УДК 681.518, 004.896 DOI:10.31673/2412-4338.2021.035462

Кузьміч М. Ю., Гордієнко Т. Б.
Державний університет телекомунікацій, Київ

ІМПЛЕМЕНТАЦІЯ ТА АВТОМАТИЗАЦІЯ РОЗГОРТАННЯ WEBRTC ЗАСТОСУНКІВ В CLOUD NATIVE ОТОЧЕННІ

Анотація: *Карантинні обмеження та сучасні тенденції розвитку IT-інфраструктур призвели до значного інтересу до технологій передачі потокових даних. Застосування та розвиток систем на основі протоколу WebRTC для аудіо- та відео конференцій на сьогодні є актуальним і затребуваним. Протокол технології Web комунікації в реальному часі (Web real-time communications, WebRTC) дозволяє веб-браузерам взаємодіяти через прикладні програмні інтерфейси (Application Programming Interface, API), що в основному використовуються для аудіо- та відеоконференцій. Шлюз вибіркового направлення (SFU) та шлюз багатоточкового контролю (MCU) є технологіями, які доповнюють WebRTC. SFU та MCU сервери використовуються для покращення роботи при впливі неоднорідностей оточення, таких як збільшення кількості користувачів, специфічне мережеве обладнання. Тому, через специфіку їх роботи розгортання в Cloud Native оточенні є нетривіальною задачею.*

В статті представлено аналіз імплементації та автоматизації розгортання WebRTC SFU шлюзу у Cloud Native оточенні і його доцільність використання. Зроблено акцент на пошуку оптимального способу розгортання. Наведено приклад декларативного опису розгортання SFU шлюзу Jitsi у вигляді коду, що може бути перевикористаний для наступних експериментів. Автоматизація розгортання SFU серверів дозволяє покращити кількісну та якісну складові експериментів, зменшивши час, затрачений на його підготовку, та мінімізацію помилок людського фактору. Запропонований і описаний метод має великий потенціал для подальшого розвитку. Авторами також запропоновані зміни в HELM пакеті через сайт Github стосовно покращення роботи із залежностями, які були прийняті та імплементовані.

Ключові слова: *WebRTC, Cloud Native, автоматизація, Cluster and Cloud computing, Kubernetes, SFU, Open source, Helm, контейнеризація, мікросервіси.*

Kuzmich M. Yu., Gordiyenko T. B.
State University of Telecommunications, Kyiv

IMPLEMENTATION AND AUTOMATION OF WEBRTC APPLICATION DEPLOYMENT IN A CLOUD NATIVE ENVIRONMENT

Abstract: *Quarantine restrictions and current trends in the development of IT infrastructures have led to significant interest in streaming technologies. The application and development of WebRTC systems for audio and video conferencing is relevant and in demand today. Web real-time communications (WebRTC) allows Web browsers to interact through Application Programming Interfaces (APIs), which are used primarily for audio and video conferencing. Selective Direction Gateway (SFU) and Multipoint Control Gateway (MCU) are technologies that complement WebRTC. SFU and MCU servers are used to improve performance when exposed to environmental inhomogeneities, such as increasing the number of users, specific network equipment. Therefore, due to the specifics of their work, deployment in a Cloud Native environment is a non-trivial task.*

The article presents an analysis of the implementation and automation of WebRTC SFU gateway deployment in a Cloud Native environment and its feasibility. Emphasis is placed on finding the optimal deployment method. An example of a declarative description of the deployment of a Jitsi SFU gateway is given as code that can be reused for subsequent experiments. Automating the deployment of SFU servers has improved the quantitative and qualitative components of the experiments, reducing the time spent on its preparation and minimizing human error. Method of great potential for further development was proposed

and described. The authors also proposed changes to the HELM package through the Github site to improve the handling of dependencies that have been adopted and implemented.

Keywords: *WebRTC, Cloud Native, automation, Cluster and Cloud computing, Kubernetes, SFU, Open source, Helm, containerization, microservices.*

Кузьмич М. Ю., Гордиенко Т. Б.

