Миколайчук В. В., магістрант,
Єфіменко А. А., к.т.н., доцент
Державний університет «Житомирська політехніка»

ДОСЛІДЖЕННЯ МЕТОДІВ ПІДВИЩЕННЯ ВІДМОВОСТІЙКОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ НА БАЗІ КЛАСТЕРУ KUBERNETES НА ПРИКЛАДІ LINUX КЛАСТЕРУ AZURE KUBERNETES SERVICE

Сучасні бізнес-вимоги стимулюють розвиток інформаційних технологій, зокрема в аспекті оптимізації витрат на інфраструктуру. Ключовим елементом у цьому контексті стали контейнери, які дозволяють забезпечувати високу надійність та безперебійність систем при зменшенні оперативних витрат. Використання оркестрації контейнерів через платформу Kubernetes значно спростило управління складними інфраструктурами, зробивши цю технологію стандартом для багатьох компаній. Спеціалісти, що працюють з інформаційними системами, повинні володіти глибокими знаннями Kubernetes для побудови надійних систем, що задовольняють високі стандарти відмовостійкості.

Kubernetes дотримується архітектури master/slave. Його компоненти можна розділити на ті, що керують окремим вузлом, і ті, які є частиною control plane.

Kubernetes control plane – це основний модуль контролю кластеру, який керує навантаженням та комунікаціями у системі. Він складається з таких компонентів, як etcd, сервер API, планувальник, менеджер контролерів.

Вузол Kubernetes (node) – це машини, на яких розгортаються контейнери. Кожен вузол має містити систему виконання, типу Docker чи containerd, а також kubelet, cAdvisor, kube-proxy [1].

Розгортання кластеру Kubernetes повністю вручну, так званий Kubernetes the Hard Way, де кожен етап конфігурації виконується вручну, є досить складним завдання і вимагає детального розуміння ролі кожного компоненту, враховуючи, що деякі компоненти адміністратор кластеру може обирати самостійно (наприклад, Container Network Interface (CNI) плагін). Навіть використання інструментів автоматизованого розгортання кластерів типу kubeadm чи kubespray все одно вимагають велику кількість людських ресурсів та часу для розгортання кластеру, який має бути надійним; кластер, в якому в першу чергу використовуються методи забезпечення високої надійності control plane. Є два основні методи – стокова топологія etcd, де etcd розміщується на усіх control plane вузлах та зовнішня топологія etcd, де розподілений кластер зберігання даних, наданий etcd, є зовнішнім щодо кластера, утвореного вузлами, які запускають компоненти control plane [2].

Для спрощення розгортання хмарні провайдери пропонують рішення, що дозволяються позбутися цієї необхідності в ручному налаштовуванні надійності control plane. В таких кластерах базові компоненти розгортаються автоматично, за допомогою вже готових рішень автоматизації від провайдерів, що невидимі кінцевим користувачам. Прикладом такого хмарного провайдеру є Azure, який пропонує сервіс Azure Kubernetes Service (AKS). В Azure Kubernetes Service control plane розгортається автоматично при створенні кластеру. Але варто враховувати service level agreements (SLA) в яких Microsoft описують зобов'язання перед користувачами щодо безперебійної роботи. Тому в безкоштовній версії AKS Microsoft не надає жодних зобов'язань по безперервній роботі, а лише в платній гарантований час безперебійної роботи для кластерів, що використовують зони доступності складає 99,95%, і для кластерів, що не використовують зони доступності – 99,9%.

Kubernetes був розроблений, як платформа, що призначена для управління контейнерами. Власне контейнери можуть використовуватись для побудови інформаційних систем з використанням мікросервісів, де замість створення однієї монолітної системи проектується дизайн системи з певною кількість малих, незалежних частин (сервісів). Використання такої архітектури дає змогу використовувати основні переваги Kubernetes, де сервіси не залежать одне від одного і, наприклад, оновлення одного з них не буде означати, що вся система стає недоступною для використання протягом оновлення, а лише його певна частина функціоналу буде тимчасово недоступною. Також в ідеальному випадку дані сервіси мають виконувати свою функцію незалежно, без будь-якої інформації про попередні запити чи дії користувача. Такі додатки називаються додатками без збереження статусу (stateless application). Такі додатки мають ряд переваг: простота масштабування, висока доступність, спрощують розробку та тестування.

Kubernetes, крім того, наділений різноманітними механізмами, що гарантують неперервну працездатність додатків. Ці інструменти та можливості включають в себе автоматичне відновлення, масштабування, виявлення та виправлення несправностей, що сприяють стабільності та ефективності кластеру.

В конфігурації сервісів можливо вказати налаштування restart policy, яка буде диктувати у яких випадках потрібно перезапустити сервіс. Можливі варіанти: Always та OnFailure. Always буде означати, що сервіс буде перезапущено завжди, незалежно від вихідного статус коду, а OnFailure, коли код завершення не буде рівний нулю [3].

Додатково також можливо налаштувати перевірки стану роботи додатку (liveness probe), щоб автоматично перевіряти роботу сервісу, не очікуючи вихідного статус коду.

Для перевірки можна вказати свою команду, наприклад, cat /tmp/healthy, яка буде виконуватись з заданою періодичністю і якщо команда буде виконана не успішно, то контейнер сервісу буде в статусі Failed.

Можливо також вказати HTTP перевірку нашого сервісу, наприклад, опитуючи URL Path /health за портом 8080 з певною періодичністю. Ми можемо, наприклад, очікувати певний HTTP Header у відповідь і якщо наш запит не поверне його, то додаток буде позначений в статусі Failed.

Також доступний третій тип перевірки за допомогою опитування додатку, що використовує TCP сокет. З цією конфігурацією kubelet намагатиметься відкрити сокет до вашого контейнера на вказаному порту. Якщо він може встановити з'єднання, контейнер вважається справним, якщо не може, він вважається несправним [4].

У випадках коли додаток інформаційної системи перебуває під високим навантаженням, можливо застосувати горизонтальне (horizontal pod autoscaling) або вертикальне масштабування (vertical pod autoscaling). Знову ж варто зазначити, що додаток має працювати без попереднього збереження даних, тоді дані типи масштабування будуть працювати коректно.

Можливі сценарії, коли під час масштабування подів на вузлах Kubernetes будуть закінчуватись доступні обчислювальні ресурси (CPU, memory), тоді потрібно застосовувати горизонтальне або вертикальне масштабування вузлів. Можливо, також налаштувати правила для масштабування вузлів за певними показниками, такі як кількість подів (дане правило можна використати, коли максимальна кількість подів на вузлі є прогнозованою).

Отже, Kubernetes має ряд вбудованих механізмів забезпечення відмовостійкості, які допомагають інженерам підвищити відмовостійкість систем. Але самі додатки мають бути спроектовані належним чином зі stateless підходом, для можливості використання конфігурацій, доступних в Kubernetes для забезпечення надійності інформаційних систем, що використовують мікросервісний підхід.

Список використаних джерел

1. Kubernetes – Архітектура
URL: https://uk.wikipedia.org/wiki/Kubernetes#Архітектура.
2. Options for Highly Available Topology URL: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/.
3. Container restart policy URL: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#restart-policy.
4. Configure Liveness, Readiness and Startup Probes URL: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/.