DOI: https://doi.org/10.36910/6775-2524-0560-2021-45-15
УДК 004.31
Чернящук Наталія Леонідівна, д.п.н., проф.,
https://orcid.org/0000-0002-3178-8377
Багнюк Наталія Володимирівна, к.т.н., доц.,
https://orcid.org/0000-0002-7120-5455
Бойко Максим Валерійович, магістр
Чикірда Микола Валерійович, магістр
Савонюк Назарій Сергійович, магістр
Луцький національний технічний університет

ПРОЦЕС РОЗГОРТАННЯ КЛАСТЕРНОГО ДОДАТКУ В KUBERNETES

Чернящук Н.Л., Багнюк Н.В., Бойко М.В., Чикірда М.В., Савонюк Н.С. Оптимізація процесу розгортання кластерного додатку в системі Kubernetes. У статті проаналізовано деякі можливості що забезпечуються хмарними технологіями. Порівняно підхід для реалізації однієї з таких систем Kubernetes – найбільшими постачальниками хмарних послуг. Розглянуто механізм розгортання додатку на платформі оркестрації контейнерів.
Ключові слова: розгортання кластерного додатку, система Kubernetes, хмарні технології, платформа оркестрації контейнерів.

Чернящук Н.Л., Багнюк Н.В., Бойко М.В., Чикирда Н.В., Савонюк Н.С. Оптимизация процесса развертывания кластерного приложения в системе Kubernetes. В статье проанализированы некоторые возможности, обеспечиваемые облачными технологиями. По сравнению подход к осуществлению одной из таких систем Kubernetes - крупнейшими поставщиками облачных услуг. Рассмотрены механизм развертывания приложения на платформе оркестрации контейнеров.
Ключевые слова: развертывание кластерного приложения, система Kubernetes, облачные технологии, платформа оркестрации контейнеров.

Chernyashchuk N, Bahniuk N., Boyko M, Chikirda N, Savonjuk N. Optimization of the cluster application deployment process in the Kubernetes system. The article analyzes some of the opportunities provided by cloud technologies. The approach to the implementation of one of these systems Kubernetes - the largest providers of cloud services. The mechanism of application deployment on the container orchestration platform is considered.
Keywords: cluster application deployment, Kubernetes system, cloud technologies, container orchestration platform.

Постановка проблеми.

Кожен веб додаток має бути опублікованим в мережі інтернет, бути доступним за запитом користувачів, оновнюватись новими налаштунками, зберігати дані і т.ін. Існує безліч способів розгорнути додаток в мережі, але основними принципами є – існування фізичного простору для зберігання даних (фізичний сервер, віртуальний хостинг, віртальний виділений сервер, VPS-хостинг або ж хмара), унікальнаий домен або адреса сайту.

Отже, щоб розгорнути деякий додаток в мережі необхідно, перевірити доступність доменного імені, зарезервувати його. Обрати надійний хостинг, оформити підписку та завантажити файли додатку через менеджер сайтів (FTP –клієнт) на хостинг. Як бачимо, послідовність кроків досить нескладна, але основна проблема у терміні «надійний». Так, зокрема звичайнй фізчний сервер або віртуальний хостинг – як машина для зберігання даних з мінімальним набором операційних послуг є досить нестабільною у використанні. І навіть, якщо не брати до уваги можливі перебої у його роботі, що відбувається доволі часто, основною проблемою є складність розширення ресурсів, оновлення додатку та відказостійкість. З ростом додатку, зростає потреба в обчислювальних ресурсах, так, їх можна додати, проте збільшення потужності сервера зазвичай не пропорційне збільшенню його вартості. Також, виникає ризик простою ресурсів, тоді як зворотнє масштабування знову викличе труднощі і додаткові витрати. Ще одним важливам фактором є неможливість перенесення даних у випадку відсутності доступу, оскільки звичайний хостинг не передбачає резервного копіювання чи бекапів даних.

Процес розгортання веб додатку в мережі.