Государственный университет телекоммуникаций, Киев

ИМПЛЕМЕНТАЦИЯ И АВТОМАТИЗАЦИЯ РАЗВЕРТЫВАНИЯ WEBRTC ПРИЛОЖЕНИЙ В CLOUD NATIVE ОКРУЖЕНИИ

Аннотация: *Карантинные ограничения и современные тенденции развития ИТ-инфраструктур привели к значительному интересу технологий передачи потоковых данных. Применение и развитие систем на основе протокола WebRTC для аудио- и видео конференций сегодня актуально и востребовано. Протокол технологии Web коммуникации в реальном времени (Web real-time communications, WebRTC) позволяет веб-браузерам взаимодействовать через прикладные программные интерфейсы (Application Programming Interface, API), используемые в основном для аудио- и видеоконференций. Шлюз выборочного направления (SFU) и шлюз многоточечного контроля (MCU) являются технологиями, которые дополняют WebRTC. SFU и MCU серверы используются для улучшения работы при воздействии неоднородностей окружения, таких как увеличение количества пользователей, специфическое сетевое оборудование. Поэтому, в силу специфики их работы развертывание в Cloud Native окружении является нетривиальной задачей.*

В статье представлен анализ имплементации и автоматизации развертывания WebRTC SFU шлюза в Cloud Native окружении и его целесообразности использования. Сделан акцент на поиске оптимального способа развертывания. Приведен пример декларативного описания развертывания шлюза SFU Jitsi в виде кода, который может быть переиспользован для последующих экспериментов. Автоматизация развертывания серверов SFU позволила улучшить количественную и качественную составляющие экспериментов, уменьшив время, затраченное на его подготовку, и минимизацию ошибок человеческого фактора. Предлагаемый и описанный метод имеет большой потенциал для дальнейшего развития. Авторами также предложены изменения в HELM пакете через сайт Github по улучшению работы с зависимостями, которые были приняты и имплементированы.

Ключевые слова: *WebRTC, Cloud Native, автоматизация, Cluster and Cloud computing, Kubernetes, SFU, Open source, Helm, контейнеризация, микросервисы.*

1. Вступ

Останніми роками прослідковується значний інтерес до технологій передачі потокових даних, викликаний вимогами часу та карантинними обмеженнями. Потреба в ефективному, надійному та безпечному каналі спілкування існує у різних сферах життєдіяльності, зокрема у державних структурах, у військових та приватних компаній. Віддалена робота також вносить корективи в різні сфери діяльності. Наприклад, компанія Twitter дозволила віддалено працювати своїм працівникам незалежно від карантинних обмежень, як і багато інших роботодавців, оскільки вбачають в цьому позитивні моменти [1]. Про зростання інтересу до даних технологій свідчить зростання цін акцій компаній на фондовому ринку, що спеціалізується на відеоконференціях. Тому актуальність застосування технологій передачі потокових даних наразі доволі висока, а необхідність у швидкому та ефективному розгортанні є на часі, що може знайти своє відображення у використанні Cloud Native екосистеми (хмарних середовищ).

Застосування та розвиток систем на основі протоколу WebRTC для аудіо- та відео конференцій на сьогодні є актуальним і затребуваним. Протокол технології Web комунікації в реальному часі (Web real-time communications, WebRTC) дозволяє веб-браузерам взаємодіяти через прикладні програмні інтерфейси (Application Programming Interface, API), що в основному використовуються для аудіо- та відеоконференцій. Шлюз вибіркового

направлення (SFU) та шлюз багатоточкового контролю (MCU) є технологіями, які доповнюють WebRTC. З рис. 1 а) видно, що при зростанні кількості користувачів в рамках однієї сесії, кількість одночасних з'єднань зростає в геометричній прогресії від кількості клієнтів, збільшуючи тим самим навантаження на мережу та систему в цілому. Як показано рис. 1 б) використання SFU, дозволяє зменшити кількість одночасних сесій. Отже, SFU та MCU сервери використовуються для покращення роботи при впливі неоднорідностей оточення, таких як збільшення кількості користувачів, специфічне мережеве обладнання, але, через складність їх роботи, розгортання в Cloud Native оточенні залишається нетривіальною задачею.

2. Аналіз літературних даних і постановка проблеми

Аналіз застосування та розвитку систем на основі протоколу WebRTC для аудіо- та відеоконференцій здійснювався багатьма авторами.

Протокол WebRTC дозволяє веб-браузерам взаємодіяти в реальному часі через прикладні програмні інтерфейси API – інтерфейси JavaScript. Вони є однією із ключових в області аудіо- та відеоконференцій, але мають певні недоліки. Зокрема, при зростанні кількості користувачів в рамках однієї сесії, кількість одночасних з'єднань зростає в геометричній прогресії від кількості клієнтів і тим самим значно збільшуючи навантаження на мережу та систему в цілому. Цю проблему було вирішено у роботі [2] з використанням шлюзу вибіркового направлення (SFU) та шлюзу багатоточкового контролю (MCU) серверів, які в свою чергу є складними інфраструктурними компонентами.

