

СИСТЕМА АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ КЛАСТЕРНОГО СЕРЕДОВИЩА

У статті розглянуто питання швидкого розгортання та підтримки складних ІТ-інфраструктур за умов зростання навантаження або відмов окремих компонентів. Показано, що одним із варіантів розгортання складних структур є застосування контейнерних технологій з використанням кластерних обчислювальних середовищ і систем оркестрації контейнерів.

В матеріалі були проаналізовані поняття контейнеризації. Показано, що такий підхід забезпечує ізоляцію застосунків разом із їхніми залежностями, що гарантує портативність програмного забезпечення, його однакове функціонування в різних середовищах та можливість автоматизації процесів розгортання.

Розглянуто основні характеристики кластерних систем. Визначено, що об'єднання незалежних обчислювальних вузлів у єдину логічну мережу дозволяє забезпечити високу доступність, масштабованість і стійкість до відмов. Проаналізовано основні характеристики кластерів, такі як масштабованість, відмовостійкість, балансування навантаження. Розкриті мережеві вимоги для функціонування кластерних середовищ.

Показано, що для системи автоматизованого розгортання кластерних середовищ контейнерна модель є пріоритетною, оскільки вона забезпечує швидке масштабування, ефективне використання ресурсів і тісну інтеграцію з системами оркестрації.

Проаналізовано систему оркестрації контейнерів, яка використовується для автоматизації розгортання, керування, масштабування та підтримки контейнеризованих застосунків Kubernetes. Показано можливість ефективного управління великою кількістю контейнерів у складних інформаційних системах.

Для проведення експериментальних досліджень запропонована, описана, реалізована і протестована на власній інфраструктурі практична реалізація системи автоматизованого розгортання і управління технологіями контейнеризації у кластерному середовищі Kubernetes.

Ключові слова: кластерні обчислювальні середовища, контейнеризація, оркестрація, Kubernetes, автоматизація розгортання, програмні застосунки.

Вступ та постановка проблеми. Сучасні інформаційні системи стають складнішими, а вимоги до їх надійності, масштабованості та безпеки постійно зростають. Організації все частіше стикаються з необхідністю швидкого розгортання та підтримки складних ІТ-інфраструктур, здатних забезпечити безперебійну роботу сервісів навіть за умов зростання навантаження або відмов окремих компонентів. У таких умовах традиційні підходи до розгортання та адміністрування програмного забезпечення виявляються недостатньо ефективними.

Одним із ключових напрямів розвитку сучасних ІТ-інфраструктур стала контейнеризація, яка дозволяє ізолювати програмні застосунки разом із їхніми залежностями та забезпечити їхнє однакове функціонування в різних середовищах. Контейнерні технології спрощують розгортання, оновлення та масштабування програмних систем, а також дають змогу ефективніше використовувати обчислювальні ресурси. У поєднанні з кластерними обчислювальними середовищами контейнеризація формує основу для побудови гнучких, відмовостійких і масштабованих систем.

З цієї точки зору важливе значення мають системи оркестрації контейнерів, які автоматизують управління великою кількістю контейнеризованих застосунків. Такі системи беруть на себе завдання розподілу навантаження, масштабування, моніторингу стану

компонентів та відновлення роботи сервісів у разі збоїв. Завдяки цьому зменшується кількість ручних операцій з боку адміністратора та знижується ймовірність помилок, пов'язаних із людським фактором.

Актуальність дослідження пов'язана також зі зростанням значення власної інфраструктури в умовах підвищених вимог до інформаційної безпеки. Для багатьох організацій, особливо у сфері державного управління, критичної інфраструктури або корпоративних мереж, використання публічних хмарних сервісів є обмеженим або небажаним. У таких випадках постає завдання розгортання сучасних кластерних середовищ безпосередньо на власних обчислювальних ресурсах із забезпеченням належного рівня захисту інформації, контролю доступу та стабільності роботи системи.

Тому особливої уваги потребує автоматизоване розгортання кластерного середовища. Автоматизація дозволяє не лише скоротити час налаштування інфраструктури, а й забезпечити відтворюваність конфігурацій, спростити подальше обслуговування системи та мінімізувати ризики помилок під час ручного налаштування. Для фахівців з кібербезпеки автоматизовані підходи є важливими також з точки зору контролю змін, стандартизації конфігурацій та зменшення поверхні атаки.

Метою статті є дослідження принципів побудови та реалізації системи автоматизованого розгортання кластерного середовища на власній інфраструктурі з використанням сучасних технологій контейнеризації та оркестрації.

Аналіз останніх досліджень і публікацій. Одним із ключових завдань сучасних ІТ-систем є ефективне та гнучке розгортання програмного забезпечення. Традиційно для цього використовувалася апаратна або програмна віртуалізація, яка дозволяла запускати кілька віртуальних машин на одному фізичному сервері. У такій моделі кожна віртуальна машина функціонує як окреме середовище, оскільки, як зазначають Н. Sturley та співавтори, віртуальні машини емулюють низькорівневі компоненти, що дозволяє детально налаштувати їх під конкретні потреби та робить їх автономними системами [1]. Водночас така автономність вимагає значних обчислювальних ресурсів і збільшує час запуску середовищ виконання.

Із ускладненням програмних систем і зростанням вимог до швидкості їх розгортання виникла потреба в більш легких підходах до ізоляції застосунків. Таким підходом стала контейнеризація, яка принципово відрізняється від класичної віртуалізації. У порівняльному аналізі віртуалізації та контейнеризації підкреслюється, що контейнери використовують спільну операційну систему хоста і не потребують віртуалізації низькорівневих апаратних компонентів [1]. Ця архітектурна особливість зумовлює менші витрати ресурсів і значно швидший запуск контейнерів.

Контейнеризація визначається як технологія упакування програмного забезпечення разом із усіма компонентами, необхідними для його коректної роботи. "Контейнеризація – це метод упакування та запуску програмного забезпечення в ізольованому середовищі, яке включає в себе все необхідне для його коректної роботи: код, що виконується, бібліотеки, конфігураційні файли, залежності тощо" [2]. Тобто контейнер можна розглядати як зручний формат доставки програмного коду, що забезпечує відтворюваність роботи застосунків у різних середовищах.

Завдяки цьому розроблений і протестований застосунок можна без істотних змін переносити з локального середовища розробника на тестовий сервер або у виробниче середовище. У науковій літературі підкреслюється, що контейнери розглядаються як стандартизована та портативна одиниця розгортання застосунків, що безпосередньо зумовлює високу портативність контейнеризованих рішень [3].

