit - допустимий відсоток виходу за пороговий ліміт, задається вручну, в моєму випадку це 20%;
- global_average_threshold - пороговий ліміт, задається вручну;
- periods - період в який слід робити перевірку;
- percent_of_limit_requests - відсоток для визначення квантелю, в даному випадку 99%.

Розглянемо як працює основний функціонал програми:

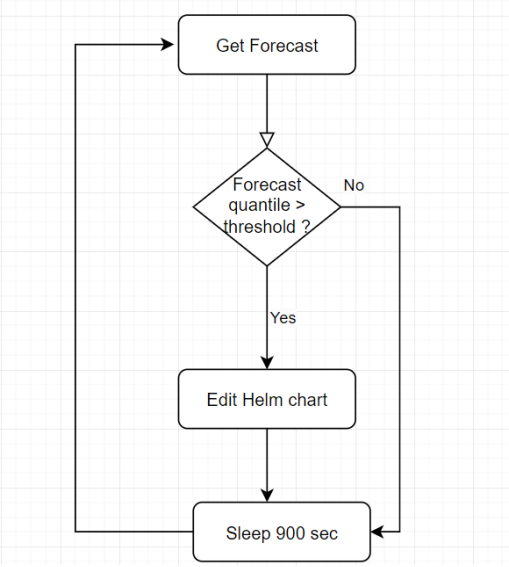

Рис. 8. Схема роботи функції перевірки кластера

Отже кожних 15 хв викликається функція, яка здійснює перевірку кластера. Спочатку отримується прогноз використовуючи модель ARIMA та зберігається в змінну forecast, після чого визначається квантиль з показником в 99% та проводиться порівняння чи не перевищує даний квантиль допустимий ліміт, якщо перевищує - викликається функція alarm, яка змінює helm параметр deploy_not_necessary_components, що заборонить розгортання додаткових компонентів на цьому кластері. Після чого зміни надсилаються в git репозиторій, звідки їх вже отримує argoCD та проводить порівняння заданої інфраструктури та поточної і у випадку змін - застосовує їх в Kubernetes кластерах. При наступних запусках функції, у випадку, якщо ж отриманий прогноз не перевищує заданий ліміт - параметр deploy_not_necessary_components знову зміниться та додаткові компоненти будуть розгорнуті на кластері, це дозволяє керувати розміщенням додаткових компонентів на кластерах, де є доступні ресурси для цього.

Для демонстрації роботи запропонованого підходу було змодельовано ситуацію при якій на кластер №2 починає зростати кількість запитів від користувачів в період з часового проміжку 0 до 80, після чого кількість запитів зменшується, на кластерах №1 та №3 розміщений клієнтська частина софту, на кластері №2 розміщено як клієнтський так і додатковий софт.

Рис. 9. Навантаження системи без використання запропонованого підходу

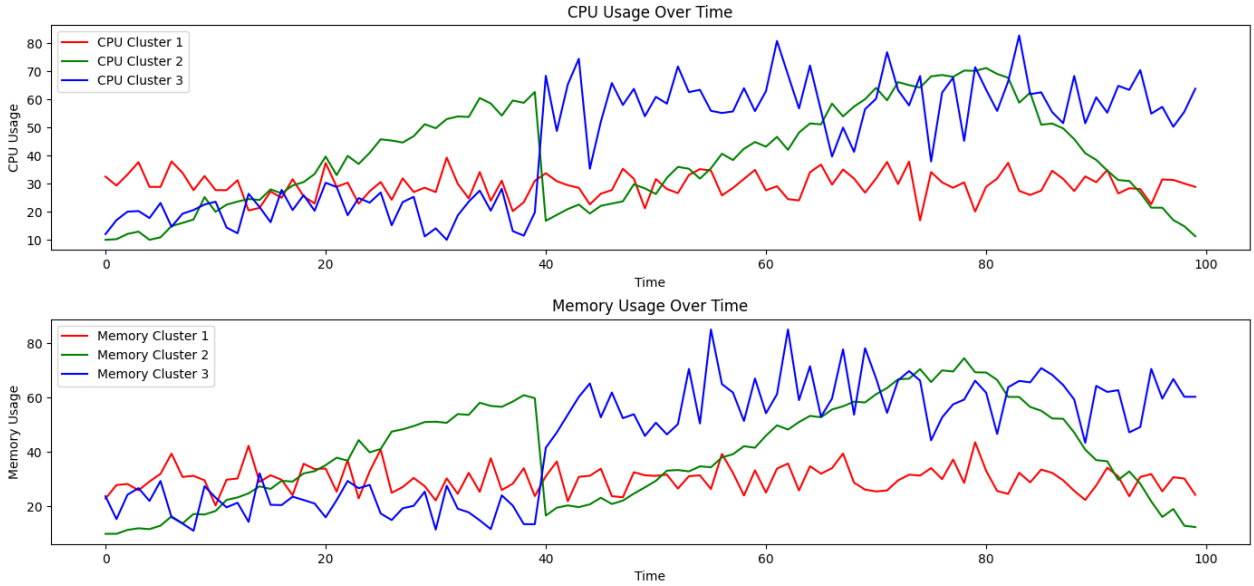

Рис. 10. Навантаження системи з використанням запропонованого підходу

Як ми бачимо у першому випадку наша система буде перенавантажена, оскільки показники CPU та Memory для кластеру №2 наблизяться до 100%

У другому випадку, з використанням запропонованого підходу міграції, в момент часу 40 - система спрогнозує майбутнє зростання навантаження та перенесе додатковий софт з кластеру №2 на кластер №3, що дозволить звільнити частину ресурсів на кластері №2 для коректної роботи при збільшені кількості клієнтських запитів.