Все ж для багатьох, зокрема, невеликих проектів віртуальний хостинг чи фізичний сервер є ідеальним рішенням, оскіль надає користівачу певний рівень безпеки, персональний дисковий простір, операційну память, потужності процесора та повний контроль над виділеними ресурсами. Та для масштабованих, багатокомпонентних, постійно розвиваючих проектів хостинг технології є складними у використанні. Тому розглянемо процес розгортання додатку, що потребуватиме активного розширення в майбутньому з використанням хмарних технологій та контейнеризації.

Наведемо таблицю порівняння послуг що пропонуються віртуальним хостингом та хмарним провайдерами для розгортання додатків:

Таблиця 2. Порівняння можливостей розгортання додатків на хостингах та хмарах

	Хостинг	Хмара
Віртуалізація	Віртуальні машини на одному сервері	Контейнери, розподілені між декількома фізичними серверами
Масштабованість	Можлива, але складно організована	Автоматична
Виділення додаткових ресурсів	Тривалий процес	Миттєво
Ресурси памяті та процесора	Розділювані	Ізольовані
Вибір ОС	Пропонується хостом	Будь-яка
Вартість	Низька	Середня
Надійність	Непередбачувана	Висока

Отже, нехай ми маємо деякий додаток з мікросервісною архітектурою, кожен мікросервіс може бути розроблений будь якою мовою програмування, та для взаємодії з іншими мікросервісами має бути сформований декий інтерфейс. Далі ці мікросервіси треба запакувати в окремі контейнери, де вони вже будуть працювати а не просто являтимуть собою набір файлів. При цьому буде створено образ сервісу і його конфігурація занотована в Dockerfile. А вже наступним кроком потрібно встановити зв'язки між такими контейнерами, забезпечити їм необхідну конфігурацію та запустити в мережі, для цих операцій потрібні сервіси оркестрації контейнерів такі як Kubernetes, OpenShift, Docker Swarm та ін. Також ці сервіси дозволяють проводити моніторинг роботи як окремого контейнера так і їх зв'язків, налаштовувати автоматичне налаштування для контейнерів та керувати їх роботою.

Створення кластера Azure Kubernetes.

Перш ніж розгорнути будь-яку програму, потрібно створити кластер. Розглянемо кілька концепцій, для розгортання нового кластера, зокрема в середовищі Azure Kubernetes Service (AKS). Kubernetes базується на кластерах, замість того, щоб мати єдину віртуальну машину (ВМ), вона використовує кілька машин, що працюють як одна. Ці ВМ називаються вузлами. Kubernetes - це кластерний оркестратор, що надає переваги доступності, моніторингу, масштабування та оновлення. Кластер базується на вузлах - нодах. У кластері Kubernetes є два типи вузлів, які забезпечують функціональність: Вузли управління - розміщують параметри управління кластера і зарезервовані для служб, які керують кластером. Вони відповідають за надання API, який всі інші вузли використовує для комунікації. На цих вузлах не розгортається навантаження; Ноди – вузли, що відповідають за виконання нестандартних робочих навантажень та додатків, таких як компоненти хмарної служби. Існує дві архітектури кластерів для розгортання на основі Kubernetes: Єдина система управління та кілька вузлів - найпоширеніший архітектурний шаблон. Цей шаблон найпростіший у розгортанні, але він не забезпечує високої доступності основних служб управління кластером. Якщо вузол управління стає недоступним з будь-якої причини, ніякої іншої взаємодії з кластером не може відбутися. Незважаючи на, можливу недоступність, цієї архітектури зазвичай достатньо. Менше ймовірно, що основні служби управління стануть недоступними порівняно з вузлом, який переходить у офлайн-режим. Вузли управління піддаються меншій модифікації, ніж ноди, і більш еластичні. Єдина система управління та єдиний вузол. Ця архітектура забезпечує лише один вузол, який розміщує управління і робочий вузол. Це корисно під час тестування або експериментування з різними

концепціями Kubernetes. Єдина площина управління та архітектура одного вузла обмежує масштабування кластера і робить цю архітектуру непридатною для виробничого та інтерактивного використання. Для групування вузлів створюється Node pools, де вказується розмір віртуальної машини кожного вузла. Пули вузлів використовують набори масштабів віртуальних машин як базову інфраструктуру, щоб дозволити кластеру масштабувати кількість вузлів у пулі вузлів. Нові вузли, створені у пулі, завжди матимуть розмір, вказаний під час створення пулу вузлів [13].

