Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Аналіз та розробка ефективних методів DevOps для оптимізації процесу розробки програмного забезпечення

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Колиняк В.М.
(підпис) (прізвище та ініціали)

Керівник _____ _____
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2024

АНОТАЦІЯ

Кваліфікаційна робота бакалавра присвячена темі: «Аналіз та розробка ефективних методів DevOps для оптимізації процесу розробки програмного забезпечення». Виконана студентом Колиняком Владиславом Михайловичем, студентом Тернопільського національного технічного університету імені Івана Пулюя, факультету комп'ютерно-інформаційних систем і програмної інженерії, кафедри програмної інженерії, спеціальності «Інженерія програмного забезпечення», групи СП-41.

Відомості про обсяг роботи: сторінок – 73, рисунків – 0, таблиць – 0, частин – 5, додатків – 2, посилань – 12, формул – 0.

Ключові слова: DevOps, автоматизація, IT-компанії, розробка програмного забезпечення, Jenkins, Kubernetes, Terraform, Azure DevOps.

Кваліфікаційна робота присвячена дослідженню та розробці системи автоматизації DevOps процесів для IT-компаній. У першому розділі кваліфікаційної роботи висвітлено теоретичні основи DevOps, включаючи історію та еволюцію, основні принципи та практики, інструменти та технології, а також роль культури та співпраці в DevOps. Аналізовано вплив DevOps на процес розробки програмного забезпечення.

У другому розділі обґрунтовано вибір архітектури та розроблено структуру системи автоматизації DevOps, включаючи інструменти для автоматизації, моніторинг та логування, а також інтеграцію практик безпеки.

У третьому розділі проаналізовано інструкції з розміщення та обслуговування системи в Інтернеті, наповнення контенту, а також популяризацію та підтримку системи.

У четвертому розділі розглянуто моделювання та прогнозування небезпечних ситуацій, описано вимоги ергономіки для організації робочого місця оператора ПК.

ABSTRACT

The bachelor's qualification work is devoted to the topic: "Development of a DevOps process automation system for IT companies." Made by Kolinyak Vladyslav Mykhailovych, a student of Ternopil National Technical University named after Ivan Pulyu, faculty of computer and information systems and software engineering, department of software engineering, specialty "Software engineering", group SP-41.

Information on the volume of work: pages – 73, figures – 0, tables – 0, parts – 5, appendices – 2, references – 12, formulas – 0.

Keywords: DevOps, automation, IT companies, software development, Jenkins, Kubernetes, Terraform, Azure DevOps.

The qualification work is dedicated to the research and development of the DevOps process automation system for IT companies. The first chapter of the thesis covers the theoretical foundations of DevOps, including history and evolution, basic principles and practices, tools and technologies, and the role of culture and collaboration in DevOps. The impact of DevOps on the software development process is analyzed.

The second chapter justifies the choice of architecture and develops the framework of a DevOps automation system, including tools for automation, monitoring and logging, and the integration of security practices.

In the third section, the instructions for placing and maintaining the system on the Internet, content filling, as well as popularization and support of the system are analyzed.

The fourth chapter examines the modeling and forecasting of dangerous situations, describes the requirements of ergonomics for the organization of the PC operator's workplace.

ЗМІСТ

ВСТУП

В умовах сучасного ринку програмного забезпечення важливою є не тільки швидкість та якість розробки, але й здатність оперативно адаптуватися до змін вимог та умов експлуатації. Застосування методів DevOps дозволяє оптимізувати процес розробки, покращити співпрацю між командами та автоматизувати рутинні задачі, що сприяє підвищенню ефективності та зниженню часу виведення продукту на ринок.

Мета дослідження полягає в аналізі та визначенні ефективних методів DevOps для оптимізації процесу розробки програмного забезпечення. Завдання дослідження включають вивчення теоретичних основ DevOps, аналіз існуючих методів оптимізації процесу розробки, розробку та тестування практичних прикладів застосування DevOps, а також оцінку ефективності впровадження цих методів.

Об'єктом дослідження є процес розробки програмного забезпечення в IT-компаніях. Предметом дослідження є методи та інструменти DevOps, що використовуються для оптимізації цього процесу.

Дослідження базується на аналізі наукової літератури, статей та інших джерел, присвячених DevOps та оптимізації процесу розробки програмного забезпечення. Також будуть використані методи експериментального дослідження для розробки та тестування практичних прикладів застосування DevOps.

Дипломна робота складається з вступу, чотирьох основних розділів, висновків, списку використаних джерел та додатків. Перший розділ присвячений теоретичним основам DevOps, другий розділ аналізує існуючі методи оптимізації процесу розробки, третій розділ містить практичне застосування методів DevOps, а четвертий розділ присвячений аналізу результатів та рекомендаціям.

1 ТЕОРЕТИЧНІ ОСНОВИ DEVOPS

1.1 Історія та еволюція DevOps

У цьому розділі ми розглянемо історію та еволюцію концепції DevOps, яка стала важливим елементом в сучасній розробці програмного забезпечення. Ми дослідимо походження терміна "DevOps", основні етапи його розвитку, а також зміни, які він приніс у процеси розробки, тестування та експлуатації ПЗ. Також ми зосередимо увагу на основних подіях та інноваціях, які сприяли розвитку та популяризації практик DevOps.

DevOps (скорочено від "development" і "operations") це набір практик, культура, філософія та набір інструментів, які сприяють співпраці між розробниками програмного забезпечення (Dev) та IT-фахівцями, які займаються експлуатацією та підтримкою (Ops), з метою покращення та прискорення процесу розробки та впровадження програмного забезпечення [1].

DevOps як поняття виникло на початку 2000-х років як відповідь на необхідність покращення співпраці між командами розробників (Dev) та командами, що забезпечують експлуатацію та підтримку ПЗ (Ops). Ідея полягала в тому, щоб зламати "стіну конфліктів" між цими двома групами, сприяючи культурі співпраці та автоматизації процесів.

На початку своєї історії DevOps було більше концепцією, ніж чітко визначеною практикою. Однак, з розвитком технологій та збільшенням складності програмного забезпечення, потреба в ефективній співпраці та автоматизації стала очевидною. Це призвело до розробки інструментів та методологій, які допомагають інтегрувати та автоматизувати процеси розробки, тестування та випуску ПЗ.

Одним з основних моментів в історії DevOps стало проведення першої конференції DevOpsDays у Генті, Бельгія, в 2009 році. Ця подія зібрала разом фахівців з різних областей, які поділилися своїм досвідом та ідеями щодо покращення співпраці та автоматизації в IT [2, с. 6]. З того часу DevOps

продовжує еволюціонувати, інтегруючи все більше новних підходів, таких як мікросервіси, контейнеризація, хмарні технології та інші. Ці технології допомагають організаціям швидше реагувати на зміни в ринкових умовах та потребах клієнтів, підвищуючи якість та безпеку програмного забезпечення [3].

DevOps приніс революційні зміни в процеси розробки, тестування та експлуатації програмного забезпечення, перетворивши їх на більш гнучкі, швидкі та ефективні. Завдяки автоматизації та неперервним процесам, як-от неперервна інтеграція та неперервне розгортання, організації можуть швидше реагувати на зміни в ринкових умовах та вимогах клієнтів, випускаючи оновлення та нові функції з великою частотою.

Культура співпраці, яка є основою DevOps, забезпечує тіснішу взаємодію між розробниками, тестувальниками та інженерами з експлуатації. Це призводить до кращого розуміння потреб проекту та швидшого вирішення виникаючих проблем. Автоматизоване тестування та моніторинг допомагають підтримувати високу якість продукту, виявляючи та усуваючи помилки на ранніх етапах розробки.

DevOps також сприяє підвищенню надійності та безпеки програмного забезпечення. Інтеграція практик безпеки в процеси розробки та експлуатації забезпечує більшу стійкість системи до зовнішніх загроз. Крім того, DevOps дозволяє швидко відновлювати систему після збоїв, знижуючи час простою та забезпечуючи стабільну роботу сервісів.

У цілому, DevOps змінив підхід до розробки та експлуатації програмного забезпечення, роблячи ці процеси більш адаптивними, швидкими та ефективними, що в кінцевому підсумку призводить до покращення якості продукту та задоволення клієнтів.

Отже, цей розділ дозволив нам зрозуміти, що DevOps є не просто набором інструментів чи методологією, а цілісною культурою, яка змінює підхід до розробки, тестування та експлуатації програмного забезпечення. Еволюція DevOps відбувалася паралельно з розвитком технологій та зростанням складності IT-систем, що зробило його невід'ємною частиною сучасних процесів

розробки. Основні етапи розвитку DevOps демонструють, як прагнення до покращення співпраці між командами та автоматизації процесів призвело до створення ефективніших та гнучкіших підходів до розробки програмного забезпечення.

1.2 Основні принципи та практики DevOps

У цьому розділі ми розглянемо основні принципи та практики DevOps, які лежать в основі цієї методології. Ми дослідимо, як ці принципи та практики сприяють покращенню співпраці між розробниками та інженерами з експлуатації, автоматизації процесів розробки та впровадження, а також підвищенню ефективності та якості програмного забезпечення. Ми також розглянемо, як застосування цих практик впливає на культуру організації та її здатність швидко адаптуватися до змін.

Основні принципи DevOps визначають підходи та цінності, які лежать в основі цієї філософії, і спрямовані на покращення співпраці між розробниками та інженерами з експлуатації, а також на автоматизацію процесів розробки, тестування та впровадження програмного забезпечення. Опишемо деякі з основних принципів DevOps:

1) культура співпраці. DevOps підкреслює важливість співпраці та відкритого спілкування між усіма учасниками процесу розробки і експлуатації ПЗ. Це передбачає подолання бар'єрів між відділами та створення культури, в якій команди працюють разом над досягненням спільних цілей;
2) автоматизація. DevOps наголошує на необхідності автоматизації рутинних процесів, таких як розгортання коду, тестування, моніторинг і управління інфраструктурою. Це дозволяє зменшити можливість людської помилки, підвищити ефективність та скоротити час

виведення продукту на ринок;

3) неперервна інтеграція та неперервне розгортання (CI/CD). DevOps сприяє практиці неперервної інтеграції коду та його автоматичному тестуванню, а також неперервному розгортанню змін у продуктивному середовищі. Це дозволяє швидко виявляти та виправляти помилки, забезпечуючи високу якість продукту;
4) інфраструктура як код. Управління інфраструктурою за допомогою коду дозволяє автоматизувати налаштування та розгортання середовищ, забезпечуючи їх відтворюваність та зменшуючи ризик помилок при ручному налаштуванні;
5) моніторинг та логування. Систематичний моніторинг середовища та збір логів допомагають виявляти та аналізувати проблеми в реальному часі, а також робити обґрунтовані рішення щодо оптимізації та вдосконалення системи;
6) неперервне вдосконалення. DevOps заохочує культуру постійного вдосконалення, де команди регулярно аналізують процеси та шукають способи їх оптимізації та автоматизації, прагнучи до підвищення ефективності та якості роботи.

Ці принципи допомагають створити більш гнучкі, ефективні та надійні процеси розробки та експлуатації програмного забезпечення, сприяючи швидкій адаптації до змін та високій якості продукту [4].

Основні практики DevOps допомагають реалізувати його принципи та досягти поставлених цілей. Опишемо деякі з основних практик:

1) неперервна інтеграція (Continuous Integration, CI). Це процес, в якому розробники регулярно (часто кілька разів на день) інтегрують зміни в кодову базу, автоматично тестуючи їх для виявлення та виправлення помилок якнайшвидше;
2) неперервне розгортання (Continuous Deployment, CD). Автоматичне розгортання змін у коді в тестове або продуктивне середовище після

успішного проходження тестування, забезпечуючи швидке впровадження нових функцій та оновлень;

3) інфраструктура як код (Infrastructure as Code, IaC). Використання коду для автоматизації налаштування та управління інфраструктурою, забезпечуючи її відтворюваність, масштабованість та ефективність;
4) автоматизація тестування. Розробка та використання автоматичних тестів для перевірки якості коду та функціоналу на різних етапах розробки, забезпечуючи високу якість продукту;
5) моніторинг та логування. Збір, аналіз та візуалізація даних про роботу системи та її компонентів для виявлення та вирішення проблем, а також для оптимізації продуктивності та безпеки;
6) мікросервісна архітектура. Розробка програмного забезпечення як набору незалежних, невеликих та модульних сервісів, які можна легко розгортати та масштабувати незалежно один від одного;
7) комунікація та співпраця. Застосування інструментів та практик для покращення взаємодії між членами команди, забезпечення прозорості процесів та сприяння культурі постійного навчання та обміну знаннями.

Варто зазначити, що ці практики сприяють створенню більш ефективних, гнучких та надійних процесів розробки та експлуатації програмного забезпечення, дозволяючи швидко адаптуватися до змін та задовольняти вимоги користувачів [5].