Контейнеризація має кілька важливих переваг. До них належать портативність, легка масштабованість, ефективне використання ресурсів, швидке розгортання та підтримка процесів Continuous Integration/Continuous Delivery (CI/CD), ізоляція та безпека, а також гнучкість у керуванні мікросервісами [2].

Портативність дає змогу використовувати той самий контейнер у різних середовищах без окремого налаштування залежностей.

Легка масштабованість забезпечується можливістю швидкого запуску або зупинки контейнерів залежно від поточного навантаження, що є особливо важливим для середовищ із динамічними вимогами до ресурсів.

Ефективне використання ресурсів досягається за рахунок меншого споживання обчислювальних потужностей порівняно з віртуальними машинами та дозволяє розміщувати більше застосунків на одному сервері.

Швидке розгортання та підтримка процесів CI/CD пов'язані з можливістю автоматизувати процеси збірки, тестування та впровадження програмного забезпечення.

Ізоляція та безпека реалізуються через виконання кожного контейнера в окремому середовищі, що обмежує вплив збоїв або вразливостей одного застосунку на інші компоненти системи.

Гнучкість у керуванні мікросервісами проявляється в можливості незалежного розгортання, оновлення та масштабування окремих сервісів, що спрощує супровід складних програмних систем і підвищує їхню стійкість.

Водночас використання контейнеризації супроводжується і низкою ризиків, серед яких особливої уваги заслуговують безпека контейнерів, моніторинг і логування, оркестрація та управління [2]. Ризики безпеки контейнерів пов'язані зі спільним використанням контейнерами ядра операційної системи хоста, що у разі неналежної ізоляції процесів або контролю доступу може призводити до ескалації привілеїв, виконання шкідливого коду чи витоку даних.

У сфері моніторингу та логування ризики зумовлені динамічним характером контейнеризованих середовищ і коротким життєвим циклом контейнерів, що ускладнює застосування традиційних підходів до спостереження за станом системи та аналізу подій.

Ризики оркестрації та управління виникають зі зростанням кількості контейнерів і потребою в автоматизованому управлінні їх запуском, масштабуванням і відновленням, що ускладнює початкові етапи впровадження контейнерних рішень на власній інфраструктурі.

На практиці контейнеризацію використовують через спеціалізовані платформи для створення, запуску та управління контейнерами. Проте зі зростанням кількості контейнерів виникає потреба в централізованому управлінні їх життєвим циклом, що зумовлює використання систем оркестрації, зокрема Kubernetes – це відкрита платформа для автоматизації розгортання, масштабування та керування контейнеризованими застосунками [2]. Такі системи забезпечують автоматизацію розгортання, масштабування та відновлення контейнеризованих сервісів.

Таким чином, контейнеризація є важливим елементом сучасних ІТ-систем, який забезпечує гнучкість, масштабованість і ефективність розгортання програмного забезпечення. Ці властивості роблять контейнерні технології теоретичною основою для побудови кластерних середовищ і автоматизованих систем розгортання.

Виклад основного матеріалу. Кластер – це група незалежних обчислювальних вузлів, з'єднаних мережею та скоординованих таким чином, що для користувача або прикладного програмного забезпечення вони функціонують як єдина система. Кожен вузол кластера має власні обчислювальні ресурси, однак завдяки програмним засобам управління ресурси всіх вузлів використовуються узгоджено, а сам кластер сприймається як один логічний обчислювальний ресурс.

Однією з головних характеристик кластерів є масштабованість, яка полягає у можливості збільшення або зменшення обчислювальних ресурсів шляхом додавання або видалення вузлів без суттєвого впливу на роботу системи. Такий підхід є особливо важливим для систем зі змінним навантаженням, у яких потреба в ресурсах може істотно коливатися залежно від кількості користувачів або характеру виконуваних завдань [4].

Ще однією важливою характеристикою є відмовостійкість. У кластерному середовищі відмова одного або кількох вузлів не призводить до повної зупинки системи, оскільки виконання завдань може бути автоматично перенесене на інші доступні вузли. Це забезпечує безперервність надання сервісів і мінімізувати час простою, що є визначальною ознакою

кластерів високої доступності [5].

Також важливим є балансування навантаження, яке полягає в розподілі запитів або обчислювальних задач між вузлами кластера таким чином, щоб уникнути перевантаження окремих компонентів. Такий механізм сприяє підвищенню загальної продуктивності системи та забезпечує більш ефективне використання наявних обчислювальних ресурсів.

Залежно від призначення виділяють чотири основні типи кластерів: кластери високої доступності (High-availability clusters), балансувальні кластери (Load balancing clusters), кластери зберігання даних (Storage clusters) та обчислювальні кластери (High-performance computing, HPC) [4, 5]. Кластери високої доступності та балансування забезпечують безперервність сервісів та стійкість до відмов. Кластери зберігання гарантують цілісність інформації шляхом її реплікації. HPC-системи орієнтовані на паралельне виконання ресурсномістких задач, зокрема наукових розрахунків і обробки великих масивів даних. Поєднання механізмів високої доступності, балансування та надійного зберігання даних створює технічну основу для побудови захищених систем автоматизованого розгортання на власній інфраструктурі.

Ефективне функціонування кластерних середовищ значною мірою залежить від мережевої інфраструктури. Обмін даними між вузлами має здійснюватися з мінімальними затримками, оскільки це безпосередньо впливає на продуктивність і стабільність роботи системи. У розподілених обчислювальних середовищах мережеві характеристики є одним із ключових факторів, що визначають ефективність взаємодії між компонентами системи [4].

Окремої уваги потребує безпека кластерних та контейнерних середовищ. Спільне використання обчислювальних ресурсів відкриває нові шляхи до вразливостей, що зумовлює необхідність впровадження чітких політик безпеки. Зокрема, важливим є забезпечення суворого контролю доступу до мережі, обмеження привілеїв компонентів та регулярне сканування на вразливості. Крім того, належний контроль доступу на рівні процесів є обов'язковою умовою для запобігання ескалації привілеїв та захисту від витоку даних [2].

У сучасних IT-системах кластерні обчислювальні середовища тісно поєднуються з контейнерними технологіями та системами оркестрації, що дозволяє автоматизувати управління ресурсами, масштабування та відновлення сервісів. Однією з найбільш поширених платформ для управління контейнерними кластерами є Kubernetes. Її широке використання зумовлене здатністю забезпечувати уніфіковану модель управління кластером, автоматизацію життєвого циклу контейнерів і механізми самовідновлення сервісів, що робить Kubernetes базовим інструментом для побудови сучасних кластерних середовищ [6].