У роботах [3, 4] досліджено переваги та специфіка протоколу WebRTC, його можливості для бізнесу та медицини з використанням WebRTC-сумісного веб-браузера Google Chrome або Firefox. У [5] розглянуті питання вирішення проблем багатоточковості аудіо- та відеоконференцій з використанням технології SFU та MCU. Запропоновано підхід при якому модуль MCU інтегрований у браузер для змішування та транскодування відео та аудіо потоків у режимі реального часу.

Сам процес розгортання SFU та MCU застосунків зазвичай залишається поза межами дослідження, при тому, що складність та експлуатаційні навантаження є високими. У роботі [6] представлена порівняльна характеристика роботи WebRTC застосунків в контейнеризованому оточенні Docker та віртуальній машині, що показало кращу ефективність використання ресурсів при використанні контейнеризації, аніж віртуалізації.

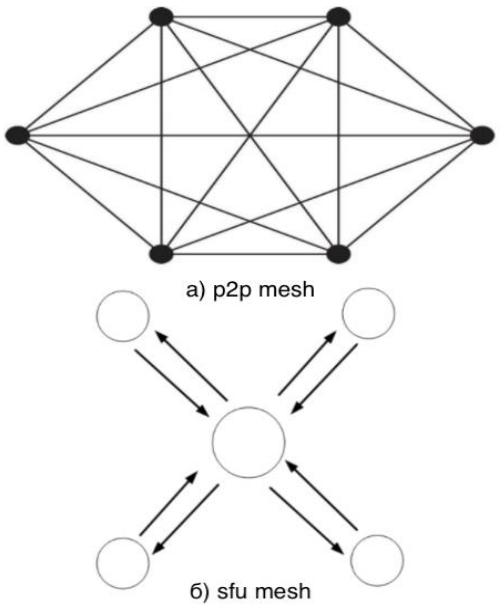

Рис. 1. Схематичне відображення з'єднань p2p mesh (один з одним) (а)
та із використанням технології SFU (б)

У роботі [7] здійснено огляд основних відкритих джерел SFU серверів (Jitsi, Janus, Mediasoup, Medooze) та використання інструментів автоматичного тестування, що є корисним для проведення майбутніх експериментів. На основі результатів цієї роботи було вибрано SFU сервер Jitsi для огляду розгортання.

Питання концепції Cloud Native розглянуто у роботі [8], а у [9] – доцільність міграції застосунків відповідно цьому патерну, оскільки в деяких випадках це економічно невигідно.

Однак у проаналізованих роботах не досліджено питання спрощення експлуатації та розгортання SFU серверів в Cloud Native екосистемі. Тому такі дослідження є актуальними і доцільними з точки зору комерційного застосування та подальших експериментів. За рахунок автоматизації очікується зменшення часу на підняття інфраструктурних компонентів та зниження впливу людського фактору, що покращить якісну та кількісну складову потенційних експериментів у цій сфері.

3. Мета і задачі дослідження

SFU сервери є одними із ключових елементів інфраструктури для WebRTC аудіо- та відеоконференцій. У зв'язку цим, існує потреба в його простому та швидкому способі розгортання. *Метою дослідження є* огляд, підбір та обґрунтування оптимального способу імплементації SFU сервера із використанням сучасних інструментів та стандартів індустрії.

Для досягнення мети розв'язуються такі наукові задачі:

– дослідження можливостей спрощення експлуатації та розгортання SFU серверів в Cloud Native екосистемі;

– розробка рекомендацій щодо автоматизації для зменшення часу на підняття інфраструктурних компонентів та зниження впливу людського фактору, з можливим покращенням якісної та кількісної складової потенційних експериментів у цій сфері.

4.1 Особливості Cloud Native концепції

Cloud Native – це термін, який описує патерни в організаціях, архітектури програмного забезпечення та застосунків, технології, які використовують переваги хмарних технологій [5]. Cloud Native технології використовуються для побудови застосунків, що складаються із сервісів ізольованих в контейнерах, доставлених у вигляді мікросервісів, які управляються з використанням гнучкої методології автоматизації технологічних процесів складання, налаштування та розгортання програмного забезпечення DevOps.

