УДК 004.4

ПІДХІД ДО ОРКЕСТРУВАННЯ ВЕБ-СЕРВІСІВ У КЛАСТЕРІ, ЩО ВИКОРИСТОВУЄ РЕСУРСИ ПРИСТРОЇВ У КОРПОРАТИВНІЙ МЕРЕЖІ

Сегеда С.А., Алєксєєв М.О.
Інститут телекомунікаційних систем КПІ ім. Ігоря Сікорського, Україна
E-mail: ti71segeda@gmail.com, alexeyev@its.kpi.ua

AN APPROACH TO WEB-SERVICES ORCHESTRATING IN A CLUSTER USING RESOURCES OF DEVICES CONNECTED TO CORPORATE NETWORK

The work is devoted to the topical problem of efficient execution of web services in a clustered environment. At the moment, there are solutions for orchestration of web services in a cluster, but they are intended primarily for clusters with a stationary architecture, while the use of resources of devices connected to the corporate network implies its frequent changes. The paper proposes an approach based on the use of the Nginx load balancer, which allows you to use the free resources of devices running various operating systems Windows, Linux, macOS connected to the network. The effectiveness of this solution was evaluated using the Artillery server throughput testing tools.

Робота присвячена актуальній проблемі ефективного виконання веб-сервісів у кластерному середовищі. На даний момент існують рішення з оркестрації веб-сервісів у кластері, але вони призначені насамперед для кластерів зі стаціонарною архітектурою, у той час як використання ресурсів пристроїв, що підключені до корпоративної мережі, передбачає її досить часті зміни. У роботі пропонується підхід на основі використання балансувальника навантаження Nginx, який дозволяє задіяти вільні ресурси пристроїв під керуванням різних операційних систем ОС Windows, Linux, macOS, що підключені до мережі. Ефективність такого рішення оцінена за допомогою засобів для тестування пропускної здатності сервера Artillery.

На сьогоднішній день, коли кількість сервісів в інтернеті зростає щосекунди, проблема продуктивності серверів, які їх обслуговують, стоїть дуже гостро. Проте, маючи певні ресурси, цю проблему досить просто вирішити на основі хмарних рішень, які дозволяють побудувати кластер майже будь якої потужності, який буде автоматично масштабуватись, виходячи з потреб. Але в умовах, коли необхідність у такому кластері епізодична, а у наявності є ресурси пристроїв, підключених до корпоративної мережі, більш вигідним може бути залучення простоюючих ресурсів пристроїв в мережі. Це не потребує вкладу грошей і в організаціях є в наявності багато не серверного обладнання - персональні комп’ютери, пристрої “розумного дому” та інтернету речей які не потребують автономності роботи та простоюють.

Отже, у роботі поставлена задача ефективного оркестрування веб-сервісів у кластері, побудованому на основі незадіяних ресурсів пристроїв, підключених до корпоративної мережі. Це гарантую найнижчу вартість володіння такою

системою, але передбачає можливість динамічно додавати або видаляти пристрої з конфігураційного файлу кластера чи балансувальника.

Аналіз існуючих рішень дозволив виділити наступні технології:

1. Dynamic Configuration of Upstreams with the NGINX Plus API.

Рішення для балансування навантаження між робочими нодами кластера с динамічною конфігурацією на основі програмного продукту Nginx plus. В цього сервісу є дуже широкий функціонал, що може задовольнити вимоги у динамічному конфігуруванні кластера [1], але це платний сервіс для великих компаній, ціна якого стартує від тисяч доларів на рік, що зводить нанівець його використання для вирішення поставленого завдання.

2. Kubernetes.

Засоби цієї системи для управління застосунками в контейнерах можна ефективно використовувати для оркестрації. Але Kubernetes зміни у конфігурації кластера сприймає перш за все як виключну ситуацію або це є наслідком автоматичного масштабування кластеру. І, що більш важливе, програмно-апаратні характеристики обчислювальних нод такого кластеру мають бути однаковими для групи нод, що є досить складним для предметної області даної задачі.

3. Docker Swarm.

Досить зручний інструмент для оркестровки, проте немає можливості динамічно змінювати конфігурацію кластера.

4. Mesos.

Працює за принципом Master - Slave, де є головний комп’ютер, який керує іншими. Проте також не можна динамічно додавати або видаляти slave з конфігурації.