Таблиця 2.

Середнє значення метрик

Назва кластера	Назва метрики	Без використання запропонованого підходу	З використанням запропонованого підходу
Cluster 1	CPU	29.48	29.48
	Memory	30.11	30.11
Cluster 2	CPU	69.34	39.7
	Memory	69.35	39.82
Cluster 3	CPU	19.74	44.1
	Memory	19.45	44.28

Таблиця 3.

Перевага використання запропонованого підходу

Метрика	Відсоток зменшення навантаження
Cluster 2 CPU	42.74%
CLuster 2 Memory	42.58%

Висновки

В даній роботі було проведено аналіз існуючих методів масштабування мікросервісів в кластерах Kubernetes, порівняння різних методів прогнозування навантаження, запропоновано новий підхід до масштабування сервісів у вигляді прогнозування навантаження на кластері та міграції частини компонентів на інший кластер. Було розглянуто існуючі підходи до прогнозування навантаження та проведено практичне порівняння таких моделей прогнозування як ARIMA, Prophet та LSTM. В результаті порівняння було обрано модель ARIMA та розроблено застосунок написаний

мовою програмування Python для реалізації запропонованого підходу міграції. В результаті практичного використання запропонованого метода міграції вдалося уникнути перенавантаження системи.

Список використаних джерел

[1] *S. J. Taylor and B. Letham, "Forecasting at scale" in The American Statistician, vol. 72, no. 1, pp. 37-45, 2018, doi: 10.1080/00031305.2017.1380080.*

[2] *J. Hamilton, "Time Series Analysis" Princeton University Press, 1994, ISBN: 9780691042893.*

[3] *B. Billah, M. King, R. Snyder, and A. Koehler, "Exponential smoothing model selection for forecasting" International Journal of Forecasting, vol. 22, pp. 239-247, 2006, doi: 10.1016/j.ijforecast.2006.03.005.*

[4] *B. Billah, M. King, R. Snyder, and A. Koehler, "Forecasting time series using a methodology based on autoregressive integrated moving average and genetic programming" 2006.*

[5] *A. Graves, "Long Short-Term Memory" Neural Networks, vol. 32, pp. 231-247, 2012, doi: 10.1016/j.neunet.2012.02.019.*

[6] *C. Beneditto, A. Satrio, W. Darmawan, B. Unrica, and N. Hanafiah, "Time series analysis and forecasting of coronavirus disease in Indonesia using ARIMA model and PROPHET" 2021.*

[7] *R. Shumway and D. Stoffer, "ARIMA Models" Springer, 2017, doi: 10.1007/978-3-319-52452-8.*

[8] *F. Beetz and S. Harrer, "GitOps: The Evolution of DevOps?" 2021.*

[9] *2B. Yuen, A. Matyushentsev, T. Ekenstam, and J. Suen, "GitOps and Kubernetes: Continuous Deployment with Argo CD, Jenkins X, and Flux" O'Reilly Media, 2021, ISBN: 978-1492094317.*

[10] *B. Yuen, A. Matyushentsev, T. Ekenstam, and J. Suen, "Forecasting Time Series Data with Facebook Prophet" Packt Publishing, 2021, ISBN: 978-1838982461.*

MIGRATION OF SERVICES IN A KUBERNETES CLUSTER BASED ON WORKLOAD FORECASTING

Bohdan Fedoryshyn, Olena Krasko

Lviv Polytechnic National University, S. Bandery Str., 12, 79013, Lviv, Ukraine

The article delves into the intricate challenge of scaling microservices within a Kubernetes cluster, thoroughly examining existing methodologies for scaling microservice architectures, and presenting a novel approach that involves migrating specific components. Unlike the conventional horizontal and vertical scaling strategies, which require additional resources, this proposed method focuses on migrating non-critical components to another Kubernetes cluster. This migration effectively frees up resources in the cluster experiencing increased load without necessitating extra resources, thus significantly reducing maintenance costs through lower server rental expenses. Furthermore, a detailed comparative analysis of reactive and proactive decision-making approaches for system scaling was conducted, with a preference shown for the proactive approach. Unlike the reactive method, where scaling is a response to an increase in load, the proactive approach relies on forecasted data to make scaling decisions before the actual load rises, thereby maintaining a high quality of service (QoS). To identify the most optimal load forecasting method, various models were reviewed and practically compared, including ARIMA, Prophet, and LSTM models. ARIMA was ultimately selected for implementing the proposed approach, realized as a Docker container with a Python application. This application retrieves system data from the Prometheus database and utilizes the ARIMA model for forecasting. Post-forecasting, it updates the deployment configuration files for ArgoCD and commits the changes to a Git repository. Once the updated configuration files are pushed to the Git repository, ArgoCD fetches the new configuration, compares it with the current deployment, and if there are discrepancies, automatically adjusts the infrastructure to the desired state. This approach not only optimizes resource usage within the Kubernetes cluster but also maintains high performance without incurring additional costs.

Keywords: *Kubernetes, workload forecasting, scalability, ArgoCD, ARIMA*