Ключовою ознакою Cloud Native застосунків є їх контейнеризація [7]. Не прикріпленість до якоїсь конкретної платформи дозволяє бути запущеними в різних умовах, таких як приватні центри даних, домашні комп'ютери, військові літаки, сервіси публічних хмарних провайдерів (AWS, Azure, Google Cloud). Невід'ємною складовою портативності є сама платформа Kubernetes, яка розробляється під егідою Cloud Native Computing Foundation (CNCF). CNCF – це проект Linux Foundation, який був заснований для допомоги просуванню технології контейнеризації, налаштування технологічної галузі щодо її розвитку, та інші інструменти, які розширюють функціонал платформи.

Іншою важливою складовою, є доступність цих технологій. Cloud Native застосунки використовують open source (відкриті джерела) для розробки, тобто програмне забезпечення з відкритим кодом. Це робить інновації доступними для компаній та навчальних закладів, які б в іншому випадку не мали б до них доступу, що стимулює та прискорює розвиток індустрії. Linux – один із найвідоміших open source продуктів.

Cloud Native концепція передбачає розподіленість застосунків – перехід від монолітної архітектури до мікросервісів. Це дозволило горизонтально масштабувати та збільшити відмовостійкість сервісів, продублювавши найбільш важливі та розташувавши їх на різних машинах.

4.2 Застосування Cloud Native та WebRTC

На сьогоднішній день WebRTC застосунки, такі як Jitsi [9], Janus WebRTC Server рекомендують інсталювати на віртуальних серверах із використанням контейнерів, але без системи їх оркестрації та без ізоляції на рівні мережі (активований мережевий режим "host networking"), що впливає на можливості автоматизації, стійкості та тісної інтеграції з іншими

частинами Cloud Native екосистеми.

Це викликано наступними причинами:

- неефективність систем контейнеризації (як от середовище для управління ізольованими Linux-контейнерами Docker) при необхідності використання великої кількості портів;

- необхідність збереження стану (statefull) програмних застосунків, що працюють із потоковими даними (WebRTC, VoIP), оскільки платформа Kubernetes на початку була створена для роботи із процесами без збереження стану (stateless).

В Kubernetes наявний компонент із назвою “stateful controller”, який створений для ефективного управління відповідними застосунками. Також, на момент написання статті такі програмні застосунки як Jitsi можуть передавати потоки даних, використовуючи один UDP(Протокол датаграм користувача) порт. Архітектурна схема компонентів SFU Jitsi зображена на рис. 2., що показує мережеві зв’язки між ними.

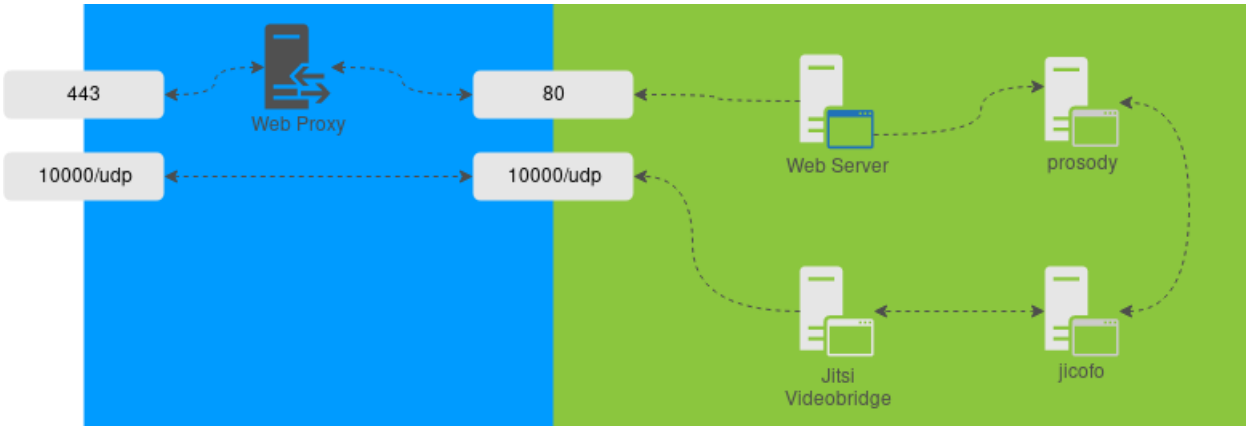

Рис. 2. Архітектурна схема компонентів SFU Jitsi

Як видно з рис. 2, для роботи клієнтської частини немає необхідності у виділенні великої кількості мережевих портів, а тільки один протокол керування передаванням (UDP) port(10000) для передачі потокових даних та два стандартних протоколи керування передаванням (TCP) port(80,443) для взаємодії із Web додатком.