Кластер Kubernetes за замовчуванням блокує всі зовнішні зв'язки. Наприклад, розгортаючи веб-сайт, доступний для всіх клієнтів, необхідно вручну створити доступ або ж спеціально нелаштувати. Конфігурація вимагає змін, пов'язаних з мережею, які переадресовують запити від клієнта на внутрішній IP-адрес кластера а потім, до додатка.

AKS дозволяє подолати складність, увімкнувши так звану маршрутизацію додатків HTTP. Ця надбудова полегшує доступ до програм на кластері за допомогою автоматичного розгортання контролера входу [20].

Контролери Ingress надають можливість розгортати та виставляти програми у всьому світі без необхідності налаштування мережевих служб.

Схема маршрутизації додатків HTTP, яка показує, як контролер входу сканує вхідні ресурси та створює правила DNS, щоб зробити програми доступними для зовнішніх клієнтів. Контролери Ingress створюють сервер зворотного проксі-сервера, який автоматично дозволяє обслуговувати всі запити з одного виходу DNS. Не потрібно створювати запис DNS кожного разу, коли розгортається нова служба. Про це потурбується контролер входу.

Коли новий вхід розгортається в кластері, контролер входу створює новий запис у керованій Azure зоні DNS і пов'язує його з існуючим балансером навантаження. Ця функціональність забезпечує легкий доступ до ресурсу через Інтернет без необхідності додаткової конфігурації. Для створення нового кластеру для розгортання серверу в систем Azure Kubernetes конфігурація буде налаштовуватись у Cloud Shell. Та найперше, для користування сервісами Azure необхідно зареєструватися у системі та створити підписку, з якою власне і буде звязано розгорнутий додаток.

Отже, найперше – створимо групу, що включатиме кластери за допомогою конанди az group create, вказавши обов'язкові параметри:

```
az group create --name rg-MicroExample --location eastus
```

Далі, в цій групі ствоимо сам кластер, вказавши основні параметри:

```
az aks create \
    --resource-group rg-MicroExample \
    --name aks-MicroExample \
    --node-count 2 \
    --enable-addons http_application_routing \
    --dns-name-prefix example-kubernetes-$RANDOM \
    --generate-ssh-keys \
        --node-vm-size Standard_B2s
```