Отже, кластерні обчислювальні середовища створюють технічну основу для побудови масштабованих і відмовостійких сервісів. Їх поєднання з контейнеризацією та автоматизованими засобами оркестрації формує технологічне підґрунтя для створення сучасних IT-інфраструктур і використання Kubernetes як ключового інструменту управління кластерними середовищами.

Для систем автоматизованого розгортання кластерних середовищ контейнерна модель є пріоритетною, оскільки вона забезпечує швидке масштабування, ефективне використання ресурсів і тісну інтеграцію з системами оркестрації, зокрема Kubernetes. Ці властивості зумовлюють її широке застосування при побудові сучасних кластерних систем.

Kubernetes – це система оркестрації контейнерів, яка використовується для автоматизації розгортання, керування, масштабування та підтримки контейнеризованих застосунків [6]. Його поява стала відповіддю на потребу ефективно управляти великою кількістю контейнерів у складних інформаційних системах. Якщо на початковому етапі контейнери могли запускатися окремо і адмініструватися вручну, то зі зростанням кількості сервісів та навантаження виникла потреба в платформі, яка б координувала їхню роботу автоматично. Таку функцію виконує Kubernetes.

Основою його роботи є кластерна архітектура. Kubernetes-кластер складається з набору вузлів, які об'єднані в єдину систему. У межах цієї системи одні вузли відповідають за керування кластером, а інші – за виконання прикладних завдань. Поділ функцій між такими

вузлами забезпечує чітку організацію роботи кластера та спрощує адміністрування всієї системи. Такий підхід дає можливість розподіляти навантаження, ефективно використовувати ресурси та забезпечувати безперервність роботи сервісів навіть у разі збоїв окремих елементів інфраструктури.

У загальному вигляді кластер Kubernetes має дві основні частини: керуючу частину (control plane) і робочі вузли (worker nodes). Відповідно до офіційної документації, керуюча частина відповідає за керування кластером і прийняття глобальних рішень щодо його стану (наприклад, планування), тоді як робочі вузли відповідають за безпосереднє виконання робочих навантажень [7].

Для того щоб кластер працював як єдина система, кожна з його частин має набір специфічних компонентів. Головним елементом керуючої частини є kube-apiserver – "головний диспетчер" кластера Kubernetes. Це основна точка взаємодії з кластером. Через API-сервер користувач надсилає команди, змінює конфігурацію, отримує інформацію про стан системи та керує ресурсами. Інші компоненти Kubernetes також взаємодіють між собою через нього. Тому API-сервер можна вважати центральним елементом, через який проходять усі основні процеси управління.

Не менш важливою складовою є etcd. Це спеціальне сховище, у якому зберігається інформація про стан кластера, його конфігурація та створені об'єкти. Наприклад, саме тут містяться дані про те, які pod створені, які налаштування вони мають і де саме працюють. Фактично etcd є основним сховищем даних, на яке спирається весь Kubernetes [7]. Тому його стабільна робота має дуже велике значення для всього кластера.

Наступним ключовим елементом системи є kube-scheduler. Його завдання полягає в тому, щоб визначити, на якому робочому вузлі потрібно розмістити новий pod. Під час цього вибору враховується наявність вільних ресурсів, завантаженість вузлів, обмеження за політиками розміщення та інші параметри. Завдяки роботі kube-scheduler навантаження в кластері розподіляється більш рівномірно, а ресурси використовуються ефективніше.

Важливе місце в архітектурі Kubernetes займає поняття pod. Згідно з офіційною технічною документацією, pod визначається як найменша обчислювальна одиниця, яку можна створити та якою можна керувати в Kubernetes. Він являє собою групу з одного або кількох контейнерів зі спільним сховищем та мережевими ресурсами [8]. Контейнери всередині одного pod використовують спільний мережовий простір і можуть спільно використовувати деякі ресурси. Це зручно, коли кілька процесів повинні працювати разом у межах одного сервісу.

Крім pod, у Kubernetes існує велика кількість об'єктів, за допомогою яких описується робота застосунків. Одним із найпоширеніших є Deployment. Він використовується для керування розгортанням застосунків, оновленням їхніх версій і підтриманням потрібної кількості копій [9]. Іншим важливим об'єктом є Service, який забезпечує постійний доступ до групи pod, навіть якщо їхній склад або внутрішні адреси змінюються [10].

Важливу роль відіграє і kube-controller-manager. Він відповідає за роботу контролерів, які стежать за тим, щоб фактичний стан системи відповідав бажаному. Наприклад, якщо в конфігурації вказано, що має працювати три pod, а один із них зупинився, контролер помічає це і ініціює створення нового. Цей механізм дозволяє Kubernetes автоматично реагувати на збої та підтримувати працездатність сервісів [7].

На робочих вузлах також функціонують власні системні служби. Передусім це kubelet – агент, який запускається на кожному вузлі. Він отримує інструкції від керуючої частини кластера, контролює запуск контейнерів, перевіряє їхній стан та передає інформацію назад у систему. Іншими словами, kubelet забезпечує зв'язок між робочим вузлом і Kubernetes як системою управління.

Ще однією системною службою робочого вузла є kube-proxy. Він відповідає за мережеву взаємодію та маршрутизацію трафіку в межах кластера. Завдяки його роботі запити можуть надходити до потрібних pod, а сервіси в Kubernetes отримують стабільну точку доступу. Це особливо важливо тому, що окремі pod можуть перезапуститися, переміщуватися

між вузлами або створюватися заново, але користувачі та інші сервіси повинні звертатися до них без змін у своїй конфігурації.

Слід також згадати контейнерне середовище виконання. Воно відповідає за безпосередній запуск контейнерів на вузлі. Цей компонент завантажує образи контейнерів, створює їх, запускає, зупиняє та видаляє за потреби. Тобто Kubernetes координує процес, але сам запуск контейнерів виконується через відповідне середовище виконання. Крім того, для забезпечення єдиного адресного простору та зв'язку між усіма контейнерами кластер вимагає встановлення мережевого плагіна стандарту CNI (Container Network Interface), який організовує віртуальну оверлейну мережу [11].

Також у Kubernetes використовуються ConfigMap і Secret. Вони потрібні для зберігання конфігураційних параметрів та службових даних. Різниця між ними полягає в тому, що Secret застосовується для більш чутливих даних, наприклад паролів або токенів доступу [10]. Для окремих сценаріїв використовуються також StatefulSet, DaemonSet, Job та інші об'єкти. Наприклад, StatefulSet підходить для сервісів, яким важлива сталість ідентифікації або порядок запуску, а DaemonSet використовується тоді, коли певний род повинен працювати на кожному вузлі кластера [12]. Таким чином, Kubernetes надає не лише базові механізми запуску контейнерів, а й досить широкий набір інструментів для різних типів навантаження.