5. Nomad.

Розроблений як альтернатива Kubernetes, проте технічно дане рішення відстає від Kubernetes.

6. Flocker.

Досить перспективний інструмент, проте розробка не продовжується починаючи з 2016 року.

7. Helios.

Підтримку завершено після виходу Kubernetes.

Результати аналізу існуючих рішень показують, що для вирішення поставленої задачі необхідно або адаптувати існуючі технології і найбільш релевантним у виборі виглядає Kubernetes, або ж спробувати створити власний оркестратор. У даній роботі розглянутий підхід на основі власної розробки.

Типовою архітектурою сучасного серверу може розглядатись наступна [2]: в нас існує кластер (сервер), у якому працює бізнес логіка, яка обслуговує

застосунок. Для того, щоб запити могли потрапити всередину сервера необхідний балансувальник навантаження, наприклад Nginx або Apache HTTP Server [3]. Все, що робить балансувальник у більшості випадків, це перенаправляє запит у вказане місце, тобто в нашому випадку, на сервер з бізнес логікою. Проте, сервер може бути не один і балансувальник може розподілити запити між ними за заданим алгоритмом.

В даній роботі запропоновано використовувати алгоритм “Round Robin”. При цьому кожен сервер має свою “вагу” і запити розподіляються відповідно до неї [4].

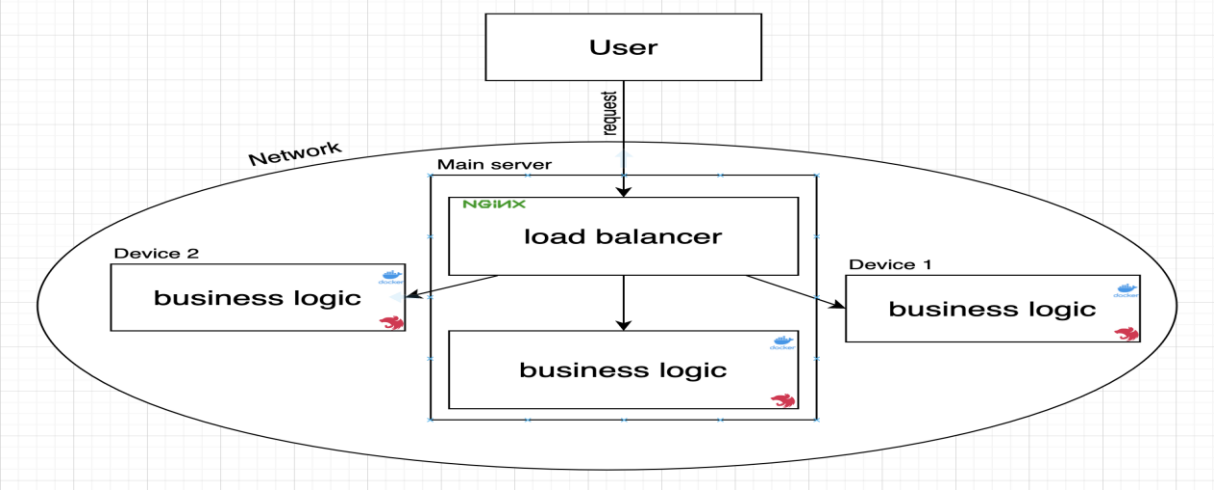

Рис.1. Структурна схема проекту.

Для вирішення поставленої задачі у роботі пропонується наступна архітектура. В нас є мережа, в якій знаходиться головний сервер, в якому в свою чергу знаходиться балансувальник та бізнес логіка упакована в Docker контейнер. Всі комп’ютери які ми хочемо застосувати також мають встановлений Docker і знаходяться в одній мережі з головним сервером, хоча останнє і не є обов’язковим. Docker зручно використовувати для контейнеризації застосунку, бо він дозволяє абстрагуватися від операційної системи, на якій буде запущено додаток, а також дозволяє запакувати все в образ, який потім може бути скачаний з репозиторія для їх зберігання, такого як Docker Hub або AWS ECR. При запуску головний сервер публікує посилання на певній адресі, перейшовши за яким починається завантаження скрипта, який після виконання скачує образ зі схожою бізнес логікою, встановить його та запустить. Бізнес логіка в свою чергу звертається в балансувальник з певним запитом. Для того щоб обробити цей запит використаємо CGI. CGI має певні переваги над іншими методами для зміни конфігурації під час роботи - це дуже легке рішення, яке не використовує багато системних ресурсів, не потребує сторонніх технологій, функціоналу яких буде забагато для цих цілей, має багато варіантів реалізації на різних мовах програмування та не потребує багато

зусиль в розробці [5]. Отримуючи запит, ми маємо ip адресу комп'ютера з якого цей запит було отримано і додаємо його до списку серверів, на які запити будуть транслюватися в подальшому.