Основні практики DevOps відіграють важливу роль у формуванні культури та співпраці в рамках DevOps. Ці практики сприяють створенню середовища, де команди можуть ефективно співпрацювати, швидко адаптуватися до змін та впроваджувати інновації.

Зокрема, неперервна інтеграція та неперервне розгортання дозволяють командам швидко вносити зміни та оновлення в продукт, забезпечуючи його актуальність та відповідність потребам користувачів. Інфраструктура як код та мікросервісна архітектура сприяють ефективному управлінню ресурсами та

масштабуванню системи відповідно до потреб. Автоматизація тестування та моніторинг допомагають підтримувати високу якість продукту та швидко виявляти та усувати проблеми. Нарешті, практики комунікації та співпраці сприяють відкритості, прозорості та ефективній взаємодії між членами команди, що є важливими чинниками для успішної реалізації проектів.

Отже цей розділ дозволив нам зрозуміти, що основні принципи та практики DevOps спрямовані на створення культури співпраці, автоматизації та неперервного вдосконалення. Вони включають неперервну інтеграцію та розгортання, інфраструктуру як код, автоматизацію тестування, моніторинг та логування, а також підтримку мікросервісної архітектури. Застосування цих практик дозволяє організаціям підвищити ефективність розробки, забезпечити високу якість продукту та швидко реагувати на зміни вимог та ринкових умов. Це, в свою чергу, сприяє покращенню задоволеності користувачів та конкурентоспроможності організації в цілому.

1.3 Інструменти та технології DevOps

У цьому розділі ми розглянемо різноманітні інструменти та технології, які використовуються в рамках DevOps для підтримки його принципів та практик. Від систем контролю версій до платформ контейнеризації та інструментів автоматизації конфігурації – ми детально розглянемо, як ці інструменти сприяють автоматизації, співпраці та ефективності в процесах розробки, тестування та експлуатації програмного забезпечення.

Технології DevOps охоплюють широкий спектр інструментів та платформ, які сприяють автоматизації, співпраці та ефективності в процесах розробки, тестування та експлуатації програмного забезпечення. Опишемо деякі з основних технологій, які часто асоціюються з DevOps:

1) контейнеризація (Docker, Kubernetes) – дозволяє упаковувати додатки з

усіма їхніми залежностями в ізольовані контейнери, спрощуючи розгортання, масштабування та управління додатками;

2) автоматизація конфігурації (Ansible, Chef, Puppet) – інструменти для автоматичного налаштування та управління серверами та інфраструктурою, забезпечуючи їх відтворюваність та стандартизацію;
3) інфраструктура як код (Terraform, CloudFormation) – методика управління інфраструктурою за допомогою коду, що дозволяє автоматизувати створення, зміну та руйнування інфраструктури в хмарних сервісах;
4) неперервна інтеграція та розгортання (Jenkins, GitLab CI, Travis CI) – інструменти, що підтримують автоматизацію процесів збірки, тестування та розгортання коду, сприяючи швидкому внесенню змін та покращенню якості продукту;
5) Prometheus, ELK Stack, Grafana – інструменти для збору, зберігання та візуалізації метрик та логів, що дозволяють відстежувати стан системи та швидко реагувати на проблеми;
6) оркестрація контейнерів (Kubernetes, Docker Swarm) – системи, які дозволяють автоматизувати розгортання, масштабування та управління контейнеризованими додатками.

Ці та інші технології DevOps допомагають створювати гнучкі, масштабовані та автоматизовані рішення, які підтримують швидку розробку та впровадження програмного забезпечення, забезпечуючи високу якість та надійність продуктів [6].

Загалом, інструменти DevOps відіграють важливу роль у підтримці принципів та практик DevOps, автоматизуючи процеси розробки, тестування та експлуатації ПЗ. Отже, розглянемо деякі з основних інструментів, які часто використовуються в DevOps:

1) Git – система контролю версій, яка дозволяє розробникам вести роботу над кодом паралельно та інтегрувати зміни з мінімальними конфліктами;

2) Jenkins – інструмент автоматизації, який підтримує неперервну інтеграцію та неперервне розгортання, дозволяючи автоматизувати збірку, тестування та розгортання коду;
3) Docker – платформа для контейнеризації, яка дозволяє упаковувати програми з усіма їхніми залежностями в легковісні контейнери, що полегшує розгортання та масштабування додатків;
4) Kubernetes – система оркестрації контейнерів, яка дозволяє автоматизувати розгортання, масштабування та управління контейнеризованими додатками;
5) Ansible – інструмент управління конфігурацією, який використовується для автоматизації налаштування серверів та розгортання програм;
6) Terraform – інструмент для інфраструктури як коду, який дозволяє автоматизувати створення, зміну та версіонування інфраструктури в хмарних сервісах;
7) Prometheus – система моніторингу та алертингу, яка використовується для збору та аналізу метрик з різних джерел для виявлення та вирішення проблем в реальному часі;
8) ELK Stack (Elasticsearch, Logstash, Kibana) – набір інструментів для логування, що дозволяє збирати, зберігати та аналізувати великі обсяги логів для моніторингу та відладки додатків.

Ці та багато інших інструментів DevOps допомагають автоматизувати та оптимізувати процеси розробки, тестування та експлуатації, забезпечуючи більшу швидкість, надійність та ефективність роботи команд [7].

Отже, цей розділ демонструє, що вибір відповідних інструментів та технологій є важливим для успішної реалізації DevOps в організації. Інструменти, такі як Git, Jenkins, Docker, Kubernetes, Ansible, Terraform, Prometheus і ELK Stack, забезпечують необхідну підтримку для автоматизації процесів, забезпечення неперервності інтеграції та розгортання, моніторингу та

логування, а також управління інфраструктурою. Використання цих інструментів та технологій сприяє підвищенню ефективності, зменшенню часу виведення продукту на ринок та покращенню якості програмного забезпечення, що є основними цілями DevOps.

1.4 Роль культури та співпраці в DevOps

У цьому розділі ми зосередимося на ролі культури та співпраці в DevOps, які є основоположними елементами для успішної інтеграції та реалізації принципів DevOps у організації. Ми розглянемо, як культура відкритості, спільної відповідальності, постійного навчання, емпатії та сприяння інноваціям впливає на покращення процесу співпраці між різними командами та значно покращує загальну ефективність розробки та експлуатації програмного забезпечення.

Відкритість та прозорість у культурі DevOps є критично важливими, оскільки вони сприяють створенню середовища, де інформація вільно циркулює серед учасників процесу. Це дозволяє командам швидко адаптуватися до змін, оскільки вони мають доступ до актуальної інформації про нові вимоги, технологічні оновлення або зміни в ринкових умовах. Коли проблеми виникають, відкритість сприяє їх швидкому виявленню та аналізу, дозволяючи ефективно вирішувати їх і зменшувати час простою.

Прозорість у доступі до даних та аналітики допомагає командам базувати свої рішення на об'єктивних даних, підвищуючи їх обґрунтованість та впевненість у прийнятих рішеннях. Це також сприяє підвищенню довіри між членами команди та різними відділами, оскільки всі мають доступ до однакової інформації та працюють над спільними цілями.

У підсумку, відкритість та прозорість не лише сприяють оперативності та ефективності роботи команд, але й є основою для створення культури довіри,

співпраці та неперервного вдосконалення, яка є важливою для успішної реалізації проектів у швидкозмінному технологічному середовищі.

Спільна відповідальність у культурі DevOps є ще одним з основних принципів, який спрямований на зміцнення загальної взаємодії та співпраці між усіма розробниками та інженерами з експлуатації в команді. У традиційному підході до розробки програмного забезпечення ці команди часто працюють окремо, що може призвести до відокремленості відповідальності та непорозумінь.

В рамках DevOps обидві команди об'єднуються з метою спільного досягнення високої якості продукту та його надійної роботи. Розробники активно беруть участь не лише в написанні коду, але й в процесах його розгортання та моніторингу, тоді як інженери з експлуатації допомагають у визначенні оптимальних умов для роботи програми та вносять свій вклад у процеси розробки, щоб забезпечити більшу стабільність та безпеку.

Це співробітництво дозволяє обом командам краще розуміти виклики, з якими стикається кожна сторона, та розробляти спільні рішення, що сприяє взаєморозумінню та взаємопідтримці. Коли розробники та інженери з експлуатації несуть спільну відповідальність за продукт, вони об'єднують свої зусилля для досягнення загальної мети – створення надійного, ефективного та якісного програмного забезпечення.

Таким чином, спільна відповідальність у DevOps не лише сприяє покращенню якості продукту та його стабільності, але й підтримує позитивну культуру співпраці та взаємної підтримки між різними командами, що є важливим для успішної реалізації проектів у динамічному та складному технологічному середовищі.

Постійне навчання та вдосконалення є ще однією важливою частиною культури DevOps, оскільки воно сприяє адаптивності та інноваційності команд. У швидкозмінному технологічному світі з'являються нові інструменти, методології та практики, які можуть покращити процеси розробки та експлуатації програмного забезпечення. В рамках культури DevOps команди

підтримують культуру неперервного навчання, щоб залишатися в курсі цих нововведень та ефективно їх впроваджувати.

Це навчання не обмежується лише технічними знаннями, але й включає розвиток м'яких навичок, таких як комунікація, вирішення конфліктів та управління проектами, що є важливими для успішної співпраці та реалізації проектів.

Крім того, культура вдосконалення спонукає команди аналізувати існуючі процеси та інструменти з метою їх оптимізації. Це може включати в себе автоматизацію рутинних завдань, впровадження нових практик для підвищення ефективності роботи або вдосконалення процедур безпеки для забезпечення захисту даних та систем.

Таким чином, постійне навчання та вдосконалення у культурі DevOps допомагає командам залишатися конкурентоспроможними, підтримувати високу якість продукту та ефективно реагувати на зміни в технологічному ландшафті та потреби користувачів.

Емпатія та взаєморозуміння також відіграють важливу роль у культурі DevOps, оскільки вони сприяють створенню позитивного та підтримуючого середовища для команди. У DevOps важливо, щоб кожен член команди міг зрозуміти виклики та проблеми, з якими стикаються інші, що в свою чергу сприяє більшій співпраці та ефективності.

Емпатія означає здатність ставити себе на місце іншої особи та відчувати її переживання. У контексті DevOps це може означати, наприклад, розуміння розробниками важливості стабільності та безпеки, які є пріоритетами для інженерів з експлуатації, або визнання інженерами з експлуатації необхідності швидкого впровадження нових функцій, що є важливим для розробників.

Взаєморозуміння допомагає уникати конфліктів та непорозумінь, оскільки члени команди краще усвідомлюють мотивацію та обставини своїх колег. Це сприяє вирішенню проблем у співпраці та пошуку компромісів, які задовольняють потреби всіх сторін.

Крім того, культура емпатії та взаєморозуміння сприяє створенню

підтримуючого середовища, де члени команди відчувають себе цінними та залученими. Це збільшує їхню мотивацію та відданість проекту, а також сприяє загальному благополуччю та задоволенню від роботи.

У підсумку, емпатія та взаєморозуміння є важливими складовими культури DevOps, оскільки вони допомагають побудувати міцну, згуртовану та ефективну команду, здатну успішно впоратися з викликами сучасної розробки програмного забезпечення.

Також сприяння інноваціям в культурі DevOps є ще ожним важливим аспектом, який полягає у створенні середовища, де експериментування та креативність є цінними та підтримуються. В такій культурі команди відчувають себе впевнено, щоб випробовувати нові ідеї та підходи, не боячись негативних наслідків за помилки. Замість цього, помилки розглядаються як цінні уроки, які можуть призвести до важливих відкриттів та покращень.

Це стимулює інновації та креативність у вирішенні проблем, оскільки команди мають свободу досліджувати різні підходи та знаходити оптимальні рішення. Також це сприяє розвитку культури постійного вдосконалення, де команди прагнуть не лише вирішувати поточні задачі, а й покращувати процеси та продукти на майбутнє.

У підсумку, сприяння інноваціям у DevOps не лише підвищує ефективність та якість роботи команд, але й допомагає організаціям залишатися конкурентоспроможними в швидко змінюваному технологічному світі.

Отже, культура та співпраця в DevOps відіграють вирішальну роль у формуванні ефективних та згуртованих команд, які здатні швидко адаптуватися до змін та вирішувати складні завдання. Культура, яка підтримує відкритість, спільну відповідальність, постійне навчання, емпатію та інновації, сприяє створенню середовища, де команди можуть ефективно співпрацювати, обмінюватися знаннями та розвивати нові ідеї [8].

Така культура є важливим фактором успіху в реалізації DevOps, оскільки вона дозволяє організаціям не лише покращувати якість та швидкість розробки програмного забезпечення, але й створювати стійкі та інноваційні продукти, які

задовольняють потреби користувачів та ринку.