Одним із головних принципів роботи Kubernetes, який об'єднує ці компоненти та об'єкти, є декларативний підхід. Його суть полягає в тому, що користувач або адміністратор не описує покроково, що саме система повинна зробити, а задає бажаний результат. Наприклад, можна вказати, що в кластері має працювати три екземпляри певного застосунку. Після цього Kubernetes постійно перевіряє, чи зберігається цей стан на практиці. Якщо виникає відхилення, система намагається його усунути автоматично. Завдяки цьому забезпечується стабільність роботи кластера [6].

Кластерна архітектура Kubernetes добре підходить для сучасних розподілених систем, особливо тих, що побудовані за мікросервісним принципом. У такій моделі окремі частини застосунку працюють як самостійні сервіси, але водночас повинні взаємодіяти між собою. Kubernetes спрощує управління такими сервісами, оскільки дозволяє централізовано керувати їхнім запуском, оновленням, масштабуванням і відновленням після збоїв. Важливою перевагою Kubernetes є те, що він не прив'язаний до конкретної фізичної інфраструктури. Його можна використовувати на локальних серверах, у корпоративних дата-центрах, у публічних хмарах або в змішаних середовищах. Для користувача це означає, що підхід до роботи із застосунками залишається однаковим незалежно від того, де саме вони розгортаються. Це значно спрощує перенесення сервісів між різними середовищами та робить інфраструктуру гнучкішою [11].

В підсумку, архітектура Kubernetes-кластера є цілісною системою, у якій кожен компонент виконує свою функцію. Її елементи забезпечують зберігання даних про стан кластера, прийняття рішень щодо розміщення навантаження, запуск контейнерів, мережеву взаємодію та контроль за працездатністю системи. Завдяки такій організації Kubernetes забезпечує автоматизацію керування кластером, зручність адміністрування та стійкість до збоїв.

Розгортання інформаційних систем на власній інфраструктурі зазвичай пов'язане з підвищеними вимогами до захисту даних, тому питання безпеки Kubernetes-кластера потребує додаткової уваги. З погляду кібербезпеки, просто запустити контейнери недостатньо – потрібно гарантувати, що в разі зламу одного з компонентів зломисник не зможе отримати контроль над усією системою. Відповідно до рекомендацій стандарту NIST щодо безпеки контейнерів, архітектура захисту має будуватися на принципі найменших привілеїв та ізоляції ресурсів [13].

У Kubernetes цей захист реалізується на кількох рівнях, основні з яких: контроль доступу, мережева ізоляція та безпечне управління чутливими даними.

Першим рівнем захисту є система контролю доступу на основі ролей – RBAC (Role-Based Access Control). У Kubernetes за замовчуванням жоден користувач або процес не має

повних прав на управління кластером. Для того, щоб виконати будь-яку дію (наприклад, створити новий pod або переглянути логи), потрібно мати відповідний дозвіл. RBAC дозволяє адміністратору створювати ролі з чітко обмеженими правами і призначати їх конкретним користувачам або самим застосункам (через ServiceAccounts) [14]. Завдяки цьому, якщо зломисник зламає вебдодаток, він отримає лише ті мінімальні права, які були виділені цьому додатку, і не зможе видалити бази даних чи змінити налаштування кластера.

Другим важливим елементом безпеки для розподілених систем є мережева ізоляція. Особливість Kubernetes полягає в тому, що одразу після встановлення всі pod в кластері можуть вільно обмінюватися даними між собою без жодних обмежень. З точки зору безпеки це створює ризик: якщо хакер знайде вразливість у публічному вебсайті (frontend), він зможе напряму підключитися до внутрішньої бази даних, яка мала б бути прихованою. Для вирішення цієї проблеми використовуються Network Policies (мережеві політики). Це своєрідний внутрішній фаєрвол кластера. За допомогою Network Policies можна прописати правила маршрутизації: наприклад, дозволити базі даних приймати запити виключно від бекенду і блокувати інший трафік. Це підвищує мережеву безпеку інфраструктури, унеможливаючи прямий доступ до критичних даних або зовнішніх серверів навіть у разі зламу суміжних контейнерів [15]. На рис. 1 показано базову модель мережевої взаємодії pod у кластері Kubernetes, у межах якої Network Policies застосовуються для контролю дозволеного трафіку.

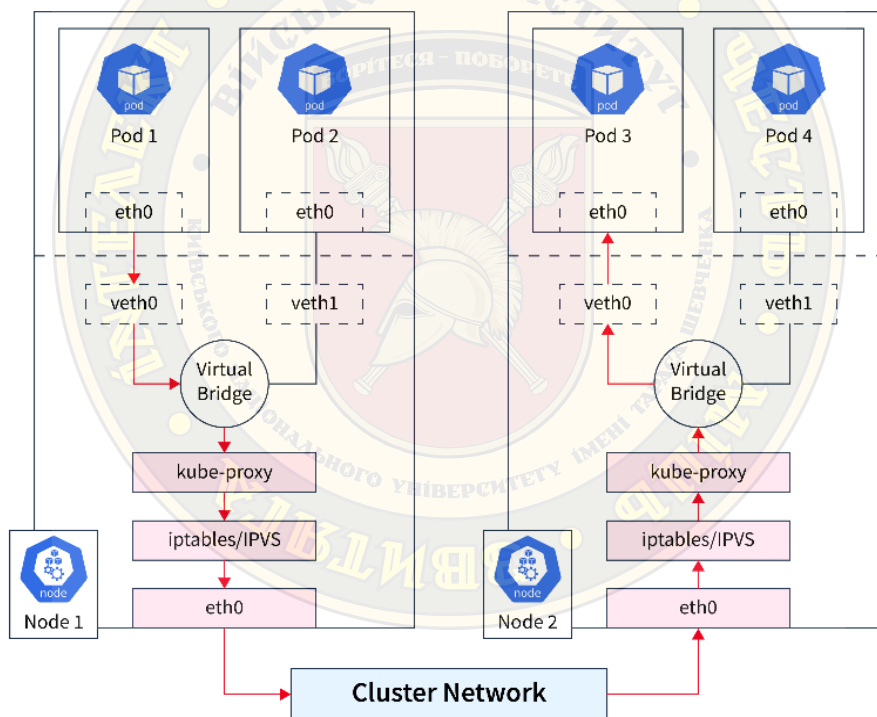


Рисунок 1 – Базова модель мережевої взаємодії pod у Kubernetes-кластері [24]

Третім важливим аспектом є робота з конфіденційною інформацією. Будь-який сучасний застосунок використовує паролі, API-ключі або сертифікати доступу. Зберігати такі дані прямо в коді – це небезпечна практика, оскільки доступ до них може отримати будь-хто, хто має доступ до вихідного коду [13]. Для вирішення цієї задачі в Kubernetes передбачено спеціальний об'єкт – Secret. Цей механізм дозволяє зберігати паролі окремо від самих застосунків у зашифрованому вигляді на рівні системи управління. Контейнер отримує доступ до цих даних лише в момент запуску і лише якщо йому це явно дозволено правилами [10].