Тому, беручи до уваги можливість передавання потокових даних через один UDP port та наявність “stateful controller” в Kubernetes, теоретично немає значних технічних проблем, через які розгортання та експлуатація SFU серверів в Cloud Native оточенні було б неможливим.

4.3 Роль пакетного менеджера HELM при розгортанні застосунків в Kubernetes

Cloud Native підхід являє собою кращі практики в індустрії експлуатації та розробки програмних продуктів. Однак, він збільшує складність систем, бо збільшується кількість об’єктів та абстракцій, якими потрібно керувати та розуміти. Як доказ, у табл. 1 наведено порівняння основних абстракцій при розгортанні на Linux сервері та Kubernetes. Ця проблема вирішується системами управління пакунками, що є набором системного програмного забезпечення, яке дозволяє управляти процесом установки, видалення, налаштування та оновлення різних компонентів програмного забезпечення. Системи управління пакетами активно використовуються в різних дистрибутивах операційної системи Linux та інших UNIX-подібних операційних системах. Програмне забезпечення представляється у вигляді пакетів, що містять, крім дистрибутива програмного забезпечення, набір певних метаданих, які можуть включати в себе повне ім’я пакета, номер версії, опис пакету, ім’я розробника, контрольну суму, відносини з іншими пакетами. Метадані зберігаються в системній базі даних пакетів.

Аналогічну концепцію використовує Helm – менеджер з управління пакетами для Kubernetes. Helm допомагає гнучко керувати програмними застосунками за допомогою

концепції чартів – пакетів із преконфігурованими шаблонами об'єктів. Така гнучкість була досягнута використанням потужних шаблонів [11] мови програмування Golang.

Helm володіє широкою сферою застосування, а саме:

- пошук та встановлення популярних програмних застосунків в Kubernetes;
- обмін своїми застосунками у вигляді Helm чартів;
- створення відтворювальних образів для застосунків;
- ефективне управління Kubernetes маніфестами;
- управління життєвим циклом Helm пакетів.

Таблиця 1.

Порівняння основних абстракцій при розгортанні на Linux сервері та Kubernetes кластері

Основні абстракції при розгортанні	
на Linux сервері	**в Kubernetes кластері**
Процес Systemd (підсистема ініціалізації і управління службами)	Розгортання (Deployment)
Мережевий порт	Розгортання із збереженням стану(Statesfullset)
Вебсервер (Web Server)	Точка входу(Ingress)
Файли конфігурацій	Сервіс
–	Горизонтальне розширення (Horisontal pod autoscaling)
–	Список конфігурацій(Configmap)
–	Секрет(Secret)
–	Капсула(Pod)

Таким чином Helm дозволяє скоротити час встановлення та налаштування застосунків в Kubernetes в рази за допомогою гнучких шаблонів, які підлаштовуються під необхідний сценарій.

5. Обговорення результатів дослідження декларативного опису розгортання WebRTC застосунку

Здійснено огляд архітектури системи для відеоконференцій на основі Jitsi. Кінцева система, на основі якої можна проводити порівняльний аналіз навантажувального тестування, включає додаткові пакети: cert-manager та ingress-nginx. Ingress-nginx виконує функцію точки входу і перенаправляє трафік на потрібний сервіс. Cert-manager забезпечує валідність протоколу захисту транспортного рівня (transport layer security, TLS) сертифікатів, оскільки для WebRTC невалідність сертифікатів призводить до блокування трафіку на рівні користувача, що відповідає політиці безпеки сучасних браузерів.

Для автоматизації розгортання декількох Helm пакетів було використано інструмент Helmfile [12], що дозволяє: здійснювати декларативний опис; працювати на рівні клієнта; та не встановлювати в кластер додаткові системи розгортання, такі як ArgoCD або Flux.