1.5 Вплив DevOps на процес розробки програмного забезпечення

У цьому розділі ми дослідимо, як впровадження принципів та практик DevOps впливає на процес розробки програмного забезпечення. Ми розглянемо зміни, які DevOps вносить у швидкість, якість, ефективність та співпрацю команд під час розробки, тестування та експлуатації програмного забезпечення. Також ми обговоримо, як ці зміни сприяють покращенню загальної продуктивності та задоволеності клієнтів.

Загалом, впровадження практик DevOps радикально змінює процес розробки програмного забезпечення, роблячи його більш ефективним, якісним та швидким. Завдяки неперервній інтеграції та неперервному розгортанню, команди можуть частіше вносити зміни та оновлення, що скорочує цикли розробки та дозволяє швидше реагувати на потреби користувачів та зміни на ринку. Це надає гнучкість у розробці та підвищує швидкість адаптації до нових вимог.

Автоматизація тестування та регулярна інтеграція коду допомагають виявляти та виправляти помилки на ранніх етапах, що забезпечує високу якість продукту. Співпраця між розробниками та інженерами з експлуатації дозволяє краще розуміти вимоги до продукту та забезпечувати його стабільність та надійність.

Завдяки швидким ітераціям, автоматизованому розгортанню та ефективній співпраці команд, час виведення продукту на ринок значно скорочується. Це дозволяє організаціям швидше отримувати зворотний зв'язок від користувачів та адаптуватися до конкурентного середовища.

Використання інструментів для автоматизації, таких як інфраструктура як код та оркестрація контейнерів, оптимізує управління ресурсами та процесами

розгортання. Це дозволяє командам зосередитися на інноваціях та розвитку функціоналу, замість витрачати час на рутинні операції.

Культура DevOps також сприяє розбудові відкритих та співпрацюючих команд, де розробники та інженери з експлуатації працюють разом над розробкою, тестуванням та підтримкою продукту. Це сприяє обміну знаннями, покращує розуміння цілей проекту та забезпечує більшу відповідальність за кінцевий результат.

Отже, вплив DevOps на процес розробки програмного забезпечення є значним, оскільки він сприяє створенню більш адаптивних, ефективних та якісних продуктів, забезпечуючи при цьому високий рівень задоволення користувачів та конкурентоспроможність на ринку [9].

Висновки з цього розділу підкреслюють, що DevOps має значний вплив на процес розробки програмного забезпечення. Завдяки автоматизації, неперервній інтеграції, неперервному розгортанню, тісній співпраці між командами та культурі постійного вдосконалення, DevOps сприяє створенню більш гнучких, швидких та якісних процесів розробки. Це дозволяє організаціям не лише ефективно реагувати на зміни вимог та ринкових умов, але й підвищувати задоволеність клієнтів та залишатися конкурентоспроможними в динамічному технологічному середовищі.

1.6 Висновки до 1 розділу

Отже, у першому розділі ми розглянули основні аспекти DevOps, включаючи його історію, еволюцію, принципи, практики, інструменти та технології, а також роль культури та співпраці. Ми дізналися, що DevOps є інтегрованим підходом, який зосереджений на співпраці між розробниками та інженерами з експлуатації для покращення ефективності та якості розробки програмного забезпечення.

Ми виявили, що основні принципи та практики DevOps, такі як неперервна інтеграція, неперервне розгортання, інфраструктура як код, автоматизація тестування та моніторинг, сприяють покращенню швидкості, гнучкості та якості розробки. Крім того, ми побачили, що інструменти та технології DevOps, такі як Docker, Kubernetes, Ansible, Terraform та Jenkins, відіграють важливу роль у підтримці цих практик та оптимізації процесів розробки.

Загалом, роль культури та співпраці в DevOps була визначена як необхідна та дуже важлива для успішної реалізації DevOps у організації. Культура, що сприяє відкритості, спільній відповідальності, постійному навчанню, емпатії та інноваціям, є основою для ефективної співпраці та вдосконалення процесів.

У підсумку, DevOps впливає на процес розробки програмного забезпечення, забезпечуючи більшу швидкість, гнучкість, якість та ефективність, а також покращуючи взаємодію та співпрацю команд. Це сприяє успішному впровадженню нових продуктів, задоволенню потреб користувачів та забезпеченню конкурентоспроможності організації на ринку.

2 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ОПТИМІЗАЦІЇ ПРОЦЕСУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Традиційні методи розробки програмного забезпечення

У цьому розділі ми розглянемо традиційні методи розробки програмного забезпечення, які були широко використані протягом багатьох років до появи агільних методологій. Ми ознайомимося з основними характеристиками та прикладами таких методів, включаючи водоспадну модель, V-модель, модель великого вибуху та спіральну модель. Мета цього розділу полягає в тому, щоб зрозуміти, як ці традиційні підходи організовують процес розробки та які переваги та недоліки вони мають.

Традиційні методи розробки програмного забезпечення зазвичай включають підходи, які слідують лінійному та послідовному процесу. Опишемо деякі з найбільш поширених традиційних методів:

1) водоспадна модель. Це один з найстаріших і найбільш відомих методів розробки. Він передбачає строгу послідовність етапів, таких як збір вимог, проектування, реалізація, тестування, впровадження та підтримка. Кожен етап має бути повністю завершений, перш ніж розпочнеться наступний;
2) V-модель. Ця модель є розширенням водоспадної моделі, де кожен етап розробки має відповідний етап тестування. V-модель підкреслює важливість тестування на ранніх етапах розробки та допомагає забезпечити якість продукту;
3) модель великого вибуху. У цьому підході розробка починається без чіткого розуміння вимог. Команда просто починає кодувати з мінімальним плануванням. Цей метод ризикований і часто призводить до непередбачених результатів та високих витрат;
4) спіральна модель. Хоча спіральна модель може включати ітеративні

елементи, вона також вважається традиційною через свою послідовність етапів. Модель поєднує елементи водоспадної моделі з ітеративним підходом, дозволяючи більшу гнучкість у врахуванні ризиків та змін.

Ці традиційні методи мають свої переваги у певних ситуаціях, зокрема, коли вимоги чітко визначені та малоймовірно зміняться. Однак, вони часто виявляються недостатньо гнучкими для адаптації до швидких змін у сучасному розробницькому середовищі.

Традиційні методи розробки програмного забезпечення, такі як водоспадна модель, відрізняються своєю лінійною та послідовною структурою. Вони вимагають від команди чіткого розуміння всіх вимог до проекту на початкових етапах розробки. Ці методи передбачають детальне планування та документування, що включає ретельне описання специфікацій та технічних рішень перед початком кодування.

У рамках традиційних підходів, кожен етап розробки, починаючи від аналізу вимог і закінчуючи впровадженням, виконується послідовно. Це означає, що команда не переходить до наступного етапу, поки попередній не буде повністю завершений. Такий підхід забезпечує високу ступінь контролю над процесом розробки, оскільки кожен етап підлягає перевірці та затвердженню.

Традиційні методи розробки зосереджені на створенні кінцевого продукту, який відповідає всім встановленим вимогам. Вони часто використовуються в проектах з чітко визначеними вимогами та мінімальною необхідністю змін під час розробки. Однак, у сучасному динамічному технологічному середовищі, де часто виникає потреба в швидких змінах та адаптації, агільні методи розробки, такі як Scrum або Kanban, можуть бути більш ефективними [10].

Отже, традиційні методи розробки програмного забезпечення, такі як водоспадна модель, V-модель, модель великого вибуху та спіральна модель, мають свої унікальні характеристики та застосування. Вони зазвичай слідують лінійному та послідовному підходу до розробки, з чітким розділенням етапів та великою увагою до планування та документування. Ці методи можуть бути

ефективними в проектах з добре визначеними вимогами та мінімальною потребою у змінах.

Однак, традиційні методи часто виявляються недостатньо гнучкими для адаптації до швидких змін та непередбачуваних обставин, які є поширеними в сучасній розробці програмного забезпечення. Обмежена гнучкість у внесенні змін та високий рівень жорсткості в процесі можуть призвести до затримок та збільшення витрат. Тому, хоча традиційні методи залишаються важливою частиною історії розробки програмного забезпечення, в багатьох випадках вони поступаються місцем більш гнучким та адаптивним підходам, таким як агільні методології.

2.2 Недоліки традиційних методів з точки зору DevOps

У цьому розділі ми розглянемо, як різні методи розробки програмного забезпечення можуть бути неефективними або обмеженими з точки зору принципів та практик DevOps. Ми дослідимо різні недоліки традиційних методів, включаючи їхню низьку гнучкість, відсутність співпраці між командами, повільний процес доставки, обмежений зворотній зв'язок від користувачів та ризик виходу продукту за межі бюджету та термінів.

Мета цього розділу полягає в тому, щоб зрозуміти, чому традиційні методи можуть бути несумісними з цілями DevOps та які виклики вони створюють для організацій, які прагнуть до швидкої та ефективної розробки та доставки програмного забезпечення.

Традиційні методи розробки програмного забезпечення мають кілька недоліків, особливо коли їх розглядати з перспективи DevOps, який прагне до швидкої реакції на зміни, ефективної співпраці між командами та неперервної доставки продукту. Опишемо деякі з основних недоліків традиційних методів з точки зору DevOps:

1) низька гнучкість та адаптивність. Традиційні методи, такі як водоспадна модель, мають жорстку структуру, що ускладнює внесення змін після того, як проект розпочато. Це суперечить принципам DevOps, які передбачають швидку адаптацію до змінних вимог та умов ринку;

2) відсутність співпраці між командами. Традиційні методи часто призводять до ізоляції команд розробників та команд інженерів з експлуатації. В результаті виникають "стіни" між відділами, що ускладнює ефективну співпрацю та комунікацію. DevOps прагне до руйнування цих бар'єрів та створення культури спільної відповідальності;

3) повільний процес доставки. Через лінійну та послідовну природу традиційних методів, час від розробки до впровадження продукту може бути довгим. Це суперечить прагненню DevOps до швидкої та неперервної доставки продукту;

4) обмежений зворотній зв'язок від користувачів. У традиційних методах зворотній зв'язок від користувачів зазвичай отримується після повного циклу розробки. В DevOps же акцент робиться на ранньому та частому залученні користувачів до процесу, щоб швидко вносити необхідні корективи та покращувати продукт;

5) ризик виходу продукту за межі бюджету та термінів. Оскільки традиційні методи вимагають детального планування на початкових етапах і мають обмежену гнучкість, існує ризик, що проект може перевищити встановлені бюджети та терміни через непередбачені зміни або проблеми.

У підсумку, хоча традиційні методи розробки можуть бути ефективними в деяких сценаріях, вони часто не відповідають вимогам сучасного динамічного технологічного середовища та принципам DevOps, які прагнуть до швидкої адаптації, тісної співпраці та неперервної доставки високоякісних продуктів.

Варто зазанчити, що традиційні методи розробки ПЗ, такі як водоспадна модель, мають свої переваги в деяких сценаріях, особливо коли вимоги до проекту чітко визначені на початковому етапі та малоймовірно зміняться протягом процесу розробки. Ці методи забезпечують строгу структуру, детальне

планування та документування, що може бути корисним для великих та складних проектів у стабільних умовах.

Однак, у сучасному динамічному технологічному середовищі, де зміни вимог і технологій відбуваються швидко, традиційні методи часто виявляються недостатньо гнучкими для адекватної адаптації. Вони можуть призводити до затримок у реагуванні на зміни, що, у свою чергу, може вплинути на конкурентоспроможність продукту та задоволеність клієнтів.

Крім того, традиційні методи часто розділяють команди розробників та інженерів з експлуатації, що може призвести до недоліків у співпраці та комунікації. Це створює бар'єри між відділами, що ускладнює ефективне вирішення проблем та впровадження інновацій.

Принципи DevOps, навпаки, прагнуть до швидкої адаптації до змін, тісної співпраці між командами та неперервної доставки високоякісних продуктів. DevOps підтримує гнучкість та інноваційність шляхом заохочення спільної відповідальності, автоматизації процесів та постійного зворотного зв'язку від користувачів. Такий підхід допомагає організаціям швидше реагувати на ринкові зміни, покращувати якість продукту та забезпечувати високий рівень задоволеності клієнтів.

Отже, хоча традиційні методи можуть бути ефективними в певних умовах, вони часто не відповідають потребам сучасного динамічного технологічного середовища. Принципи та практики DevOps пропонують більш гнучкий та адаптивний підхід, який дозволяє організаціям швидко розвиватися та підтримувати конкурентоспроможність у постійно змінюваному світі.

2.3 Порівняння ефективності різних методів

У цьому розділі ми здійснимо порівняльний аналіз ефективності різних методів розробки програмного забезпечення, зокрема традиційних підходів,

таких як водоспадна модель, та агільних методів, включаючи Scrum, Kanban та інші. Ми оцінимо їхні переваги та недоліки на основі критеріїв, таких як гнучкість, швидкість реакції на зміни, якість продукту, співпраця команд та задоволеність клієнтів. Загальною метою цього розділу є допомога організаціям у виборі найбільш підходящого методу розробки для їхніх конкретних проектів та цілей.