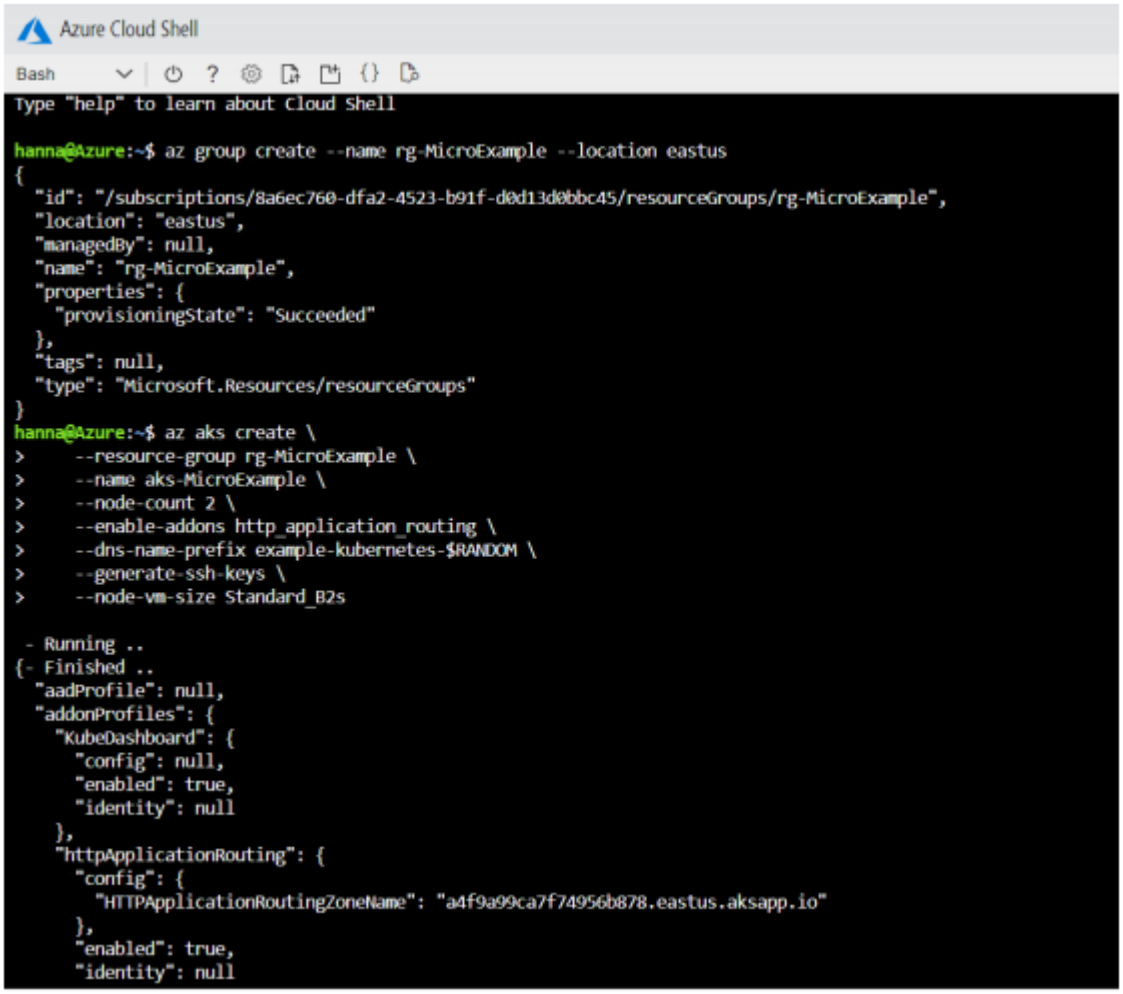

Рисунок 1 встановлення конфігурації в Cloud Shell

Команда az aks create створить кластер з назвою aks-MicroExample та у групі rg-MicroExample, що буде складатися з 2х нод, маршрутизація до кластера буде здійснюватись через http_application_routing до ДНС кластера example-kubernetes, а розмір віртуальної машини задано Standard_B2s.

Наступним кроком необхідно встановити зв'язок кластера і групи:

```
az aks get-credentials --name aks-MicroExample --resource-group rg-
MicroExample
```

команда додасть конфігуацію в файл управління, який містить всю інформацію для доступу до кластерів. Kubectl дозволяє управляти кількома кластерами з одного інтерфейсу командного рядка. Для перевірки створеної конфігурації існує команда kubectl get nodes, що поверне інформацію по обом створеним нодам: NAME, STATUS, ROLES, AGE, VERSION

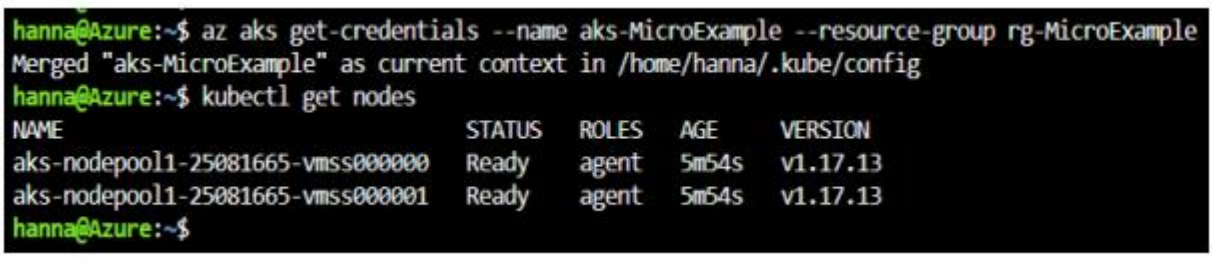

Рисунок 2 статус створеної конфігурації

Важливим для налаштування контейнеру є файл маніфест Kubernetes, що дозволяє декларативно описувати навантаження у форматі YAML та спрощувати управління об'єктами Kubernetes. І містить всю інформацію, необхідну для створення та управління робочим навантаженням.