Простота інструменту Helmfile, в порівнянні із іншими аналогами, є вагомим чинником, оскільки це дозволяє збільшити охоплення та кількість якісних досліджень в цій галузі. Також декларативний опис необхідний для досягнення мінімізації похибки відтворюваності експерименту, оскільки опис кінцевої системи представлений у вигляді коду (IaaC – інфраструктура як код). Вихідний код розгортання розміщено у відкритому доступі Github за відповідним посиланням [13], а знімок його екрану наведено на рис. 3.

Відповідні рекомендації щодо покращення Helm пакета, що стосується управління залежностями, було доведено до розробника через платформу github [14].

На рис. 4 наведено схему взаємодії відповідних мікросервісів та їх ієрархічне положення в обробці трафіку (початкова точка входу – Ingress Nginx, кінцева – мікросервіс JVB), де назви на рисунку відповідають назвам у висхідному коді.

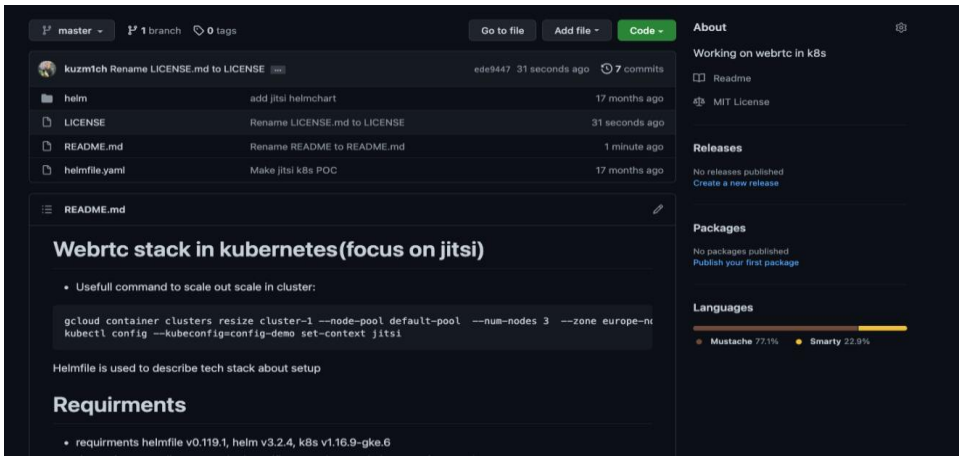

Рис. 3. Кодова база, що перебуває у вільному доступі на платформі github

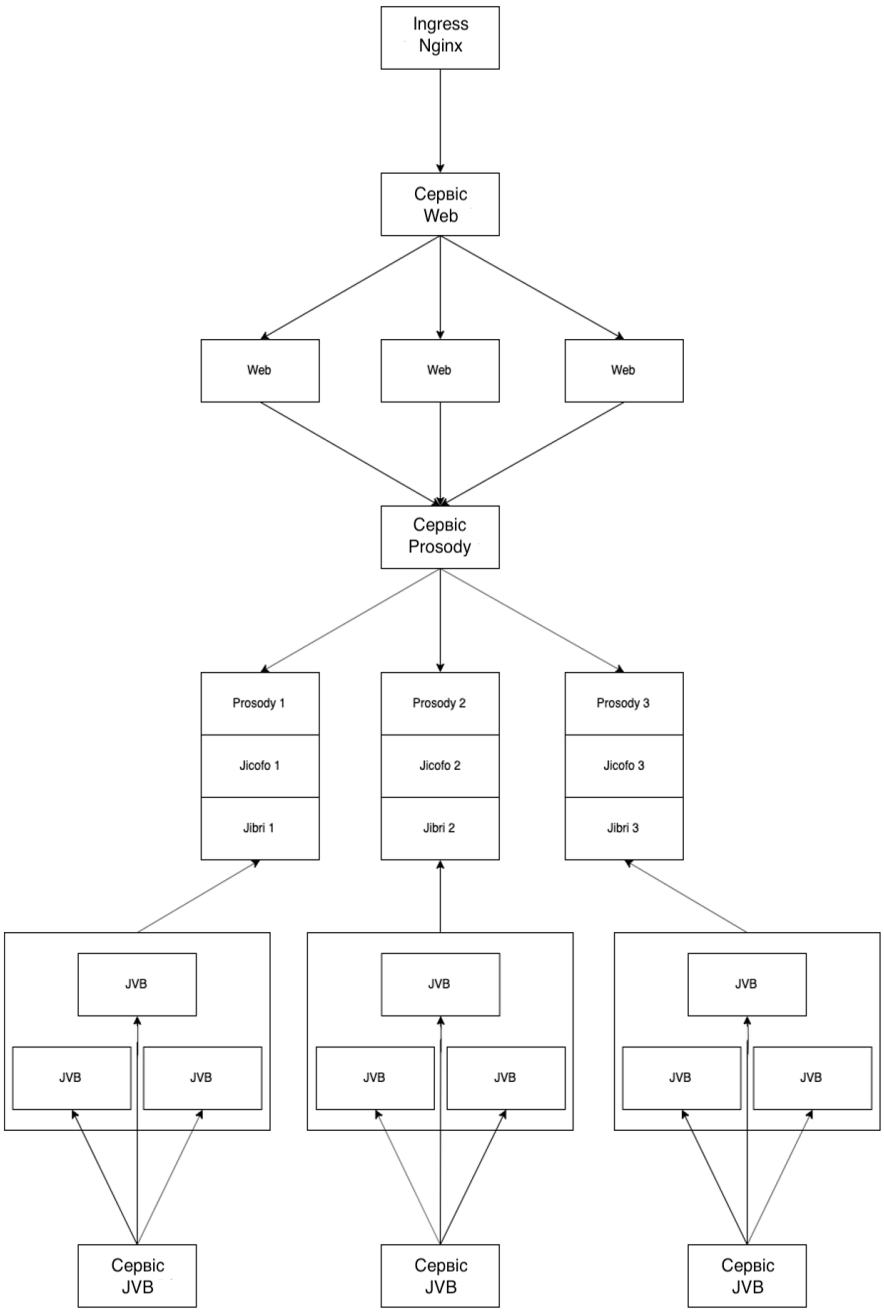

Рис. 4. Мікросервіси SFU Jitsi в Kubernetes.

Ingress nginx – це контролер, який відповідає за вхідний трафік в кластері, перенаправляє запити клієнта на потрібний сервіс. Prosody, JVB, Jibri, Jicofo, Web – це мікросервіси застосунку SFU Jitsi. Prosody – XMPP (мережевий протокол для швидкого обміну повідомленнями та інформацією про присутність) сервер, що використовується для сигналізації. JVB – це WebRTC сумісний сервер, що відповідає за маршрутизацію стрімів між учасниками конференції. Web – вебсервер, що обробляє клієнтські HTTP-запити від веббраузерів. Jibri – це компонент фокусування на стороні сервера, який керує медіа-сесіями і діє як балансувальник навантаження. Jibri – це набір інструментів для запису та/або потокової передачі конференції, який працює шляхом запуску екземпляра Chrome для запису та кодування результату за допомогою ffmpeg.