Варто зазначити, що крім традиційних існують також агільні методи розробки ПЗ. Агільні методи розробки відрізняються від традиційних підходів своєю гнучкістю та зосередженістю на співпраці та швидкій відповіді на зміни. Вони передбачають ітеративний процес, де проект розбивається на невеликі частини, які розробляються та тестуються протягом коротких циклів, відомих як спринти. Це дозволяє командам швидко адаптуватися до змінних вимог та умов ринку, а також забезпечує постійний зворотний зв'язок від користувачів та зацікавлених сторін.

Однією з основних характеристик агільних методів є співпраця між усіма членами команди та замовником. Комунікація та взаємодія вважаються важливішими, ніж жорстке дотримання планів та документації. Агільні методи також підкреслюють важливість гнучкості та відкритості до змін, навіть на пізніх етапах розробки.

Крім того, агільні методи покладаються на самоорганізовані команди, які відповідальні за вирішення проблем та досягнення цілей проекту. Вони прагнуть до мінімізації бюрократії та надмірного планування, замість цього зосереджуючись на створенні цінності для користувачів та неперервному вдосконаленні процесів.

Агільні методи, такі як Scrum, Kanban, Extreme Programming (XP) та інші, надають рамки та практики для ефективної розробки в умовах невизначеності та змін. Вони допомагають командам досягати більшої прозорості, швидкості та якості продукту, одночасно зберігаючи високий рівень задоволеності клієнтів та учасників проекту [11].

Порівняння ефективності різних методів розробки програмного

забезпечення, таких як традиційні підходи (водоспадна модель) та агільні методи (наприклад, Scrum або Kanban), дозволяє зрозуміти їхні переваги та недоліки в різних контекстах. Цей розділ буде зосереджений на аналізі критеріїв, таких як гнучкість, швидкість реакції на зміни, якість продукту, співпраця команд та задоволеність клієнтів, для порівняння ефективності різних методів розробки.

Традиційні методи, такі як водоспадна модель, можуть бути ефективними для проектів з чітко визначеними вимогами та мінімальними змінами. Вони забезпечують детальне планування та контроль, що може бути корисним для великих та складних проектів у стабільних умовах. Однак, вони часто виявляються недостатньо гнучкими та повільними для адаптації до змін, що є критичним у динамічному технологічному середовищі.

З іншого боку, агільні методи, такі як Scrum та Kanban, зосереджені на гнучкості, швидкій адаптації до змін та неперервній взаємодії з клієнтами. Вони підтримують ітеративний процес розробки, швидке впровадження нових функцій та зворотний зв'язок від користувачів. Агільні методи покращують співпрацю в командах та забезпечують більш високу задоволеність клієнтів завдяки швидкій доставці продукту.

Порівняння ефективності різних методів розробки вимагає врахування специфіки проекту, ринкових умов, ресурсів та культури організації. У деяких випадках гібридні підходи, які поєднують елементи традиційних та агільних методів, можуть бути найбільш ефективними для досягнення цілей проекту [12]. Отже, порівняння ефективності різних методів розробки програмного забезпечення виявило, що кожен метод має свої переваги та обмеження залежно від специфіки проекту та середовища. Традиційні підходи, такі як

водоспадна модель, можуть бути корисними для проектів з чітко визначеними вимогами та мінімальною необхідністю у змінах, але вони часто виявляються недостатньо гнучкими та повільними для адаптації до швидких змін.

З іншого боку, агільні методи, такі як Scrum і Kanban, забезпечують

високу гнучкість, швидку адаптацію до змін та ефективну співпрацю між членами команди та замовниками. Вони підтримують ітеративний процес розробки, забезпечуючи швидку доставку продукту та високий рівень задоволеності клієнтів.

У деяких випадках гібридні підходи, які поєднують елементи традиційних та агільних методів, можуть бути найбільш ефективними, оскільки вони дозволяють організаціям гнучко підходити до розробки програмного забезпечення, адаптуючись до різних вимог та умов проекту. Вибір підходу до розробки залежить від цілей організації, характеру проекту та необхідності адаптації до змінних умов.

2.4 Висновки до 2 розділу

Отже, у другому розділі ми провели аналіз існуючих методів оптимізації процесу розробки програмного забезпечення, зокрема розглянули традиційні методи, виявили їх недоліки з точки зору DevOps та порівняли ефективність різних підходів.

Ми встановили, що традиційні методи розробки, такі як водоспадна модель, мають свої переваги в певних сценаріях, але вони часто виявляються недостатньо гнучкими та адаптивними для сучасного динамічного технологічного середовища. Їхні основні недоліки включають низьку гнучкість, відсутність співпраці між командами, повільний процес доставки та обмежений зворотній зв'язок від користувачів.

Порівняння ефективності різних методів показало, що агільні методи, такі як Scrum та Kanban, надають більшу гнучкість, швидкість адаптації та ефективну співпрацю, що відповідає вимогам DevOps. Вони підтримують ітеративний процес розробки, забезпечують швидку доставку продукту та високий рівень задоволеності клієнтів.

У підсумку, аналіз існуючих методів оптимізації процесу розробки програмного забезпечення підкреслює важливість вибору підходу, який найкраще відповідає цілям проекту, специфіці середовища та потребам користувачів. Агільні методи, які підтримують принципи DevOps, забезпечують ефективний інструментарій для досягнення гнучкості, швидкості та якості у процесі розробки програмного забезпечення.

3 ПРАКТИЧНЕ ЗАСТОСУВАННЯ МЕТОДІВ DEVOPS ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕСУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір інструментів та технологій для реалізації DevOps

У цьому розділі ми зосередимося на важливості правильного вибору інструментів та технологій для ефективної реалізації практик DevOps. Вибір підходящих інструментів є важливим фактором успіху в автоматизації процесів, співпраці між командами та забезпеченні безперервної інтеграції та розгортання. Ми розглянемо різні категорії інструментів, які використовуються в DevOps, та надамо рекомендації щодо їх вибору залежно від потреб проекту та організації.

Спочатку розглянемо інструменти для управління версіями та контролю коду.

1) Git – це розподілена система контролю версій, яка дозволяє розробникам ефективно керувати змінами в коді та співпрацювати над проектами. Вона забезпечує швидке створення та злиття гілок, відстеження історії змін та можливість відкату до попередніх версій коду. Git підтримує розподілену роботу, дозволяючи кожному розробнику мати локальну копію репозиторію, що сприяє паралельній розробці та знижує залежність від центрального сервера.
2) GitHub – це веб-сервіс для хостингу Git-репозиторіїв який надає інструменти для співпраці та управління проектами. Він дозволяє розробникам спільно працювати над кодом, використовуючи функції, такі як pull requests, issues, code review та wiki. GitHub також інтегрується з багатьма іншими сервісами та інструментами розробки, що робить його популярним вибором для управління вихідним кодом у проектах різного масштабу.
3) GitLab – це платформа для управління життєвим циклом програмного забезпечення, яка включає в себе хостинг Git-репозиторіїв, а також інструменти для неперервної інтеграції/доставки (CI/CD), відстеження

задач, огляду коду та багато іншого. GitLab підтримує автоматизацію різних етапів розробки та забезпечує комплексне рішення для команд, які прагнуть досягти ефективності та автоматизації процесів.

4) Bitbucket – це веб-сервіс для хостингу репозиторіїв, що підтримує як Git, так і Mercurial. Він надає інструменти для співпраці, такі як pull requests та code review, а також інтеграцію з Jira та Trello для відстеження задач. Bitbucket також пропонує функції неперервної інтеграції та доставки через Bitbucket Pipelines, дозволяючи автоматизувати процеси збірки, тестування та розгортання програмного забезпечення.

Далі розглянемо інструменти для неперервної інтеграції та доставки (CI/CD).

1) Jenkins – це відкритий інструмент для автоматизації різних завдань у процесах неперервної інтеграції та неперервного розгортання. Він дозволяє створювати та налаштовувати пайплайни для автоматичної збірки, тестування та розгортання коду. Jenkins підтримує широкий спектр плагінів, які розширюють його функціональність і дозволяють інтегрувати його з різними інструментами та платформами, що робить його дуже гнучким рішенням для автоматизації процесів розробки.
2) CircleCI – це хмарний сервіс для автоматизації CI/CD, який забезпечує швидке налаштування та виконання пайплайнів. Він підтримує автоматичне тестування та розгортання коду за кожним комітом, забезпечуючи швидку зворотню відповідь та високу якість продукту. CircleCI інтегрується з GitHub та Bitbucket, дозволяючи легко налаштувати процеси CI/CD для репозиторіїв, розміщених на цих платформах.
3) Travis CI – це ще один популярний облачний сервіс для автоматизації CI/CD. Він легко інтегрується з GitHub, автоматично запускаючи процеси збірки та тестування для проектів після кожного коміту або

pull request. Travis CI підтримує багато мов програмування та фреймворків, дозволяючи налаштувати пайплайни відповідно до потреб проекту.

4) GitLab CI є частиною платформи GitLab і забезпечує інтегровані інструменти для автоматизації CI/CD. Він дозволяє створювати пайплайни безпосередньо у репозиторіях GitLab, використовуючи конфігураційний файл .gitlab-ci.yml. GitLab CI підтримує паралельне виконання завдань, кешування залежностей та різні середовища для розгортання, що робить його ефективним рішенням для комплексної автоматизації процесів розробки та розгортання.

Наступним кроком розглянемо інструменти для управління інфраструктурою та конфігурацією.

1) Terraform – це інструмент для управління інфраструктурою як кодом (Infrastructure as Code, IaC), розроблений HashiCorp. Він дозволяє користувачам описувати інфраструктуру за допомогою конфігураційних файлів, що спрощує процес створення, зміни та управління різними ресурсами в хмарних сервісах, таких як AWS, Google Cloud та Azure. Terraform використовує декларативний підхід, де користувачі визначають бажаний стан інфраструктури, а Terraform автоматично визначає необхідні кроки для досягнення цього стану. Це дозволяє ефективно управляти ресурсами, забезпечуючи їхню відтворюваність та безпеку.
2) Ansible – це інструмент для автоматизації конфігурації серверів та управління інфраструктурою. Він використовує просту мову опису конфігурацій у форматі YAML, що робить його зрозумілим та легким у використанні. Ansible не вимагає встановлення агентів на віддалених машинах, використовуючи замість цього SSH для виконання команд та переносу файлів. Це дозволяє легко автоматизувати розгортання додатків, управління конфігураціями та оркестрацію завдань на

великій кількості серверів.

3) Chef – це інструмент для автоматизації конфігурації серверів та інфраструктури. Він використовує код для опису конфігурації інфраструктури та розгортання додатків, забезпечуючи їхню відтворюваність та автоматизацію. Chef використовує Ruby як мову для написання рецептів (recipes) і куховарських книг (cookbooks), які описують бажаний стан системи та необхідні кроки для його досягнення. Chef підтримує роботу з різними платформами та сервісами, включаючи хмарні сервіси, віртуалізацію та контейнеризацію.
4) Puppet – це інструмент для автоматизації управління конфігураціями інфраструктури. Він дозволяє користувачам описувати бажаний стан системи за допомогою декларативної мови та автоматично застосовувати ці конфігурації до великої кількості серверів. Puppet використовує модель клієнт-сервер, де агенти Puppet на віддалених машинах комунікують з майстер-сервером для отримання та застосування конфігурацій. Це дозволяє централізовано управляти конфігураціями та забезпечує їхню консистентність та безпеку у великих інфраструктурах.

Також опишемо інструменти для контейнеризації та оркестрації.

1) Docker – це відкрита платформа для розробки, доставки та запуску додатків у легковісних контейнерах. Контейнеризація з Docker дозволяє упакувати додаток разом із його середовищем та залежностями в стандартизований блок, що забезпечує консистентність середовища на всіх етапах розробки, тестування та продуктивного використання. Docker використовує Dockerfile для опису інструкцій збірки образу контейнера, який можна легко розгортати та масштабувати на будь-якій платформі, що підтримує Docker. Це спрощує процес розгортання додатків, зменшує конфлікти між середовищами та покращує використання ресурсів за рахунок

ізоляції додатків у контейнерах.

2) Kubernetes (або K8s) – це відкрита система для автоматизації розгортання, масштабування та управління контейнеризованими додатками. Розроблений Google, Kubernetes став стандартом де-факто для оркестрації контейнерів, підтримуючи роботу великих та складних розподілених систем на різних обчислювальних середовищах: від локальних серверів до хмарних платформ. Kubernetes пропонує механізми для автоматичного розміщення, самовідновлення та масштабування додатків, керування конфігурацією та секретами, балансування навантаження між контейнерами та багато іншого. Використання Kubernetes дозволяє розробникам зосередитися на розробці додатків, не турбуючись про інфраструктурні виклики, пов'язані з їх розгортанням та масштабуванням.