Структура файлів маніфесту відрізняється залежно від типу ресурсу. Однак файли маніфесту мають спільні інструкції, якы визначають різні аспекти, наприклад API, потрібні для використання, та тип робочого навантаження для створення. Зокрема, файл маніфесту обов'язково має включати основні елементи:

```
apiVersion: apps/v1
kind: Deployment
metadata:
        name: micro-website # the name of the deployment
```

Параметр apiVersion визначає API ендпоінт сервера, що управляє об'єктом, який розгорнете. Параметр kind визначає навантаження, яке створить це розгортання. Інші загальні параметри - це метадані та ключі імен. Усі ресурси Kubernetes повинні мати назву, що знаходиться всередині параметра метаданих. Щоб застосувати метафайл для встановлення системи використовують команду `kubectl`:

```
kubectl apply -f ./deploy.yaml
```

Наступним кроком є створення образу контейнера, і розміщення його в реєстрі контейнера. Azure забезпечує Azure Container Registry для зберігання зображень (образів) контейнерів, а команда

```
az  acr  create
```

дозволяє створити такий образ:

```
az acr create \
    --resource-group rg-MicroExample \
    --name ACRMicroExample\
        --sku Basic
```

Але щоб пов'язати кластер ресурсної групи та образ контейнера, необхідно обновити кластер командою az aks update зв'язавши всі ці три параметри:

```
az aks update \
    --name aks-MicroExample \
    --resource-group rg-MicroExample \
        --attach-acr ACRMicroExample
```

Розгорнемо для прикладу деяку програму збережену в гіт хабі що вже має конфігурацію докера і відповідно, являє собою контейнер. Для цього склонуємо файли додатку в створений образ контейнера

git clone https://github.com/hannagm2016/MicroExample.git

а потім в Cloud Sell перейдемо в створену дерикторію за допомогою
команди cd:
cd MicroExample

Щоб згенерувати апустити контейнер, необхідно виконати команду az acr build з наступними параметрами:

```
az acr build \
    --image Example-website \
    --registry ACRMicroExample \
        --file Dockerfile .
```

Параметр --image додає до образу тег веб-сайту Example-website. Зображення автоматично надсилається до реєстру після успішного завершення збірки.

Після створення всіх вищеописаних елементів, необхідно додати їм параметри в конфігурацію – уже існуючий маніфест файл deploy.yaml

Потрібно описати под, що є шаболоном специфікації. Назва пода буде згенерована автоматично. Щоб деплоймент згрупував поди, необхідний тег labels Под обєднує в собі контейнери, кожен з яких містить інформацію під тегом app: щодо контейнера. Отже, треба оновити файл даними про под:

```
    spec:
    template: # Шаблон пода деплоймента
      metadata: # Метадані pod
      labels:
            app: Example-website
```

Також в файл конфігурації, після описання поду варто додати інформацію про контейнер, зокрема конфігурацію посилання на розгорнутий сайт:

```
    spec:
    containers:
    - image: <acr_name>.azurecr.io/Example-website
    name: contoso-website
```

Обмеження щодо використовуваних ресурсів, зокрема мінімальне та максимальне навантаження процесора, виділену пам'ять:

```
    resources:
          requests:
            cpu: 50m
            memory: 128Mi
          limits:
            cpu: 200m
                memory: 256Mi
```

А також дані порта, на якому буде сконфігуровано сайт:

ports:

- containerPort: 80

name: http

Далі необхідно зберегти змінений файл конфігурації, і засосувати зміни за допомогою команди kubectl apply

```
kubectl apply -f ./deploy.yaml
```