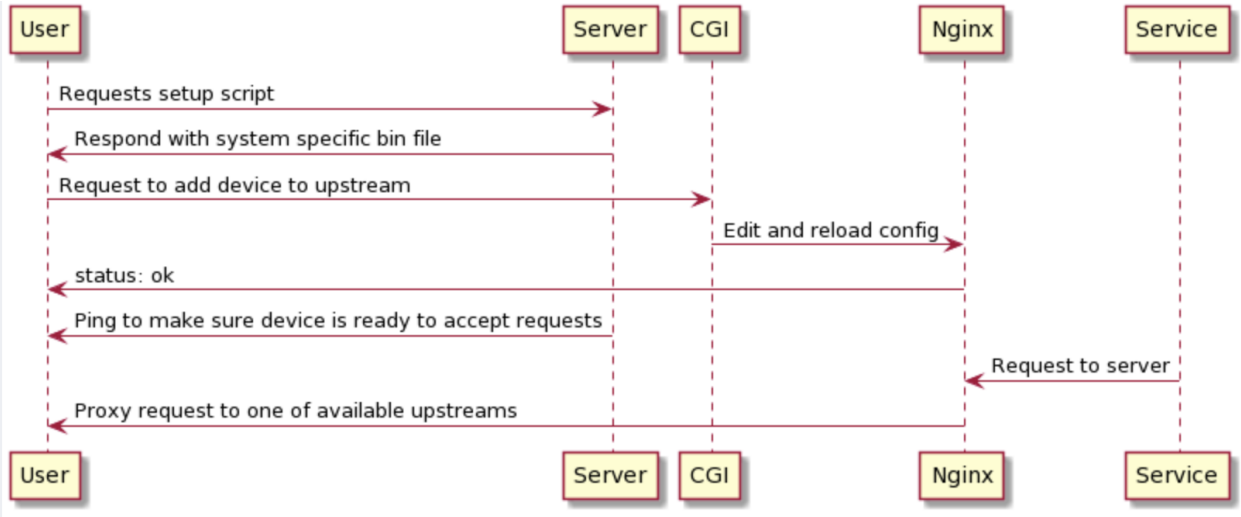

Рис.2. Діаграма станів проекту.

Після цього залишається лише перезавантажити конфігурацію і подальші запити будуть розподілятись між нашими серверами.

Висновки. Не дивлячись на наявність аналогів, запропоноване рішення має переваги над ними, а саме свою легкість у розробці, швидкодію, гнучкість і простоту та відчутно підвищити продуктивність сервера. Маючи не задіяні обчислювальні ресурси комп'ютерів підключених до мережі можна підключити їх до балансувальника навантаження та розподілити запити від користувачів між цими пристроями.

Література

1. Nginx documentation, "Dynamic Configuration of Upstreams with the NGINX Plus API", available at: [https://docs.nginx.com/nginx/admin-guide/load-balancer/dynamic-configuration-api/].
2. Chris Richardson, "Pattern: Microservice Architecture", 2020, available at: [https://microservices.io/patterns/microservices.html].
3. Nginx documentation, "What Is Load Balancing?", available at: [https://www.nginx.com/resources/glossary/load-balancing/#:~:text=A%20load%20balancer%20acts%20as,overworked%2C%20which%20could%20degrade%20performance].
4. Tony Mauro of F5, "Choosing an NGINX Plus Load-Balancing Technique", October 29, 2015, available at: [https://www.nginx.com/blog/choosing-nginx-plus-load-balancing-techniques/].
5. Nginx documentation, "FastCGI Example", available at: [https://www.nginx.com/resources/wiki/start/topics/examples/fastcgiexample/].