Наостанок розглянемо інструменти для моніторингу та легування:

1) Prometheus – це відкритий інструмент для моніторингу та алертингу, який широко використовується в DevOps для збору та зберігання метрик у реальному часі. Він дозволяє визначати часові ряди даних, зібраних з моніторованих об'єктів, таких як сервери, мікросервіси або бази даних. Prometheus використовує простий та потужний запитовий мову PromQL для обробки та аналізу метрик, що дозволяє користувачам легко створювати алерти та відстежувати стан системи. Завдяки своїй архітектурі та ефективності, Prometheus став популярним вибором для моніторингу у контейнеризованих та розподілених середовищах, таких як Kubernetes.
2) Grafana – це відкрита платформа для аналітики та візуалізації даних, яка дозволяє створювати красиві та інформативні дашборди для моніторингу метрик. Grafana підтримує інтеграцію з різними джерелами даних, включаючи Prometheus, InfluxDB, Elasticsearch та багатьма іншими. Вона надає широкий спектр віджетів, таких як графіки, термометри, карти та інші, для візуалізації різноманітних

метрик і показників. Grafana дозволяє користувачам легко налаштувати вигляд дашбордів, керувати доступом та спільно використовувати їх з командою або замовниками.

3) Elastic Stack, також відомий як ELK Stack, – це набір інструментів для збору, обробки та аналізу великих обсягів логів та інших даних. Він складається з трьох основних компонентів:
4) Elasticsearch – пошуковий та аналітичний движок, який забезпечує швидке зберігання, пошук та аналіз великих обсягів даних;
5) Logstash – інструмент для збору та трансформації даних з різних джерел, таких як логи, метрики та події, перед їхнім відправленням до Elasticsearch;
6) Kibana – веб-інтерфейс для візуалізації даних, збережених в Elasticsearch. Kibana дозволяє створювати дашборди, графіки та карти для інтерактивного аналізу та моніторингу даних.
7) Elastic Stack широко використовується для централізованого логування, моніторингу та аналізу логів від різних частин системи, забезпечуючи глибоке розуміння стану та поведінки додатків та інфраструктури.

При виборі інструментів та технологій для реалізації DevOps важливо враховувати специфіку проекту, вимоги до інфраструктури, командну культуру та інші фактори, щоб забезпечити максимальну ефективність та відповідність цілям організації.

Отже, правильний вибір інструментів та технологій є фундаментальним аспектом успішної реалізації DevOps. Використання відповідних інструментів для управління версіями, автоматизації CI/CD, конфігурації інфраструктури, контейнеризації та моніторингу може суттєво покращити ефективність розробки, забезпечити високу якість продукту та спростити управління інфраструктурою. Важливо враховувати специфіку проекту та організаційні потреби при виборі інструментів, щоб забезпечити їхню сумісність та

максимальну ефективність. Застосування правильних інструментів та технологій у рамках стратегії DevOps дозволяє організаціям досягти більшої гнучкості, швидкості та інноваційності у своїх проектах.

3.2 Автоматизація процесів з використанням DevOps інструментів

У цьому розділі ми зосередимося на практичному застосуванні DevOps інструментів для автоматизації основних процесів у розробці ПЗ. Ми розглянемо приклади використання популярних інструментів, таких як Jenkins для неперервної інтеграції, Terraform для управління інфраструктурою як код, та Docker з Docker Compose для розгортання контейнеризованих додатків. Цей розділ демонструє, як за допомогою автоматизації можна підвищити ефективність процесів, знизити ймовірність помилок та прискорити доставку продукту до кінцевого користувача.

Приклад 1: автоматизація неперервної інтеграції з Jenkins (ліст. 3.1). Jenkins – це відкритий інструмент для автоматизації неперервної інтеграції та неперервного розгортання. Він дозволяє створювати та тестувати код автоматично кожного разу, коли відбуваються зміни у репозиторії.

Лістинг коду 3.1 – Приклад автоматизації неперервної інтеграції з Jenkins

```
groovy pipeline {
agent any stages {
stage('Build') { steps {
echo 'Building...'
// Команди для збірки проекту
}
}
stage('Test') { steps {
echo 'Testing...'
// Команди для запуску тестів
}
}
```

```
stage('Deploy') { steps {
echo 'Deploying...'
// Команди для розгортання проекту
}
}
}
}
```

У цьому прикладі ми використовуємо Jenkins для автоматизації процесу неперервної інтеграції. Jenkins слідкує за змінами у репозиторії коду і автоматично виконує певні дії, такі як збірка проекту, запуск тестів та розгортання продукту. Цей процес забезпечує швидке виявлення та виправлення помилок, а також підтримує високу якість коду.

Приклад 2: конфігурація інфраструктури як код з Terraform (ліст. 3.2). Terraform – це інструмент для створення, зміни та версіонування інфраструктури безпечним та ефективним способом. Він дозволяє описувати інфраструктуру за допомогою конфігураційних файлів.

Лістинг коду 3.2 – Приклад конфігурації інфраструктури як код з Terraform

```
hcl
resource "aws_instance" "example" {
ami             = "ami-
0c55b159cbfafe1f0" instance_type =
"t2.micro"

tags = {
Name = "ExampleInstance"
}
}
```

Використання Terraform дозволяє описати інфраструктуру у вигляді коду, що спрощує процес її створення, зміни та управління. У цьому прикладі ми використовуємо Terraform для створення екземпляра EC2 у AWS. Завдяки цьому підходу інфраструктура стає легко відтворюваною та поділяється на версії, що підвищує надійність та зменшує ризик помилок при ручному налаштуванні.

Приклад 3: автоматизація розгортання з Docker та Docker Compose (ліст.

3.3). Docker дозволяє упаковувати додатки та їх залежності в контейнери, а Docker Compose спрощує управління багатоконтейнерними додатками.

Лістинг коду 3.3 – Приклад автоматизації розгортання з Docker та Docker Compose

```
yaml version: '3' services:
web:
image: "my-web-app:latest" ports: - "80:80"
db:
image: "mysql:5.7" environment:
MYSQL_ROOT_PASSWORD: "example"
```

У цьому прикладі ми використовуємо Docker та Docker Compose для автоматизації процесу розгортання додатків. Docker дозволяє упаковувати додаток разом з його залежностями у контейнер, що забезпечує його однакову роботу в різних середовищах. Docker Compose допомагає керувати багатоконтейнерними додатками, визначаючи та запускаючи всі необхідні сервіси за допомогою одного конфігураційного файлу. Це спрощує процес розгортання та управління додатками.

Кожен з цих прикладів демонструє, як застосування певних інструментів та практик DevOps може автоматизувати та оптимізувати різні аспекти процесу розробки програмного забезпечення, від неперервної інтеграції до розгортання додатків.

Отже, у цьому розділі ми розглянули лише кілька прикладів автоматизації з використанням DevOps інструментів. Реальні сценарії можуть бути значно складнішими та вимагати інтеграції різних інструментів та практик для досягнення оптимальних результатів.

Автоматизація процесів з використанням DevOps інструментів є важливою складовою сучасної розробки програмного забезпечення. Інструменти, такі як Jenkins, Terraform та Docker, дозволяють автоматизувати різні етапи розробки, від неперервної інтеграції до розгортання додатків. Це сприяє підвищенню ефективності, зменшенню часу на впровадження змін та покращенню якості продукту. Розглянуті практичні приклади демонструють, як

автоматизація може бути застосована у реальних проектах, що дозволяє організаціям оптимізувати свої процеси розробки та досягти конкурентних переваг на ринку.

3.3 Моніторинг та логування в DevOps

У цьому розділі ми зосередимося на ролі моніторингу та логування в контексті DevOps та розглянемо, як ці процеси можуть бути автоматизовані за допомогою відповідних інструментів. Моніторинг та логування є важливими компонентами DevOps, оскільки вони дозволяють командам відстежувати стан системи, виявляти та аналізувати проблеми, а також забезпечувати високу якість та стабільність ПЗ.

Ми розглянемо практичні приклади використання таких інструментів, як Prometheus для моніторингу метрик, Elastic Stack (ELK) для логування та аналізу логів, та Grafana для візуалізації даних. Цей розділ демонструє, як автоматизація моніторингу та логування може покращити прозорість системи та сприяти ефективній взаємодії між різними членами DevOps команди.

Приклад 1: моніторинг за допомогою Prometheus. Prometheus – це відкритий інструмент моніторингу та алертингу, який часто використовується в DevOps для збору та аналізу метрик.

Лістинг коду 3.4 – Приклад моніторингу за допомогою Prometheus

```
yaml
# Приклад конфігурації для Prometheus global:
scrape_interval: 15s

scrape_configs:
- job_name: 'my-application' static_configs:
- targets: ['localhost:8080']
```

Prometheus широко використовується в DevOps для збору та зберігання метрик в реальному часі. У цьому прикладі ми налаштовуємо Prometheus для

збору метрик з додатка, що працює на локальному хості порту 8080.

Конфігураційний файл визначає інтервал збору даних (scrape_interval) та ціль для моніторингу (targets). Після налаштування Prometheus може відстежувати різні параметри додатка, такі як час відповіді, кількість запитів та використання ресурсів.

Приклад 2: логування з Elastic Stack. Elastic Stack (відомий також як ELK Stack) поєднує в собі Elasticsearch, Logstash та Kibana для збору, зберігання та візуалізації логів.

Лістинг коду 3.5 – Приклад логування з Elastic Stack

```
json
{
"message": "Приклад логування", "level": "INFO",
"timestamp": "2021-01-01T12:00:00Z"
}
```

Elastic Stack є популярним рішенням для збору, обробки та аналізу логів.

У цьому прикладі ми використовуємо JSON формат для структурування логів, що дозволяє легко інтегрувати їх з Elastic Stack. Logstash може збирати логи з різних джерел, трансформувати їх за необхідності та передавати до Elasticsearch для зберігання. Kibana використовується для візуалізації та аналізу логів, дозволяючи користувачам створювати графіки, дашборди та виконувати пошук по логах.

Приклад 3: використання Grafana для візуалізації метрик. Grafana – це інструмент для візуалізації метрик, який може бути інтегрований з різними джерелами даних, включаючи Prometheus.

Лістинг коду 3.6 – Приклад використання Grafana для візуалізації метрик

```
ini
# Приклад конфігурації панелі в Grafana [dashboard]
title = "Моя панель" uid = "my-dashboard"
tags = ["performance", "metrics"] timezone = "browser"

# Визначення віджета для відображення часового ряду [widget
"cpu_usage"]
```

```
type = "graph" datasource = "Prometheus"
title = "Використання CPU" span = 6

# Конфігурація запиту до джерела даних [widget "cpu_usage".series]
name = "CPU Usage"
query = "rate(process_cpu_seconds_total{job='my-app'}[5m])"
```

У цьому прикладі ми налаштовуємо дашборд у Grafana для відображення метрик, зібраних Prometheus. Конфігураційний файл визначає назву дашборду, унікальний ідентифікатор (uid) та інші параметри. Після налаштування користувачі можуть переглядати важливі метрики у вигляді графіків та діаграм, що допомагає у прийнятті рішень та моніторингу стану системи.

У цьому прикладі ми додали декілька додаткових параметрів до конфігурації панелі в Grafana, включаючи теги для панелі, часовий пояс для відображення даних та визначення віджета (widget) для графічного відображення часового ряду метрик використання CPU. Конфігурація запиту до джерела даних (у цьому випадку Prometheus) вказує, які дані слід відображати на графіку.

Загалом, кожен з цих прикладів демонструє, як застосування інструментів для моніторингу та логування може покращити здатність команд DevOps відстежувати, аналізувати та реагувати на події у системі, забезпечуючи високий рівень прозорості та контролю над розробкою та експлуатацією програмного забезпечення.

3.4 Впровадження практик безпеки в процес розробки

У цьому розділі ми зосередимо увагу на важливості інтеграції практик безпеки в процес розробки програмного забезпечення. Забезпечення безпеки є важливим аспектом DevOps, який відомий як DevSecOps. Ми розглянемо практичні приклади використання інструментів та методологій для виявлення та

усунення потенційних загроз на ранніх етапах розробки. Це включає сканування вразливостей, перевірку коду перед комітом та інтеграцію тестів безпеки в процеси CI/CD. Метою цього розділу є показати, як застосування цих практик може покращити безпеку програмного забезпечення та зменшити ризики для бізнесу.

Приклад 1: сканування вразливостей з використанням OWASP ZAP. OWASP – це інструмент для автоматичного тестування веб-додатків на наявність вразливостей.