Таким чином автоматизація розгортання SFU серверів дозволила покращити кількісну та якісну складові експериментів, зменшити час, затрачений на його підготовку, а також мінімізувати вплив та помилки людського фактору.

6. Висновки

Досліджено можливості спрощення експлуатації та розгортання SFU серверів в Cloud Native екосистемі. Встановлено, що розгортання та експлуатація SFU серверів в Cloud Native є можливим та значно ефективнішим під час використання переваг платформи Kubernetes, що відкриває можливості для подальших досліджень в цьому напрямку.

Запропонований процес розгортання було автоматизовано з використанням пакетного менеджера Helm та інструмента розгортання helmfile. Рекомендації з покращення Helm пакета, які було знайдено при експлуатації, доведено до розробника через платформу github [14]. Встановлено, що автоматизація розгортання SFU серверів дозволила покращити кількісну та якісну складові експериментів, зменшити час, затрачений на його підготовку, а також мінімізувати вплив та помилки людського фактору.

Список використаної літератури

1. Heater B. Twitter says staff can continue working from home permanently – TechCrunch. TechCrunch. URL: https://techcrunch.com/2020/05/12/twitter-says-staff-can-continue-working-from-home-permanently/.

2. Pasha M., Shahzad F., Ahmad A. “Analysis of challenges faced by WebRTC videoconferencing and a remedial architecture”. *International Journal of Computer Science and Information Security*. 2016, Vol. 14, No. 10, P. 698-705.

3. Mário Antunes, Catarina Silva, Joaquim Barranca. “Telemedicine application using webrtc”. *Procedia computer science*. 2016, no. 100, P. 414–420.

4. “An innovative WebRTC solution for e-Health services” / Paola Pierleoni, Luca Pernini et all. IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom), Munich, 14 September 2016, 16484380.