Практична реалізація розгортання кластера на власній інфраструктурі. Перед розгортанням Kubernetes-кластера необхідно визначити вимоги до інфраструктури, на якій він

буде працювати. Від правильного підбору ресурсів залежить не лише стабільність роботи кластера, а й можливість його подальшого масштабування та оновлення. Якщо інфраструктура спроектована без урахування реальних потреб системи, у процесі експлуатації можуть виникати проблеми з продуктивністю, доступністю сервісів та керуванням вузлами.

Під власною інфраструктурою (on-premises) розуміється середовище, яке розгортається і контролюється самостійно, без використання готових керованих Kubernetes-сервісів від хмарних провайдерів (наприклад, AWS EKS чи Google GKE). На відміну від використання таких сервісів, розгортання на власному обладнанні дає повний контроль над апаратною частиною та безпекою даних, але водночас вимагає самостійного налаштування основних компонентів та забезпечення їхньої взаємодії [11].

Однією з базових вимог є достатній обсяг обчислювальних ресурсів. Для коректної роботи кластера потрібно передбачити ресурси як для керуючої частини, так і для робочих вузлів [7]. Виходячи з цих вимог, апаратною базою для практичної частини розгортання кластерного середовища став фізичний міні-кластер, побудований на п'яти одноплатних мікрокомп'ютерах Raspberry Pi 5 (з 8 ГБ оперативної пам'яті кожен). Вибір цієї платформи зумовлений високою енергоефективністю, достатнім обсягом RAM для запуску контейнерів та підтримкою архітектури ARM64 з боку екосистеми Kubernetes. Архітектурну схему побудованого Kubernetes-кластера наведено на рис. 2.

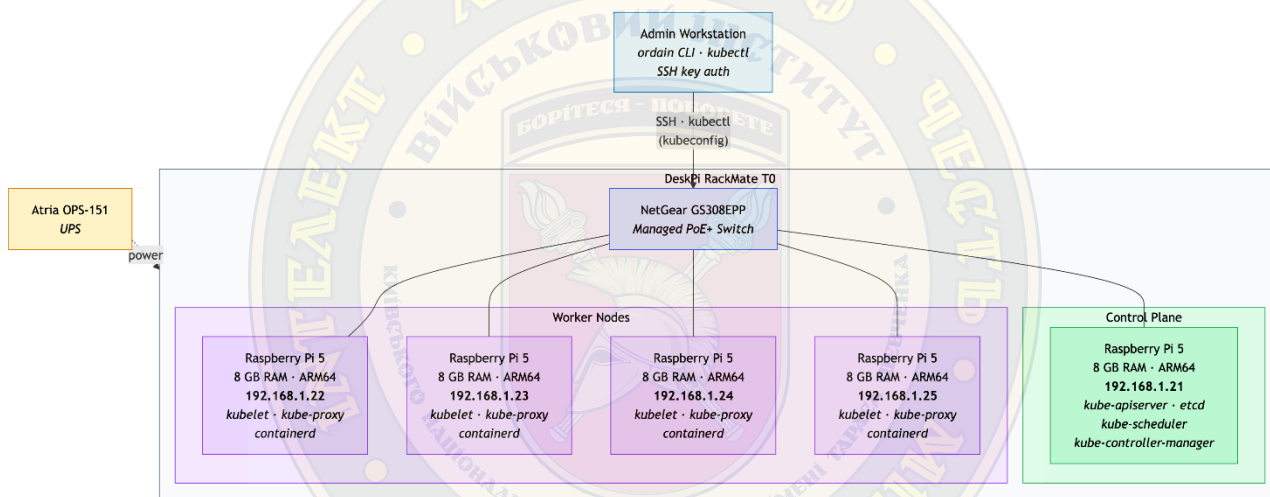


Рисунок 2 – Архітектурна схема запропонованого Kubernetes-кластера

Для організації простору та забезпечення належного охолодження усі п'ять вузлів фізично розміщені у спеціалізованій стійці DeskPi RackMate T0. Апаратна інфраструктура Kubernetes-кластера складається з наступних елементів:

Склад інфраструктури:

- Raspberry Pi 5, 8 GB RAM - 5 од.;
- DeskPi RackMate T0 - 1 од.;
- NetGear GS308EPP - 1 од.;
- Atria OPS-151 - 1 од.

Функціональний розподіл вузлів:

- control plane node - 1;
- worker nodes - 4.

Фізичне підключення:

- усі вузли підключені до комутатора NetGear GS308EPP;
- тип підключення – дротове Ethernet;
- адміністрування вузлів – через SSH.

Живлення:

- вузли кластера та комутатор підключені до ДБЖ Atria OPS-151.

Мережева взаємодія реалізована наступним чином. Усі елементи Kubernetes-кластера повинні мати стабільне з'єднання з мінімальними затримками для обміну службовими повідомленнями та роботи API-сервера [16]. Для реалізації цієї вимоги в інфраструктуру інтегровано керований комутатор (світч) NetGear GS308EPP. Він забезпечує надійну маршрутизацію внутрішнього трафіку між вузлами кластера та ізоляцію мережевих сегментів, що є важливим з точки зору безпеки.

Під час проєктування власної інфраструктури слід приділяти увагу відмовостійкості та безперебійному живленню. Раптове знеструмлення серверів може призвести до пошкодження системних файлів бази даних etcd, у якій зберігається поточний стан усього кластера [7]. Для вирішення цієї проблеми до апаратного комплексу підключено джерело безперебійного живлення (ДБЖ) Atria OPS-151. Це гарантує базову апаратну відмовостійкість: у разі перебоїв в електромережі кластер продовжить функціонувати, даючи час на коректне збереження даних.

З точки зору кібербезпеки, доступ до керуючих інтерфейсів і вузлів має бути суворо обмеженим [13]. У зібраному кластері це досягається налаштуванням локальної мережі, доступом виключно через протокол SSH з використанням криптографічних ключів та відсутністю прямого доступу вузлів до публічної мережі Інтернет без необхідності.

Спроектвана та зібрана апаратна інфраструктура загалом відповідає основним вимогам до ресурсів, мережевої ізоляції та енергонезалежності, створюючи умови для автоматизованого розгортання Kubernetes.