Лістинг коду 3.7 – Приклад використання OWASP ZAP

```
bash
# Запуск сканування вразливостей для веб-додатку
zap-cli quick-scan --self-contained --start-options '-config api.disablekey=true' http://example.com
```

У цьому прикладі команда zap-cli quick-scan запускає швидке сканування веб-додатку за вказаною адресою (наприклад, http://example.com). Це дозволяє виявити потенційні слабкі місця, такі як SQL ін'єкції або XSS атаки, на ранніх етапах розробки, що сприяє підвищенню безпеки продукту.

Приклад 2: використання Git Hooks для перевірки коду перед комітом. Git Hooks можуть бути використані для автоматичної перевірки коду на відповідність стандартам безпеки перед його комітом в репозиторій.

Лістинг коду 3.8 – Приклад використання Git Hooks

```
bash #!/bin/sh
# pre-commit hook для перевірки коду


# Перевірка коду за допомогою лінтера lint_result=$(eslint .)

if [ "$lint_result" != "" ]; then
echo "Помилки лінтера виявлені, коміт скасовано:" echo "$lint_result"
exit 1
fi
exit 0
```

Кінець лістингу 3.8У цьому прикладі використовується pre-commit hook,

який запускає лінтер (наприклад, ESLint) для аналізу коду. Якщо лінтер виявляє помилки, коміт скасовується, і розробник отримує повідомлення про необхідність виправлення коду. Це допомагає запобігти внесенню небезпечного або неякісного коду в репозиторій.

Приклад 3: інтеграція тестів безпеки в CI/CD з використанням Snyk. Snyk – це інструмент, який дозволяє інтегрувати тестування безпеки безпосередньо в процеси CI/CD, забезпечуючи автоматичне виявлення вразливостей у залежностях yaml.

Лістинг коду 3.9 – Приклад використання Snyk

```
# Приклад конфігурації GitHub Actions для інтеграції Snyk name: Security Test on: [push] jobs:
snyk:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v2
- name: Run Snyk to check for vulnerabilities uses: snyk/actions/node@master
env:
SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
```

У цьому прикладі показано конфігурацію GitHub Actions, яка запускає Snyk для перевірки безпеки при кожному push в репозиторій. Це дозволяє виявляти та усувати потенційні загрози на ранніх етапах розробки, забезпечуючи неперервну безпеку ПЗ.

Кожен з цих прикладів демонструє, як можна інтегрувати практики безпеки в процес розробки для виявлення та усунення потенційних загроз, забезпечуючи безпеку програмного забезпечення на всіх етапах його життєвого циклу.

Отже, впровадження практик безпеки в процес розробки є важливим елементом стратегії DevSecOps, яка покликана забезпечити безпеку програмного забезпечення на всіх етапах його життєвого циклу. Як показано в цьому розділі, використання інструментів, таких як OWASP ZAP, Git Hooks та Snyk, дозволяє автоматизувати виявлення та усунення вразливостей, перевіряти код на

відповідність стандартам безпеки та інтегрувати тестування безпеки в процеси CI/CD.

Загалом, це сприяє ранньому виявленню та виправленню проблем, знижує ризики для бізнесу та підвищує довіру користувачів до продукту. Отже, інтеграція практик безпеки в процес розробки є важливою складовою успішної стратегії DevOps.

3.5 Висновки до 3 розділу

Отже, у третьому розділі курсової роботи ми розглянули практичне застосування різних методів DevOps для оптимізації процесу розробки програмного забезпечення, включаючи вибір інструментів та технологій, автоматизацію процесів, моніторинг та логування, а також впровадження практик безпеки.

Ми визначили основні категорії інструментів, необхідних для успішної реалізації DevOps, включаючи системи контролю версій, інструменти для CI/CD, управління інфраструктурою, контейнеризацію та моніторинг. Правильний вибір інструментів є надзвичайно важливим для ефективної автоматизації та співпраці в процесі розробки.

Ми розглянули приклади автоматизації основних процесів, таких як неперервна інтеграція, неперервне розгортання та управління інфраструктурою, з використанням інструментів, таких як Jenkins, Terraform та Docker. Автоматизація дозволяє зменшити ручні втручання, збільшити швидкість розгортання та підвищити якість продукту.

Ми вивчили важливість моніторингу та логування для забезпечення стабільності та прозорості системи. Інструменти, такі як Prometheus, Grafana та Elastic Stack, дозволяють відстежувати метрики, аналізувати логи та виявляти проблеми в реальному часі.

Також ми підкреслили, яке значення має інтеграція основних практик безпеки в DevOps, відомих як DevSecOps. Інструменти, такі як OWASP ZAP, Git Hooks та Snyk, допомагають виявляти вразливості, перевіряти код на відповідність стандартам безпеки та інтегрувати тестування безпеки в процеси CI/CD.

У підсумку, практичне застосування методів DevOps значно покращує ефективність процесу розробки ПЗ, забезпечуючи швидкість, якість та безпеку продукту. Використання відповідних інструментів та практик дозволяє організаціям досягати високого рівня автоматизації, співпраці та інновацій у своїх проектах.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ТА РЕКОМЕНДАЦІЇ

4.1 Оцінка ефективності впровадження DevOps

У цьому розділі ми зосередимо увагу на методах оцінки ефективності впровадження практик DevOps в організації. Оцінка дозволяє визначити, наскільки успішно інтегровані процеси DevOps сприяють поліпшенню процесу розробки програмного забезпечення, збільшенню продуктивності команди та підвищенню якості продукту. Ми розглянемо основні метрики та показники, які допоможуть оцінити досягнення цілей DevOps та виявити потенційні області для подальшого вдосконалення.

Оцінка ефективності впровадження DevOps є важливим кроком для визначення впливу практик DevOps на процес розробки програмного забезпечення та загальну продуктивність організації. Для цього можна використовувати різні показники та методики:

1) швидкість розгортання:
    a) вимірює частоту розгортання коду в продуктивне середовище;
    b) збільшення частоти розгортання свідчить про підвищену здатність команди швидко впроваджувати нові функції та виправлення;
2) час від початку до розгортання:
    a) визначає час, необхідний для переходу від створення зміни в коді до її розгортання в продуктивному середовищі;
    b) скорочення цього часу свідчить про підвищену ефективність процесів розробки та доставки;
3) частота збоїв:
    a) відсоток розгортань, що призводять до збоїв в продуктивному середовищі або вимагають відкату;
    b) зниження цього показника свідчить про підвищення стабільності та надійності розробленого програмного забезпечення;

4) час відновлення після збою:
    a) середній час, необхідний для відновлення системи після збою або виникнення проблеми.
    b) скорочення цього часу свідчить про підвищену здатність команди швидко реагувати на непередбачені ситуації та відновлювати нормальне функціонування системи.
5) Задоволеність команди та замовників:
    a) опитування та відгуки від команди розробників та замовників можуть надати інформацію про вплив впровадження DevOps на якість робочого процесу та загальне задоволення сторін.

Для проведення оцінки ефективності впровадження DevOps можна використовувати наступну методику:

1) визначення метрик та показників:
    a) потрібно визначити основні метрики, які будуть використовуватися для оцінки ефективності DevOps. Це можуть бути швидкість розгортання, час від початку до розгортання, частота збоїв, час відновлення після збою тощо;
    b) варто встановити базові показники для кожної метрики перед впровадженням DevOps, щоб мати можливість порівняти результати;
2) збір даних:
    a) потрібно використовувати інструменти моніторингу та аналітики для збору даних за обраними метриками протягом визначеного періоду;
    b) можна зберігати дані у вигляді часових рядів, щоб мати можливість спостерігати тенденції та зміни;
3) аналіз результатів:
    a) варто порівняти показники після впровадження DevOps з базовими показниками, щоб оцінити вплив змін;
    b) також потрібно проаналізувати тенденції та зміни у метриках, щоб

виявити покращення або потенційні проблеми;

4) виявлення висновків та рекомендацій:
    a) на основі аналізу можна визначити, які аспекти DevOps принесли найбільшу користь, а які потребують подальшого вдосконалення;
    b) також можна розробити рекомендації для покращення процесів DevOps на основі отриманих даних;
5) планування та впровадження змін:
    a) на основі аналізу можна скласти план дій для впровадження рекомендованих змін та покращень у процесах DevOps;
    b) також варто провести перегляд ефективності після впровадження змін, щоб оцінити їх вплив та внести корективи за необхідності.

Потрібно повторювати цикл оцінки регулярно, щоб забезпечити постійне вдосконалення процесів DevOps та адаптацію до змін у технологіях та бізнес-вимогах.

Отже, оцінка ефективності впровадження DevOps є важливим елементом для забезпечення успішної інтеграції та постійного вдосконалення процесів розробки програмного забезпечення. Аналіз метрик, таких як швидкість розгортання, частота збоїв, час відновлення після збою та задоволеність команди, дозволяє оцінити вплив DevOps на продуктивність та якість продукту. На основі отриманих результатів організації можуть розробити стратегії для покращення своїх DevOps практик, що сприятиме досягненню більшої ефективності, інноваційності та конкурентоспроможності на ринку.

4.2 Переваги та недоліки застосування DevOps

У цьому розділі ми розглянемо основні переваги та недоліки, пов'язані з впровадженням та застосуванням практик DevOps в організаціях. DevOps, як методологія, що поєднує розробку (Dev) та операції (Ops), спрямована на

підвищення швидкості доставки програмного забезпечення, поліпшення якості продукту та підвищення ефективності співпраці між командами. Однак, як і будь-який підхід, DevOps має свої виклики та недоліки, які важливо враховувати при впровадженні.

Отже, розглянемо спочатку основні переваги. Перша з них – підвищення швидкості та ефективності розгортання. Автоматизація процесів CI/CD дозволяє організаціям значно прискорити процес розгортання нових функцій та виправлень помилок. Це досягається шляхом мінімізації ручних операцій і використання автоматичних пайплайнів для збірки, тестування та розгортання коду. Швидке розгортання дозволяє командам швидше вносити зміни та адаптуватися до нових вимог, підвищуючи загальну ефективність розробки.

Також DevOps сприяє скороченню часу від ідеї до реалізації продукту, дозволяючи організаціям швидше реагувати на зміни ринку та потреби клієнтів. Швидка доставка та розгортання продуктів дозволяє компаніям оперативно впроваджувати інновації та забезпечувати конкурентну перевагу на ринку.

Неперервна інтеграція та тестування є важливими компонентами DevOps, які сприяють ранньому виявленню та виправленню помилок. Автоматизовані тести та інтеграція коду виконуються після кожного коміту, що дозволяє забезпечити високу якість коду та продукту в цілому. Це призводить до зменшення кількості помилок у продуктивному середовищі та підвищує довіру користувачів до продукту.

Автоматизація інфраструктури та використання інструментів моніторингу дозволяють DevOps командам забезпечити стабільність та надійність сервісів. Це досягається шляхом автоматичного розгортання змін в контрольованих умовах та постійного моніторингу стану системи. Це допомагає своєчасно виявляти та усувати проблеми, забезпечуючи безперебійну роботу сервісів.

Також DevOps сприяє культурі співпраці та відкритості між різними командами, зокрема розробниками, тестувальниками та операційними спеціалістами. Це досягається шляхом спільної відповідальності за якість продукту та процеси розгортання. Співпраця дозволяє краще розуміти вимоги та

цілі проекту, а також сприяє обміну знаннями та досвідом між командами, підвищуючи загальну ефективність розробки.

Тепер розглянемо основні недоліки. Перший з них – потреба в культурних змінах. Впровадження DevOps вимагає переходу від традиційних ієрархічних структур до більш гнучких та колаборативних підходів. Це може викликати опір з боку співробітників, які звикли до старих методів роботи. Для подолання цього виклику необхідно зосередитися на поступовій зміні культури, включаючи підвищення обізнаності, навчання та створення позитивних прикладів успішного застосування DevOps.

Також для ефективного використання нових інструментів та практик DevOps співробітникам може знадобитися додаткове навчання. Це стосується як технічних навичок, таких як володіння інструментами автоматизації та контейнеризації, так і м'яких навичок, таких як співпраця та управління змінами. Важливо створити програми навчання та розвитку, щоб підтримати співробітників у цьому процесі.

Інтеграція різноманітних інструментів та процесів в рамках DevOps може ускладнити управління інфраструктурою та процесами розробки. Це вимагає від команд високого рівня координації та управління конфігураціями. Важливо використовувати інструменти для оркестрації та моніторингу, щоб забезпечити ефективне управління складними середовищами.

Швидкість розгортання та автоматизація, які є основними аспектами DevOps, можуть призвести до пропусків у безпеці, якщо вони не інтегровані в процеси розробки. Важливо вбудовувати практики безпеки в усі етапи життєвого циклу розробки, включаючи автоматизоване сканування вразливостей, управління доступом та моніторинг безпеки.

Також впровадження та підтримка DevOps можуть вимагати значних інвестицій у технології, інструменти та інфраструктуру. Це може становити виклик для організацій з обмеженим бюджетом. Важливо ретельно планувати інвестиції та оцінювати вартість володіння та повернення інвестицій для DevOps ініціатив.

