УДК 004.75:004.056.5

Орловський Д.І. студент гр. РТ-818 НУ «Запорізька політехніка», Бакум А.О. слухач гр. 4208 Національний університет оборони України
Науковий керівник: Куцак С.В., старший викладач кафедри захисту інформації
(*Національний університет «Запорізька політехніка», м. Запоріжжя, Україна*)

ВИКОРИСТАННЯ МЕХАНІЗМУ КОНТЕЙНЕРИЗАЦІЇ З МЕТОЮ ПОБУДОВИ БЕЗПЕЧНОЇ КЛІЄНТ-СЕРВЕРНОЇ ІНФРАСТРУКТУРИ

Сучасний стан розробки програмного забезпечення констатує перехід від монолітних сервісів (створюваних для персональних комп'ютерів) до великих хмаро орієнтованих – для великих дата-центрів. Переваги використання хмарних технологій, з точки зору користувачів і бізнесу, є зрозумілими: можливість створення програмних засобів, які будуть працювати однаково ефективно, незалежно від потужності машини користувача; більш гнучкі механізми передплати програмного забезпечення.

Переважно хмарні системи у своїй інфраструктурі використовують мікросервіси: бази даних, обчислювальні сервери, сервери кешування, веб-сервери, балансувальники навантаження. До організації розробки програмного забезпечення для такої хмарної інфраструктури висувається ряд вимог, що полягають в недопущенні (мінімізації) програмно-апаратних збоїв, несанкціонованого доступу до оброблюваної інформації, фінансових та репутаційних втрат для бізнесу.

В даній роботі досліджується сучасний підхід для вирішення зазначених проблем – використання DevOps методології, яка знаходиться на межі розробки програмного забезпечення та системного адміністрування [1]. Основна мета DevOps – це автоматизація розробки програмного забезпечення шляхом використання єдино-довіреного сервісу «розгортання» та управління застосунків, для досягнення якої в DevOps наявні різні інструменти: Docker, GitLab CI/CD, GitHub CI/CD, Jenkins, CHEF, VictorOps.

Особливе місце в цій методології займає Docker – програмне забезпечення з відкритим програмним кодом, яке використовується як інструмент з автоматизації та універсалізації розгортання, підтримки програмного забезпечення та управління ним в ізольованих «контейнерах». Docker-контейнери можуть бути запущені у різних середовищах (локальній мережі, хмарі) незалежно від операційної системи (підтримує як UNIX системи, так і MS Windows).

За замовчуванням Docker має у своїй конфігурації daemon - сервер (систему, яка дозволяє запускати процеси у фоновому режимі) з можливістю налаштування контейнера за допомогою командного терміналу [2]. Daemon-сервер забезпечує: ізоляцію програмного застосунку у контейнері на рівні файлової системи (наявність власної файлової системи у кожного контейнера); на рівні розділення процесів (налаштування лімітів на системні ресурси); на мережному рівні (наявність симуляції мережного інтерфейсу, що ізолює один трафік програмного застосунку від іншого).

Механізм контейнеризації дозволяє системному адміністратору легко інсталювати інфраструктуру тестового сервера з використанням тестових конфігурацій баз даних, внутрішнього стану сервера тощо. Під час виникнення кіберінцидентів даний механізм дозволяє розпочати швидкий аудит системи. Подібний функціонал реалізують такі інструменти як Prometheus, Jenkins.

Одним із засобів організації безпечної інфраструктури є підтримка контейнерами використання bash-скриптів, що допомагає автоматизувати процес резервного копіювання інформації з контейнера на захищений файловий сервер.

Можливим є адміністрування життєвого циклу застосунку через механізм залежності та «зв'язування» контейнерів. Так, при зупинці центрального контейнера вимикаються залежні від нього бази-даних, сервіси кешування [3].

Docker підтримує «багаторівневу» побудову коду: під час збирання та ініціалізації коду всередині контейнера виконуються всі необхідні команди, а сам контейнер у фінальній версії має лише готовий до виконання код. Використання такого підходу зменшує кількість зв'язаних залежностей, що мінімізує поверхню кібератаки [4].

Існують вбудовані механізми мережної ізоляції інфраструктури. Наприклад, реальною є атака на відмову в обслуговуванні СУБД. Переважно кожна СУБД має загальновідомий порт за замовчуванням (для PostgresSQL - 5432, для MS SQL - 1433), який зловмисник може експлуатувати зі злочинною метою. Запобігаючи цьому, Docker блокує всі зовнішні мережеві підключення, відкриваючи їх лише вказаним контейнерам, створюючи локальний мережевий «міст» між ними.

На рисунку 1 зображено схематичний приклад використання Docker-контейнеризації у реальному застосунку: на серверній хост-машині розгорнуті два мікросервіси (веб-сервер, база даних), кожен з яких знаходиться в окремому контейнері. Ці контейнери зв'язані між собою завдяки ізольованому мережному каналу зв'язку та комунікують шляхом RESTfull API. Не дивлячись на те, що обидва сервіси мають два власні порти для комунікації всередині контейнера (5432 для бази даних, 3000 для веб-сервера), відкритим для комунікації ззовні є лише 80 порт (запити переадресовуються до 3000 порту у веб-серверному застосунку).

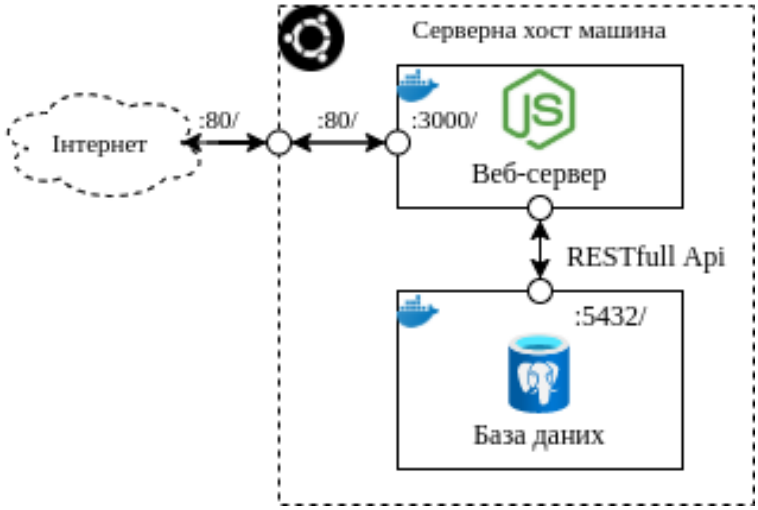

Рисунок 1 – Типовий приклад використання Docker-контейнеризації

Таким чином, провівши аналіз використання DevOps методології, можемо констатувати переваги використання механізму контейнеризації під час процесу розробки та експлуатації програмного застосунку. Даний механізм значно підвищує швидкість та безпечність розробки програмного забезпечення через підтримку різноманітних конфігурацій налаштування ресурсів сервера, які в сукупності забезпечують конфіденційність, цілісність та доступність хмаро орієнтованих сервісів.

Перелік посилань

1. The Guide to Docker DevOps. URL: https://dinarys.com/blog/the-guide-to-docker-devops (last accessed: 12.10.2021).

2. Configure and troubleshoot the Docker daemon. URL: https://docs.docker.com /config/daemon/ (last accessed: 12.10.2021).

3. Docker security. URL: https://docs.docker.com/engine/security/ (last accessed: 12.10.2021).

4. Docker Security Tips & Best Practices. URL: https://www.threatstack.com/blog/ docker-security-tips-best-practices (last accessed: 12.10.2021).