Після підготовки фізичної інфраструктури наступним етапом є вибір інструментів для автоматизації налаштування операційних систем та встановлення Kubernetes. Це важливий етап, оскільки від нього залежать складність налаштування, швидкість розгортання, можливість повторного використання конфігурацій та зручність подальшого адміністрування системи. Автоматизація дозволяє мінімізувати кількість ручних дій, усунути ризик помилок та забезпечити відтворюваність конфігурацій на всіх вузлах [17].

Для автоматизації розгортання Kubernetes зазвичай використовують готові системи управління конфігураціями (наприклад, Ansible або Terraform) у поєднанні зі спеціалізованими утилітами, такими як kubectl. Ansible є популярним вибором завдяки безагентній архітектурі та роботі через стандартний протокол SSH. Проте використання готових рішень часто приховує від адміністратора сам процес налаштування вузлів і взаємодії компонентів.

Зважаючи на це, у межах дослідження було прийнято рішення не використовувати Ansible, а розробити власний інструмент автоматизації мовою Rust.

Вибір мови Rust обґрунтований кількома факторами, які є важливими для кібербезпеки та системного адміністрування. По-перше, Rust гарантує безпечну роботу з пам'яттю (memory safety), що мінімізує ризик виникнення вразливостей, притаманних системним утилітам на C/C++. По-друге, ця мова дозволяє компілювати ефективні бінарні файли, які не вимагають встановлення важких залежностей (наприклад, Python-середовища) на цільових мікрокомп'ютерах Raspberry Pi.

Конфігурація запуску розробленого інструменту може задаватися двома способами: через аргументи командного рядка або через єдиний опціональний файл конфігурації у форматі YAML чи JSON. Такий підхід дає змогу або передавати параметри безпосередньо під час запуску, або зберігати типову конфігурацію для повторного використання. Якщо один і той самий параметр задано в обох формах, тоді пріоритет має значення, яке передане через CLI.

Розроблений інструмент призначений для виконання повного циклу підготовки вузлів. Його ключові можливості включають:

Розгортання за IP-діапазонами: інструмент дозволяє задавати діапазони IP-адрес під час запуску (наприклад, 192.168.1.22-192.168.1.25), автоматично розгортаючи їх у перелік окремих адрес та підключаючись до кожного вузла кластера.

Підключення та конфігурація вузлів: програма підключається до вузлів через протокол SSH та автоматично виконує необхідні системні налаштування (відключення swap, конфігурація ядра, встановлення залежностей). За замовчуванням вузли обробляються послідовно, однак за наявності відповідного параметра конфігурації підготовка може виконуватися паралельно для пришвидшення процесу.

Оркестрація через kubernetes: інструмент самостійно ініціює команду kubernetes init на керуючому вузлі, перехоплює згенерований криптографічний токен та розсилає команду kubernetes join на всі робочі вузли.

Встановлення мережевого плагіна та додатків кластера: після ініціалізації інструмент автоматично розгортає обраний CNI-плагін (Flannel або Calico), а також, залежно від конфігурації, встановлює балансувальник навантаження MetalLB, контролер вхідного трафіку ingress-nginx, постачальник сховищ (local-path або Longhorn) та metrics-server.

Режим валідації (dry-run): інструмент підтримує режим пробного запуску, який перевіряє з'єднання з вузлами та відповідність апаратних вимог без внесення будь-яких змін до системи.

Скидання кластера (reset): програма забезпечує повне або часткове скидання кластера. У частковому режимі видаляється лише стан Kubernetes, тоді як повне скидання додатково деінстальовує всі пакети та повертає системну конфігурацію вузлів до початкового стану.

Для зручності адміністрування розроблений інструмент забезпечує запуск процесу розгортання через командний рядок і надає службову інформацію про хід виконання операцій. Це дозволяє в режимі реального часу відстежувати статус підключення до кожного вузла та загальний прогрес розгортання.

Отже, розробка власної утиліти на Rust замість використання стандартних засобів дозволила створити легковаговий, безпечний та адаптований під конкретну ARM-інфраструктуру інструмент. Це підтвердило можливість використання Rust для автоматизації розгортання в межах запропонованої інфраструктури.

Після визначення вимог до інфраструктури та вибору інструментів наступним етапом є безпосередня реалізація автоматизованого розгортання Kubernetes-кластера. На цьому етапі теоретичні положення реалізуються на практиці: підготовка вузлів, вибір системних налаштувань, ініціалізація кластера і забезпечення автоматичного приєднання до нього робочих вузлів.

Реалізація автоматизованого розгортання починається з фази попередньої перевірки (preflight). На цьому етапі інструмент підключається до кожного вузла через SSH, перевіряє відповідність процесорної архітектури заданому параметру (наприклад, ARM64), а також визначає стан файлу підкачки (swap). Ця фаза не вносить жодних змін до системи і дозволяє виявити проблеми з підключенням або невідповідність апаратних вимог ще до початку розгортання.

Після успішного проходження preflight починається базова підготовка серверів. На кожному вузлі має бути встановлена підтримувана операційна система (Linux архітектури ARM64) та налаштований мережевий доступ для віддаленого управління. Також необхідно виконати низку попередніх дій на рівні ядра ОС: налаштувати мережеву маршрутизацію (sysctl), повністю вимкнути файл підкачки (swap) та встановити контейнерне середовище виконання. Без виконання цих системних вимог ключові компоненти Kubernetes не зможуть працювати коректно [7].

На практиці ці підготовчі дії було автоматизовано за допомогою розробленого власного інструмента мовою Rust. Програма приймає задані діапазони IP-адрес під час запуску, автоматично розгортає їх у перелік окремих адрес та підключається до кожного вузла через протокол SSH для виконання налаштувань. За наявності відповідного прапорця конфігурації підготовка вузлів виконується паралельно. Інструмент підтримує запуск через інтерфейс командного рядка (CLI), що дозволяє адміністратору наочно відстежувати процес налаштування кожного комп'ютера. Перевага такого підходу полягає в тому, що всі дії є формалізованими і виконуються без ручного втручання.

Після підготовки операційної системи розроблений інструмент ініціює створення керуючої частини за допомогою офіційної утиліти `kubeadm` [18]. На цьому етапі автоматично виконуються службові дії, необхідні для початкової ініціалізації кластера, зокрема генеруються криптографічні сертифікати, формуються параметри конфігурації та розгортаються базові компоненти керування, зокрема API-сервер і `etcd`. Після успішного завершення ініціалізації скрипт автоматично отримує згенеровану команду приєднання та токен, що надалі використовуються для підключення робочих вузлів до кластера.