Загалом, попри деякі недоліки та виклики, переваги впровадження DevOps зазвичай переважають потенційні труднощі, сприяючи покращенню процесів розробки, збільшенню продуктивності та забезпеченню вищої якості продуктів.

Отже, застосування практик DevOps пропонує організаціям ряд переваг, включаючи підвищення швидкості та ефективності розгортання, покращення якості продукту та поліпшення співпраці між командами. Однак, впровадження DevOps також ставить перед організаціями певні виклики, такі як потреба в культурних змінах, необхідність навчання та розвитку навичок, управління складністю та забезпечення безпеки.

Для досягнення успішного впровадження DevOps важливо ретельно планувати та забезпечувати адекватні ресурси, а також створювати відповідну культуру та процеси, що підтримують неперервне вдосконалення та постійні інновації.

4.3 Рекомендації щодо оптимізації процесу розробки з використанням DevOps

У цьому розділі ми представимо рекомендації щодо оптимізації процесу розробки ПЗ з використанням практик DevOps. Мета цих рекомендацій полягає в тому, щоб допомогти організаціям максимально використати переваги DevOps, зокрема швидкість, ефективність, якість та безпеку. Ми розглянемо основні аспекти, такі як впровадження культури співпраці, автоматизація процесів, інфраструктура як код, моніторинг та логування, забезпечення безпеки, неперервне навчання та адаптація, а також використання метрик та зворотного зв'язку для постійного вдосконалення.

По-перше, забезпечте тісну взаємодію та співпрацю між командами розробників, тестувальників та операційних спеціалістів. Сприяйте обміну знаннями та досвідом між командами для покращення розуміння загальних

цілей та вимог проекту. Культура співпраці є основою практики DevOps і має на меті зламати традиційні бар'єри між командами розробників, тестувальників та операційних спеціалістів. Важливо створити середовище, де всі учасники процесу розробки відчувають відповідальність за загальний успіх проекту та мають можливість вільно обмінюватися знаннями та досвідом.

Для досягнення тісної взаємодії та співпраці можна використовувати наступні підходи:

1) встановіть спільні цілі для команд розробників, тестувальників та операційних спеціалістів, щоб забезпечити єдине розуміння завдань і вимог. Заохочуйте культуру взаємної відповідальності за якість продукту та ефективність процесів;
2) створіть ефективні канали комунікації, такі як щоденні стендапи, спільні зустрічі та чат-групи, для обміну знаннями та досвідом між командами. Заохочуйте відкрите обговорення проблем та ідей для покращення процесів;
3) включайте представників усіх команд у процеси планування та перегляду проекту. Це дозволить краще розуміти вимоги, очікування та обмеження, а також сприятиме координації зусиль на різних етапах розробки;
4) організуйте спільні тренінги та навчальні сесії для розвитку навичок та знань учасників команд. Це допоможе підвищити компетенції співробітників та сприятиме кращому розумінню різних аспектів процесу розробки.

Впровадження культури співпраці в рамках DevOps сприяє створенню більш гнучких, ефективних та інноваційних команд, які здатні швидко реагувати на зміни та виклики сучасного ринку програмного забезпечення.

Також автоматизуйте якомога більше процесів, включаючи збірку коду, тестування, розгортання та моніторинг. Використовуйте інструменти CI/CD для неперервної інтеграції та доставки, щоб зменшити ручні втручання та підвищити швидкість розгортання.

Автоматизація є основним елементом практик DevOps, який дозволяє значно підвищити ефективність процесу розробки та розгортання програмного забезпечення. Автоматизація процесів забезпечує консистентність та швидкість виконання завдань, зменшує ймовірність помилок, пов'язаних з ручними втручаннями, та дозволяє командам зосередитися на більш складних та творчих аспектах розробки.

Основні процеси для автоматизації включають збірку коду, тестування, розгортання та моніторинг. Автоматизація процесу збірки коду дозволяє швидко компілювати та готувати додаток до тестування та розгортання. Інструменти, такі як Maven, Gradle або Ant, можуть бути використані для автоматизації збірки.

Автоматизація тестування включає виконання юніт-тестів, інтеграційних тестів та інших видів тестування без ручного втручання. Інструменти, такі як JUnit, Selenium або TestNG, можуть бути використані для автоматизації процесу тестування.

Автоматизація розгортання дозволяє швидко та надійно виводити додаток на продуктивне середовище або середовища тестування. Інструменти, такі як Jenkins, GitLab CI/CD, або Ansible, можуть бути використані для автоматизації процесів неперервної інтеграції та доставки.

Автоматизація моніторингу дозволяє постійно відстежувати стан системи та додатків, виявляти проблеми та отримувати сповіщення про виникнення помилок або відхилень. Інструменти, такі як Prometheus, Nagios або Zabbix, можуть бути використані для автоматизації моніторингу.

Також наведемо деякі рекомендації щодо автоматизації.

1) почніть з найбільш рутинних та часозатратних завдань. Це дозволить швидко досягти результатів та звільнити час команди для більш важливих завдань;
2) використовуйте стандартизовані інструменти та практики. Це спрощує інтеграцію різних інструментів та забезпечує сумісність між різними етапами розробки;

3) поступово розширюйте автоматизацію. Не намагайтеся автоматизувати все одразу. Почніть з найважливіших процесів та поступово розширюйте обсяг автоматизації;
4) забезпечте моніторинг та логування автоматизованих процесів. Це дозволить швидко виявляти та вирішувати проблеми, які можуть виникнути під час автоматизації.

Автоматизація процесів в рамках DevOps є важливим кроком до підвищення ефективності та швидкості розробки програмного забезпечення, а також забезпечення його високої якості та надійності.

Також використовуйте підходи інфраструктури як код для автоматизації налаштування та управління інфраструктурою. Це дозволяє легко відтворювати та масштабувати інфраструктуру, а також забезпечує консистентність середовищ.

Використання IaC має багато переваг. По-перше, IaC дозволяє автоматизувати процеси створення та налаштування інфраструктури, зменшуючи ризик людських помилок та збільшуючи швидкість виконання завдань.

Використання коду для опису інфраструктури забезпечує консистентність середовищ, зменшуючи розбіжності між розробницьким, тестовим та продуктивним середовищами. Також IaC дозволяє легко відтворювати та масштабувати інфраструктуру, спрощуючи процеси розгортання та реагування на зміни у вимогах.

Інфраструктура, описана як код, може бути піддана версіонуванню та контролю змін, дозволяючи відстежувати зміни та відновлювати попередні стани при необхідності.

Рекомендації щодо використання IaC:

1) оберіть інструмент для IaC, який найкраще відповідає потребам вашої інфраструктури та команди. Terraform підходить для управління інфраструктурою в хмарних сервісах, Ansible – для конфігурації серверів, Chef і Puppet – для управління конфігураціями та

забезпечення політик безпеки;

2) зберігайте код інфраструктури в репозиторіях з контролем версій, таких як Git, для забезпечення версіонування та співпраці між командами;
3) структуруйте код інфраструктури таким чином, щоб сприяти модульності та повторному використанню компонентів. Це дозволить зменшити кількість дублювання коду та спростити управління інфраструктурою;
4) використовуйте автоматизовані тести для перевірки коректності коду інфраструктури та валідації змін перед розгортанням. Це допоможе уникнути помилок та забезпечити надійність інфраструктури.

Використання підходів інфраструктури як код сприяє ефективному управлінню інфраструктурою, забезпечує її швидке масштабування та адаптацію до змінних вимог, а також підвищує загальну якість та надійність середовищ розробки та експлуатації.

Щодо моніторингу та логування, встановіть ефективні системи моніторингу та логування для відстеження стану інфраструктури та додатків. Це дозволить швидко виявляти та усувати проблеми, а також забезпечить прозорість процесів.

Ефективні системи моніторингу та логування є невід'ємною частиною DevOps, оскільки вони дозволяють відстежувати стан інфраструктури та додатків, виявляти проблеми та забезпечувати прозорість процесів.

Моніторинг включає збір метрик та показників про стан системи та її компонентів, таких як сервери, бази даних, мережеві елементи тощо. Це дозволяє відстежувати роботу інфраструктури в реальному часі та отримувати сповіщення про відхилення від нормального стану.

Інструменти, такі як Prometheus, Nagios, Zabbix, або Datadog, дозволяють збирати метрики про роботу системи та візуалізувати їх у вигляді графіків та дашбордів. Налаштуйте сповіщення про критичні події або показники, які перевищують порогові значення. Це дозволить оперативно реагувати на

проблеми та забезпечити стабільність системи. Також використовуйте інструменти APM, такі як New Relic або Dynatrace, для моніторингу продуктивності додатків, відстеження запитів, аналізу помилок та оптимізації роботи програм.

Логування включає збір та зберігання записів про події, які відбуваються в системі та її компонентах. Це дозволяє аналізувати історію роботи системи, виявляти причини помилок та вдосконалювати процеси.

Використовуйте системи централізованого логування, такі як Elasticsearch, Logstash, Kibana (ELK Stack) або Graylog, для збору та аналізу логів з різних джерел. Забезпечте структурування логів для полегшення їх аналізу та пошуку. Використовуйте стандартизовані формати, такі як JSON, для зберігання логів.

Також налаштуйте ротацію логів для уникнення переповнення дискового простору та забезпечте архівацію старих логів для можливості їх аналізу в майбутньому.

Ефективне використання систем моніторингу та логування дозволяє забезпечити високу доступність та надійність сервісів, своєчасно виявляти та усувати проблеми, а також підвищити загальну якість процесів розробки та експлуатації програмного забезпечення.

Також інтегруйте практики безпеки на всіх етапах розробки та розгортання. Використовуйте автоматизоване сканування вразливостей, управління доступом та шифрування для захисту даних та інфраструктури. Інтеграція практик безпеки на всіх етапах розробки та розгортання є важливою для забезпечення безпеки даних та інфраструктури. Це включає використання різних інструментів та методик для захисту від потенційних загроз та вразливостей.

Використовуйте інструменти, такі як OWASP ZAP, Nessus, або Qualys, для автоматизованого сканування вразливостей у коді додатків та інфраструктурі. Це дозволить своєчасно виявляти потенційні слабкі місця. Інтегруйте процес сканування вразливостей у пайплайни неперервної інтеграції та доставки, щоб автоматично перевіряти безпеку коду перед його розгортанням.

Забезпечте надійні механізми ідентифікації та аутентифікації користувачів та сервісів. Використовуйте сильні паролі, багатофакторну аутентифікацію та сертифікати для забезпечення безпечного доступу. Встановіть політики контролю доступу, які визначають, хто та як може взаємодіяти з ресурсами та даними. Використовуйте принцип найменших привілеїв для обмеження доступу до мінімуму, необхідного для виконання завдань.

Застосовуйте шифрування для захисту чутливих даних як у стані спокою, так і під час передачі. Використовуйте сучасні алгоритми та протоколи шифрування, такі як AES, TLS/SSL, для забезпечення конфіденційності та цілісності даних. Впровадіть надійні механізми управління криптографічними ключами, включаючи їх зберігання, ротацію та відновлення, для забезпечення безпеки шифрування.

Рекомендації щодо забезпечення безпеки:

1) включайте практики безпеки на ранніх етапах розробки, щоб ідентифікувати та вирішувати потенційні ризики до розгортання продукту;
2) забезпечте регулярне навчання співробітників з питань кібербезпеки, щоб підвищити їх обізнаність про загрози та кращі практики захисту;
3) розробіть план реагування на інциденти, який визначає процедури виявлення, аналізу та усунення безпекових інцидентів, а також відновлення системи після атаки.

Загалом, інтеграція практик безпеки на всіх етапах розробки та розгортання є важливою для забезпечення захисту інформаційних ресурсів та підтримання довіри користувачів та партнерів.

Наступна рекомендація – створюйте культуру неперервного навчання та вдосконалення. Заохочуйте співробітників до освоєння нових інструментів та методологій, а також адаптуйтеся до змін у технологіях та ринкових умовах.

Неперервне навчання та адаптація: Створення культури неперервного навчання та вдосконалення є важливим аспектом DevOps, оскільки воно дозволяє

організаціям швидко адаптуватися до постійно змінюваних технологічних та ринкових умов.