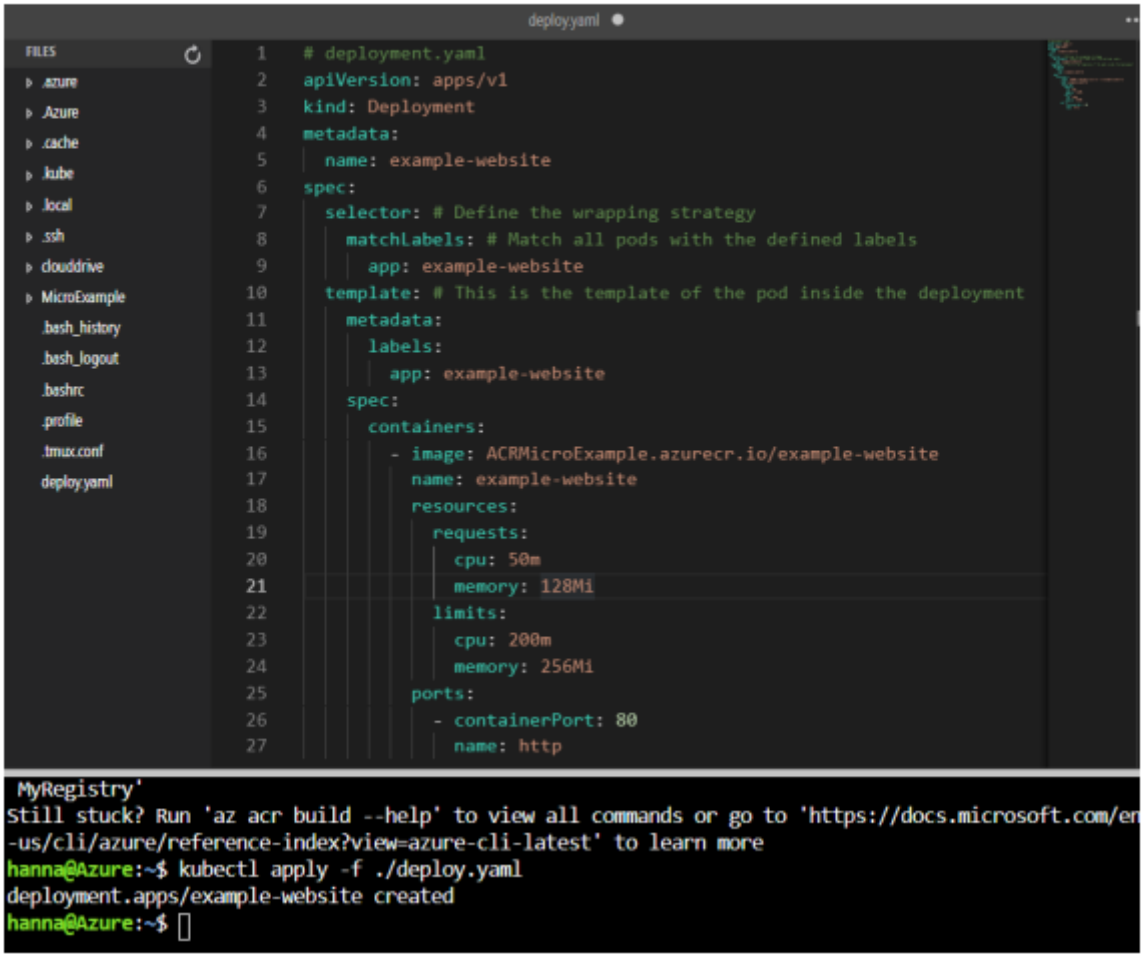

Рисунок 3 Застосувнання файлу конфігурації

Щоб перевірити успішність деплойменту, запуску пода і переглянути його назву, треба виконати команду: kubectl get pods

Отриманий результат матиме вигляд:

```
NAME                               READY   STATUS             RESTARTS   AGE
example-website-75bfc74bc-cx764    0/1     ImagePullBackOff   0          2m29s
```

Рекомендації щодо розгортання контейнерного додатку.

Одним з завдань роботи є розроблення рекомендацій для розгортання додатку з використанням служби Kubernetes. З огляду на всі вищеописані методи і технології, найперше – з огляду на складність процесу кластеризаціїї, потрібно визначити межі проекту, і встановити чи потрібна взагалі така методологія для розгортання веб сервісу, можливо послуги веб хостингу або просто хмарного хостингу цілком задовольнять потреби проекту. Також, хмарні провайдери пропонують широкий спектр послуг Paas та навіть Saas які можуть значно спростити розробку додатку і прискорити процес отримання готового рішення.