5. Kwok-Fai N., Yang L., Wu C. “A P2P-MCU Approach to Multi-Party Video Conference with WebRTC”. *International Journal of Future Computer and Communication*. 2014. Vol. 3, no. 5. P. 319–324.

6. “Performance comparison of a WebRTC server on Docker versus virtual machine” / C. C. Spoiala et al. International Conference on Development and Application Systems (DAS), Suceava, 19 April 2016.

7. “Comparative Study of WebRTC Open Source SFUs for Video Conferencing” / E. André et al. Principles, Systems and Applications of IP Telecommunications (IPTComm), Chicago, 18 October 2018, 18324713.

8. Gannon D., Barga R., Sundaresan N. “Cloud-Native Applications”. *IEEE Cloud Computing*. 2017. Vol. 5, no. 4. P. 16–21.

9. D. Linthicum. “Cloud-Native Applications and Cloud Migration: The Good, the Bad, and the Points Between”. *IEEE Cloud Computing*. 2017. Vol. 5, no. 4. P. 12–14.

10. Вихідний код проекту Jitsi. GitHub. URL: https://github.com/jitsi (дата звернення: 18.01.2022).

11. Документація шаблонізації мови програмування golang. pkg.go.dev. URL: https://pkg.go.dev/text/template (дата звернення: 18.01.2022).

12. Вихідний код helmfile. GitHub. URL: https://github.com/roboll/helmfile (дата звернення: 19.01.2022).

13. Вихідний код розгортання webrtc застосунку в Kubernetes. github.com. URL: https://github.com/kuzm1ch/k8s-webrtc (дата звернення: 18.01.2022).

14. A Helm chart for jitsi-meet by Zempashi Pull Request #235 jitsi/docker-jitsi-meet. GitHub. URL: https://github.com/jitsi/docker-jitsi-meet/pull/235#issuecomment-658381692 (дата звернення: 18.01.2022).

References

1. Heater B. Twitter says staff can continue working from home permanently – TechCrunch. TechCrunch. URL: https://techcrunch.com/2020/05/12/twitter-says-staff-can-continue-working-from-home-permanently/.

2. Pasha M., Shahzad F., Ahmad A. “Analysis of challenges faced by WebRTC videoconferencing and a remedial architecture”. *International Journal of Computer Science and Information Security*. 2016, Vol. 14, No. 10, P. 698-705.

3. Mário Antunes, Catarina Silva, Joaquim Barranca. “Telemedicine application using webrtc. Procedia computer science”. *Computer and information sciences*. 2016, no. 100, P. 414–420.

4. “An innovative WebRTC solution for e-Health services” / Paola Pierleoni, Luca Pernini et all. IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom), Munich, 14 September 2016, 16484380.

5. Kwok-Fai N., Yang L., Wu C. “A P2P-MCU Approach to Multi-Party Video Conference with WebRTC”. *International Journal of Future Computer and Communication*. 2014. Vol. 3, no. 5. P. 319–324.

6. “Performance comparison of a WebRTC server on Docker versus virtual machine” / C. C. Spoiala et al. International Conference on Development and Application Systems (DAS), Suceava, 19 April 2016.

7. “Comparative Study of WebRTC Open Source SFUs for Video Conferencing” / E. André et al. Principles, Systems and Applications of IP Telecommunications (IPTComm), Chicago, 18 October 2018, 18324713.

8. Gannon D., Barga R., Sundaresan N. “Cloud-Native Applications”. *IEEE Cloud Computing*. 2017. Vol. 5, no. 4. P. 16–21.

9. D. Linthicum. “Cloud-Native Applications and Cloud Migration: The Good, the Bad, and the Points Between”. *IEEE Cloud Computing*. 2017. Vol. 5, no. 4. P. 12–14.

10. Project source code Jitsi. GitHub. URL: https://github.com/jitsi .

11. Programming language template documentation golang. pkg.go.dev. URL: https://pkg.go.dev/text/template.

12. Source code helmfile. GitHub. URL: https://github.com/roboll/helmfile.

13. The source code of the webrtc application deployment in Kubernetes. github.com. URL: https://github.com/kuzm1ch/k8s-webrtc.

14. A Helm chart for jitsi-meet by Zempashi Pull Request #235 jitsi/docker-jitsi-meet. GitHub. URL: https://github.com/jitsi/docker-jitsi-meet/pull/235#issuecomment-658381692.