Наступним кроком є приєднання робочих вузлів. Розроблений інструмент автоматично передає згенерований токен на інші чотири комп'ютери Raspberry Pi та виконує на них команду `kubeadm join`. Завдяки цьому процес розгортання стає надзвичайно швидким і таким, що не потребує додаткових ручних дій. Особливо це важливо під час розробки та тестування, коли фізичний кластер може потребувати періодичного повного перерозгортання з нуля.

Після приєднання всіх робочих вузлів інструмент автоматично отримує файл `kubeconfig` з керуючого вузла та зберігає його за вказаним шляхом на локальній машині адміністратора. Якщо в конфігурації задано назву кластера (`cluster_name`), вона автоматично підставляється у вміст `kubeconfig`-файлу. Це дозволяє одразу після завершення розгортання працювати з кластером через `kubectl` без додаткових ручних дій.

Після створення базової структури кластера необхідно налаштувати мережевий плагін стандарту CNI (Container Network Interface), який підтримує архітектуру ARM. Без нього `Pods` не можуть отримувати IP-адреси та повноцінно взаємодіяти між собою, що є ключовою умовою функціонування Kubernetes [13]. Розроблений інструмент підтримує два CNI-плагіни: `Flannel` та `Calico`. Вибір конкретного плагіна задається через параметр конфігурації, а його встановлення виконується автоматично в процесі розгортання кластера.

Після встановлення CNI-плагіна інструмент переходить до фази розгортання додаткових компонентів кластера (`post-install`). Залежно від обраної конфігурації на цьому етапі автоматично встановлюються: балансувальник навантаження `MetallB` з виділеним діапазоном IP-адрес для сервісів типу `LoadBalancer`, контролер вхідного трафіку `ingress-nginx`, постачальник класу сховищ (`local-path` для полегшеного локального зберігання або `Longhorn` для реплікованих блочних томів) та `metrics-server` для збору даних про використання ресурсів. Кожен із цих додатків є опціональним і активується відповідним параметром у `preset`-файлі або через CLI.

Після цього етапу інструмент автоматично здійснює фінальну перевірку працездатності кластера. На цьому етапі програма послідовно перевіряє готовність усіх вузлів (статус `Ready`), коректність запуску `CoreDNS` та `kube-proxy`, а також сканує кластер на наявність `Pods` у помилкових станах. Якщо система розгорнута правильно, усі компоненти керуючої частини і мережі працюють без помилок, і адміністратор отримує підсумковий звіт про стан кластера.

Варто зазначити, що використання власного програмного інструмента дозволяє не обмежуватися лише первинним створенням кластера. Завдяки підтримці IP-діапазонів архітектура інструмента враховує можливість масштабування кластера в майбутньому (додавання нових вузлів). Крім того, інструмент реалізує режим скидання кластера (`reset`), який дозволяє повністю або частково відновити вузли до початкового стану, що є особливо важливим під час розробки та тестування, коли фізичний кластер може потребувати періодичного повного перерозгортання з нуля.

Таким чином, реалізація автоматизованого розгортання включала попередню перевірку вузлів (`preflight`), підготовку апаратної бази, виконання системних налаштувань за допомогою `Rust`-інструмента, встановлення контейнерного середовища, ініціалізацію керуючого вузла, приєднання робочих вузлів і налаштування мережевого плагіна. Після цього було виконано фінальну перевірку працездатності кластера.

Висновки. У результаті дослідження принципів побудови та реалізації системи автоматизованого розгортання кластерного середовища була запропонована реалізація системи автоматизованого розгортання кластерного середовища Kubernetes на власній інфраструктурі і отримані наступні результати.

Досліджено архітектуру та механізми системи оркестрації Kubernetes. З'ясовано, що її декларативний підхід та розділення на керуючу частину та робочі вузли дозволяють автоматизувати управління тисячами контейнерів. Вбудовані механізми самовідновлення, горизонтального масштабування (HPA) та сувора політика мережевої безпеки (Network Policies, RBAC) роблять Kubernetes еталонним рішенням для мікросервісної архітектури.

Визначено та реалізовано вимоги до власної (on-premises) інфраструктури. Замість хмарних сервісів було спроектовано та зібрано фізичний міні-кластер на базі п'яти мікрокомп'ютерів Raspberry Pi 5 (по 8 ГБ оперативної пам'яті). Інфраструктуру розміщено у спеціалізованій стійці DeskPi RackMate T0, ізолювано через керований комутатор NetGear та забезпечено апаратною відмовостійкістю за допомогою джерела безперебійного живлення Atria.

Обґрунтовано вибір інструментів автоматизації. Замість використання сторонніх систем управління конфігураціями (таких як Ansible), з метою оптимізації під ARM-архітектуру та підвищення рівня безпеки (memory safety), було розроблено власний програмний інструмент мовою Rust для взаємодії з офіційною утилітою kubectl. Цей Rust-інструмент забезпечив паралельне підключення до вузлів через протокол SSH, конфігурацію операційних систем (вимкнення swappiness, налаштування ядра), встановлення контейнерного середовища, ініціалізацію керуючого вузла, автоматичне приєднання чотирьох робочих вузлів із застосуванням перехоплених токенів безпеки, розгортання CNI-плагіна (Flannel або Calico) та додаткових компонентів кластера (MetalLB, ingress-nginx, постачальник сховищ, metrics-server). Також реалізовано режим скидання кластера.

Результати практичних тестів підтвердили, що всі вузли перейшли в робочий статус, розгорнутий CNI-плагін забезпечив внутрішню мережеву маршрутизацію, а автоматизована тестова система успішно перевірила розгортання кластера в численних конфігураціях (обидва CNI-плагіни, різні комбінації додатків, паралельний та послідовний режими, користувацькі CIDR-діапазони), а також коректність часткового та повного скидання кластера.

Розроблений підхід до розгортання підтвердив практичну придатність для побудови локального Kubernetes-кластера в експериментальних умовах.