Та з огляду на масштабність проекту, звичайно, деплоймент мікросервісного додатку що вже розміщений в контейнері в системі оркестрації контеєнерів є цілком обгрунтованим та слушним. Загальна схема має складатися з таких етапів:

- Розгортання контейнерів за допомогою системи оркестрації типу Kubernetes, та налаштування необхідної їм конфігурації:

- Створення кластеру в групі ресурсів;
- Розміщення одного чи декількох контейнерів в кластері;
- Створення конфігурації для деплойменту, встановлення параметрів мережі та прав доступу.
- Деплоймент такого кластеру з використанням вищезазначених конфігурацій.
- Моніторинг і управління роботи кластерів в системі оркестрації.

Такий підхід надає проекту високу мобільність, відмовостійкість, продуктивність і швидкодію, а головне, повну автоматичну масштабованість і ефективне використання ресурсів.

Висновки.

В даній статті було порівняно два підходи до деплойменту веб додатку: з використанням хостингу та хмарних технологій. Виділено головну відмінність, що не може бути вирішена в рамках віртуальних хостингів. З використаням офіційної документації служби Azure Kubernetes розроблено оптимізоване розгортання контейнеру. Розроблено рекомендації щодо розгортання кластероного додатку за допомогою системи оркестрації.

Список бібліографічного опису

1. Арундел Д., Домингус Д. - Kubernetes для DevOps: развертывание, запуск и масштабирование в облаке, - Бестселлеры O'Reilly, 2020, 384 с.
2. Загороднюк А.О. Архитектура Docker - Молодой исследователь: вызовы и перспективы: материалы LXIX междунар. науч.-практ. конф. – № 16(69). – М., «Интернаука», 2018. – 482 с.
3. Марко Лукша: Kubernetes в действии, Переводчик: Логунов А. В.: ДМК-Пресс, 2019 г.
4. Сайфан Д. Осваиваем Kubernetes. Оркестрация контейнерных архитектур, Питер. – 2019, 400с.
5. Мікросервісна архітектура [Електронний ресурс] – Режим доступу до ресурсу: - https://medium.com/@IvanZmerzlyi/microservicesarchitecture-461687045b3d.
6. Начало работы с Azure Kubernetes [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/ru-ru/services/kubernetesservice/#getting-started
7. Основы Кубернетис [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/258443/
8. Хмарні обчислення [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Cloud_computing

References.

1. Amazon, Microsoft Or Google: Which Is The Best Play On Surging Cloud Infrastructure Demand? [Електронний ресурс] – Режим доступу до ресурсу: https://www.forbes.com/sites/greatspeculations/2020/05/14/amazonmicrosoft-or_google-which-is-the-best-play-on-surging-cloud-infrastructuredemand/
2. Comparing Kubernetes Services on AWS vs. Azure vs. GCP [Електронний ресурс] – Режим доступу до ресурсу: https://www.sumologic.com/blog/kubernetes-aws-azure-gcp/
3. Deploy a containerized application on Azure Kubernetes Service [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/learn/modules/aks-deploy-container-app/
4. Developers bring their ideas to life with Docker [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/why-docker
5. Nikhil Buduma Fundamentals of Deep Learning. Designing nextgeneration machine intelligence algorithms. O'Reilly Media, 2017. 277 c
6. Gens, Frank (2009-10-05). IDC's New IT Cloud Services Forecast: 2009-2013
7. DevOps [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/devops-a-complete-guide
8. Kubernetes blog [Електронний ресурс] – Режим доступу до ресурсу: https://cloudacademy.com/blog/category/kubernetes/
9. Kubernetes on AWS [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/ru/kubernetes/
10. Microsoft learn [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/learn/browse/
11. Why (and when) you should use Kubernetes [Електронний ресурс] – Режим доступу до ресурсу: https://hackernoon.com/why-and-when-youshould-use-kubernetes8b50915d97d8
12. Why Google Cloud [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/why-google-cloud