ЛІТЕРАТУРА:

1. Sturley H., Fournier A., Salcedo-Navarro A., Garcia-Pineda M., Segura-Garcia J. Virtualization vs. Containerization, a Comparative Approach for Application Deployment in the Computing Continuum Focused on the Edge / Future Internet. 2024. Vol. 16, No. 11, Article 427.
2. Придибайло О. Б., Придибайло Р. В. Роль контейнеризації у хмарних обчисленнях / Зв'язок. 2025. № 4. С. 92-98.
3. Pahl C., Jamshidi P., Zimmermann O. Cloud Container Technologies: A State-of-the-Art Review // IEEE Transactions on Cloud Computing. 2019. Vol. 7, No. 3. pp. 677–692.
4. IBM Cloud Education. *Cluster computing* [Електронний ресурс]. – URL: <https://www.ibm.com/topics/cluster-computing>
5. Red Hat. What is high availability? [Електронний ресурс]. – URL: <https://www.redhat.com/en/topics/linux/what-is-high-availability>
6. Kubernetes Documentation. Overview [Електронний ресурс]. – URL: <https://kubernetes.io/docs/concepts/overview/> (дата звернення: 23.01.2026)
7. Kubernetes Documentation. Kubernetes Components [Електронний ресурс]. – URL: <https://kubernetes.io/docs/concepts/overview/components/> (дата звернення: 23.01.2026)
8. Kubernetes Documentation. Pods [Електронний ресурс]. – URL: <https://kubernetes.io/docs/concepts/workloads/pods/>
9. Kubernetes Documentation. Deployments [Електронний ресурс]. – URL: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>
10. Kubernetes Documentation. Service [Електронний ресурс]. – URL: <https://kubernetes.io/docs/concepts/services-networking/service/>
11. Linhuber, M., et al. On-Premise Kubernetes in Software Engineering Education: An Experience Report. TUM, 2026. URL: <https://aet.cit.tum.de/research/publications/linhuber2026icse-seet.pdf>

12. Kubernetes Documentation. Configuration [Електронний ресурс]. – URL: <https://kubernetes.io/docs/concepts/configuration/>
13. Souppaya M., Morello J., Scarfone K. Application Container Security Guide // NIST Special Publication 800-190. 2017. URL: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf>
14. Kubernetes Documentation. Using RBAC Authorization [Електронний ресурс]. – URL: <https://kubernetes.io/docs/reference/access-authn-authz/rbac/>
15. Kubernetes Documentation. Network Policies [Електронний ресурс]. – URL: <https://kubernetes.io/docs/concepts/services-networking/network-policies/> (дата звернення: 23.01.2026)
16. Sharma V. Complete Guide to Kubernetes Networking // Scaler Topics. Last updated: 9 Jan 2024. URL: <https://www.scaler.com/topics/kubernetes/kubernetes-networking/> (дата звернення: 22.01.2026)
17. Hidayat A., Andi S. Automating Server Deployment With Ansible to Improve Performance, Virtualization, and Network Security // ResearchGate. 2025. pp.10618-10626. URL: <https://www.researchgate.net/publication/394960982> (дата звернення: 23.01.2026)
18. Kubernetes Documentation. Installing kubeadm [Електронний ресурс]. – URL: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>

REFERENCES:

1. Sturley H., Fournier A., Salcedo-Navarro A., Garcia-Pineda M., Segura-Garcia J. Virtualization vs. Containerization, a Comparative Approach for Application Deployment in the Computing Continuum Focused on the Edge / Future Internet. 2024. Vol. 16, No. 11, Article 427.
2. Prydybailo O. B., Prydybailo R. V. (2025). Rol konteyneryzatsiyi u khmarnykh obchyslennyakh [Role of containerization in cloud computing]. Zvyazok, (4), pp.92–98.
3. Pahl C., Jamshidi P., Zimmermann O. Cloud Container Technologies: A State-of-the-Art Review // IEEE Transactions on Cloud Computing. 2019. Vol. 7, No. 3. pp. 677–692.
4. IBM Cloud Education. *Cluster computing*. URL: <https://www.ibm.com/topics/cluster-computing>
5. Red Hat. What is high availability? URL: <https://www.redhat.com/en/topics/linux/what-is-high-availability>
6. Kubernetes Documentation. Overview. URL: <https://kubernetes.io/docs/concepts/overview/>
7. Kubernetes Documentation. Kubernetes Components. URL: <https://kubernetes.io/docs/concepts/overview/components/>
8. Kubernetes Documentation. Pods. URL: <https://kubernetes.io/docs/concepts/workloads/pods/>
9. Kubernetes Documentation. Deployments. URL: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>
10. Kubernetes Documentation. Service. URL: <https://kubernetes.io/docs/concepts/services-networking/service/>
11. Linhuber, M., et al. On-Premise Kubernetes in Software Engineering Education: An Experience Report. – TUM, 2026. URL: <https://aet.cit.tum.de/research/publications/linhuber2026icse-seet.pdf>
12. Kubernetes Documentation. Configuration. URL: <https://kubernetes.io/docs/concepts/configuration/>
13. Souppaya M., Morello J., Scarfone K. Application Container Security Guide // NIST Special Publication 800-190. 2017. URL: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf>
14. Kubernetes Documentation. Using RBAC Authorization. URL: <https://kubernetes.io/docs/reference/access-authn-authz/rbac/>
15. Kubernetes Documentation. Network Policies. URL: <https://kubernetes.io/docs/concepts/services-networking/network-policies/>
16. Sharma V. Complete Guide to Kubernetes Networking // Scaler Topics. Last updated: 9 Jan 2024. URL: <https://www.scaler.com/topics/kubernetes/kubernetes-networking/>
17. Hidayat A., Andi S. Automating Server Deployment With Ansible to Improve Performance, Virtualization, and Network Security // ResearchGate. 2025. pp.10618-10626. URL: <https://www.researchgate.net/publication/394960982>
18. Kubernetes Documentation. Installing kubeadm [Електронний ресурс]. – URL: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>

AUTOMATED CLUSTER ENVIRONMENT DEPLOYMENT SYSTEM

The article examines the issues of rapid deployment and maintenance of complex IT infrastructures under conditions of increasing workload or failures of individual components. It is shown that one of the effective approaches to deploying complex infrastructures is the use of container technologies in combination with cluster computing environments and container orchestration systems.

The paper analyzes the concept of containerization. It is demonstrated that this approach provides application isolation together with all required dependencies, ensuring software portability, consistent operation across different environments, and the possibility of automating deployment processes.

The main characteristics of cluster systems are considered. It is determined that combining independent computing nodes into a single logical network provides high availability, scalability, and fault tolerance. The principal characteristics of clusters, including scalability, fault tolerance, and load balancing, are analyzed. In addition, the networking requirements necessary for the operation of cluster environments are described.

It is shown that, for an automated cluster deployment system, the container-based model is the preferred approach because it enables rapid scaling, efficient resource utilization, and seamless integration with container orchestration platforms.

The Kubernetes container orchestration platform, which is used to automate the deployment, management, scaling, and maintenance of containerized applications, is analyzed. Its capability to efficiently manage a large number of containers in complex information systems is demonstrated.

For experimental evaluation, a practical implementation of an automated deployment and management system for containerization technologies in a Kubernetes cluster environment was proposed, described, implemented, and tested on the authors' own infrastructure.

Keywords: cluster computing environments, containerization, orchestration, Kubernetes, deployment automation, software applications.