Основні аспекти неперервного навчання та адаптації:

1) створіть середовище, де навчання та професійний розвиток співробітників є пріоритетом. Забезпечте доступ до ресурсів для навчання, таких як онлайн- курси, вебінари, конференції та тренінги;
2) заохочуйте співробітників досліджувати та впроваджувати нові інструменти та практики, які можуть підвищити ефективність розробки та операцій. Це може включати вивчення нових мов програмування, фреймворків, інструментів автоматизації та методологій управління проектами;
3) підтримуйте гнучкість та відкритість до змін, щоб швидко реагувати на нові технологічні тренди та зміни на ринку. Це включає регулярний перегляд та оновлення процесів, інструментів та практик у відповідності до поточних потреб;
4) сприяйте культурі спільного навчання, де співробітники можуть обмінюватися знаннями та досвідом. Організовуйте внутрішні зустрічі, технічні сесії та "hackathons", де команди можуть експериментувати з новими ідеями та рішеннями;
5) виділіть ресурси, такі як час та бюджет, для підтримки навчальних ініціатив та професійного розвитку співробітників. Це може включати оплату курсів, участь у конференціях та надання часу на самоосвіту.

Створення культури неперервного навчання та адаптації дозволяє організаціям підтримувати високий рівень інноваційності та конкурентоспроможності, підвищувати кваліфікацію співробітників та ефективно адаптуватися до змін у технологіях та бізнес-вимогах.

Ще одна рекомендація – використовуйте метрики для оцінки ефективності процесів та продукту. Збирайте зворотний зв'язок від користувачів та команди для постійного вдосконалення продукту та процесів.

Метрики та зворотний зв'язок: Використання метрик та зворотного зв'язку є

важливим елементом DevOps, оскільки вони дозволяють оцінити ефективність процесів та продукту, а також виявити області для подальшого вдосконалення.

Встановіть конкретні метрики, які відображають цілі вашої організації та проекту. Це можуть бути показники, такі як час від початку розробки до розгортання (Lead Time), частота розгортання (Deployment Frequency), час відновлення після збою (Mean Time to Recovery) та інші.

Також використовуйте інструменти моніторингу та аналітики для збору даних за обраними метриками. Аналізуйте ці дані для виявлення тенденцій, проблем та можливостей для оптимізації. Створіть інформативні дашборди, які відображають основні метрики в реальному часі. Це дозволить командам швидко отримувати необхідну інформацію та реагувати на зміни.

Стосовно зворотного зв'язку від користувачів, використовуйте опитування, інтерв'ю, аналітику користувачів та інші методи для збору відгуків від кінцевих користувачів. Це дозволить зрозуміти їхні потреби, задоволеність продуктом та області для покращення.

Заохочуйте співробітників до відкритого обговорення та надання відгуків щодо процесів, інструментів та практик. Регулярно проводьте ретроспективи для аналізу успіхів та недоліків завершених проектів. Використовуйте зібрані метрики та зворотний зв'язок для постійного вдосконалення процесів та продукту. Вносьте зміни та спостерігайте за їхнім впливом на загальну ефективність.

Застосування метрик та зворотного зв'язку в рамках DevOps сприяє створенню культури постійного вдосконалення, де команди можуть ефективно адаптуватися до змін, оптимізувати робочі процеси та підвищити якість продукту.

Впровадження цих рекомендацій допоможе оптимізувати процес розробки програмного забезпечення, підвищити ефективність команди та забезпечити високу якість та безпеку продуктів.

Отже, впровадження рекомендацій щодо оптимізації процесу розробки з використанням DevOps може значно підвищити ефективність, швидкість та

якість розробки програмного забезпечення. Це вимагає від організацій створення культури співпраці, автоматизації основних процесів, впровадження надійних практик безпеки, а також забезпечення неперервного навчання та адаптації. Використання метрик та збір зворотного зв'язку дозволяють постійно вдосконалювати процеси та продукти. Впровадження цих рекомендацій сприятиме досягненню головної мети DevOps – створення високоефективної та інноваційної розробки програмного забезпечення, здатної швидко адаптуватися до змінних потреб бізнесу та ринку.

4.4 Висновки до 4 розділу

У цьому розділі ми провели оцінку ефективності впровадження практик DevOps, розглянули переваги та недоліки їх застосування та надали рекомендації щодо оптимізації процесу розробки.

Аналіз показав, що впровадження DevOps сприяє підвищенню швидкості розгортання, поліпшенню якості продукту та покращенню співпраці між командами. Однак, для досягнення максимальної ефективності необхідно забезпечити належне планування, підтримку змін в культурі організації та навчання співробітників.

Переваги включають швидкість, гнучкість, покращення якості та стабільності продукту. Однак, організації можуть зіткнутися з викликами, такими як необхідність культурних змін, потреба в навчанні та розвитку навичок, управління складністю та забезпечення безпеки.

Рекомендації включають впровадження культури співпраці, автоматизацію основних процесів, застосування інфраструктури як код, встановлення ефективних систем моніторингу та логування, забезпечення безпеки на всіх етапах, створення культури неперервного навчання та використання метрик та зворотного зв'язку для постійного вдосконалення.

У підсумку, впровадження практик DevOps може значно покращити процес розробки програмного забезпечення, проте вимагає комплексного підходу, уваги до деталей та відданості постійному вдосконаленню.

5 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

5.1 . Критичні стани людини.

До критичних станів належать непритомність, колапс, шок, стенокардія, гіпертонічний криз.

Непритомність (зомління) – найпоширеніший і легкий прояв гострої судинної недостатності через раптове короткочасне зниження кровопостачання головного мозку. Виникає при крововтратах, травмах, хвилюванні, духоті, голодуванні, перевтомленні, захворюваннях серцево-судинної системи.

Основні симптоми:

1) нерухомий стан;
2) втрата свідомості, відсутність контакту;
3) часте дихання і пульс;
4) блідість шкіри, холодні кінцівки, закочування очей.

Перша допомога:

1) покласти горизонтально, забезпечити доступ свіжого повітря;
2) звільнити від тісного одягу, підняти ноги вище тулуба;
3) сприскати холодною водою, дати понюхати ватку з нашатирним спиртом;
4) напоїти гарячим чаєм або кавою, розтерти руки й ноги, дати грілку.

Колапс – більш важка форма судинної недостатності, яка виникає при кровотечах, травмах, інтоксикаціях, інфаркті міокарду.

Основні симптоми:

1) в’ялість, загальмованість, затьмарена свідомість, неохоче вступає в контакт;
2) часте дихання, слабкий ниткоподібний пульс, низький артеріальний тиск;
3) різка блідість шкіри і слизових оболонок, холодний липкий піт, запалі очі, загострені риси обличчя, кінцівки холодні, можлива блювота.

Перша допомога:

1) покласти горизонтально з дещо опущеною головою;
2) обкласти нижні кінцівки грілками, накрити ковдрою;
3) забезпечити доступ свіжого повітря, припинити дію зовнішніх подразників;
4) зупинити кровотечу (якщо є), іммобілізувати переломи

Шок – комплекс симптомів, що супроводжуються порушенням нервової регуляції життєво важливих функцій органів і систем. Особливо страждає центральна нервова система (ЦНС). Травматичний шок має дві фази: еретильну і торпідну.

Ознаки еретильної фази:

1) збудливість, рухливість, ясна свідомість, нав'язливий контакт;
2) дещо часте дихання, напружені м'язи, пітливість, частий пульс, підвищений тиск.

Ознаки торпідної фази:

1) в'ялість, загальмованість, затьмарена свідомість, неохочий контакт;
2) часте дихання, виражена блідість, низький тиск, ниткоподібний пульс, холодний липкий піт, загострені риси обличчя.

Перша допомога:

1) усунути джерело патологічної дії;
2) покласти, забезпечити прохідність дихальних шляхів, зупинити кровотечу, ввести знеболювальне, іммобілізувати травмовані кінцівки.

Стенокардія (грудна жаба) – гостра ішемія міокарду через погіршення його кровопостачання з наступним відновленням кровообігу в зоні ішемії.

Основні симптоми:

1) періодичний стискаючий біль за грудиною (від 1 до 30 хвилин) з іррадіацією в ліву руку, під лопатку, плече, шию, нижню щелепу;
2) блідість, слабкість, страх, пітливість, тремтіння тіла, прискорений пульс.

Перша допомога:

1) заспокоїти, забезпечити доступ свіжого повітря, сидяче положення;

2) валідол або нітрогліцерин під язик, серцеві краплі;
3) на груди – гірчичники, для рук і ніг – гарячу ванну.

Інфаркт – результат гострої стійкої тривалої ішемії міокарду.

Основні симптоми:

1) постійний стискаючий біль за грудиною (від 30 хвилин до кількох діб) з іррадіацією в ліву руку, під лопатку, плече, шию, щелепу;
2) валідол і нітропрепарати не знімають біль, стан важкий, страх, блідість шкіри з ціанотичним відтінком, пітливість, ниткоподібний пульс, слабкий тиск, можливе затьмарення свідомості, розлади мови.

Перша допомога: Забезпечити повний спокій, доступ свіжого повітря, викликати кардіологічну бригаду.

Гіпертонічний криз – загострення гіпертонічної хвороби через тривале хвилювання і нервове перенавантаження, що призводить до підвищення артеріального тиску до 170/95 - 200/120 мм рт. ст.

Основні симптоми: Різкий головний біль (особливо в потилиці), запаморочення, шум у вухах, нудота, блювота, носові кровотечі, слабкість, тремтіння рук і ніг, пітливість, невизначений біль у серці.

Перша допомога: Хворого покласти, забезпечити повний фізичний і психічний спокій, доступ свіжого повітря, масаж шиї і потиличної ділянки, гірчичники на шию, потилицю і литкові м’язи, холодний компрес до голови, транспортування напівсидячи.

Гострий живіт – клінічний симптомокомплекс при пошкодженнях і гострих захворюваннях органів черевної порожнини та заочеревного простору.

Основні причини:

1) пошкодження органів черевної порожнини і заочеревного простору (закриті і відкриті);
2) запальні процеси в червоподібному відростку, жовчному міхурі, підшлунковій залозі, перфорація шлунку, дванадцятипалої кишки, дивертикули товстого і тонкого кишківника;

3) внутрішні кровотечі в шлунково-кишковий тракт, позаматкова вагітність;
4) непрохідність кишківника (заворот, защемлення, стискання спайками).

Основні симптоми: Біль локалізований або по всьому животу, шок, блювота, ікота, сухий язик, пронос з кров’ю, блідість шкірних покривів, частий пульс.

Перша допомога: Транспортування в лікарню дуже обережно, на ношах, в положенні лежачи на спині.

Гострий перитоніт – запалення черевної порожнини з важкими загальними симптомами і порушенням функцій життєво важливих органів і систем. Виникає через проникнення мікробів у черевну порожнину.

Перша допомога:

1) транспортування на жорстких ношах або дерев’яному щиті, на спині, в положенні «жаби»;
2) згинають ноги в колінних суглобах і злегка розводять, під коліна підкладають валик або згорнутий одяг.

Ці стани потребують негайного реагування та першої допомоги, оскільки вони загрожують життю і здоров'ю людини. Основні симптоми варіюються від втрати свідомості та гострого болю до порушень серцевої діяльності і дихання. Перша допомога часто включає забезпечення горизонтального положення, доступу свіжого повітря, звільнення від тісного одягу, обігрів та зупинку кровотечі. Швидке і правильне реагування на такі стани може врятувати життя і мінімізувати наслідки.

5.2 Економічне значення заходів щодо покращення умов та охорони праці.

Заходи щодо покращення умов та охорони праці мають суттєве економічне значення як для окремих підприємств, так і для економіки в цілому.

В першу чергу, такі заходи дозволяють знизити витрати на медичне обслуговування та компенсації, оскільки підприємства витрачають менше коштів на лікування травмованих працівників та виплати по інвалідності. Це сприяє зниженню загальних витрат компанії.

Крім того, безпечні та здорові умови праці підвищують продуктивність праці, оскільки працівники мають менше лікарняних днів. Задоволеність працівників умовами праці знижує рівень плинності кадрів, що дозволяє підприємствам зекономити на витратах, пов'язаних з набором і навчанням нових працівників. Зменшення кількості нещасних випадків на виробництві веде до зниження страхових внесків, що сплачує компанія, а також до зниження ризиків, пов'язаних з судовими позовами.

Компанії, що дбають про охорону праці, мають кращу репутацію, що може сприяти збільшенню кількості клієнтів та інвесторів, а також залученню кваліфікованих працівників. Інвестиції в покращення умов праці можуть стимулювати впровадження нових технологій та методів роботи, що підвищує конкурентоспроможність підприємства на ринку. Безпечні умови праці зменшують кількість інцидентів, що можуть призвести до простоїв виробництва, таким чином забезпечуючи стабільний виробничий процес.

Інвестиції в охорону праці мають довгостроковий позитивний вплив на економіку підприємства та суспільства, сприяючи сталому розвитку та підвищенню економічної стабільності. Заходи щодо покращення умов та охорони праці мають значний економічний вплив, сприяючи зниженню витрат, підвищенню продуктивності, покращенню репутації підприємства та довгостроковій економічній стабільності.

ВИСНОВКИ

У рамках цього дослідження ми розглянули теоретичні основи DevOps, провели аналіз існуючих методів оптимізації процесу розробки програмного забезпечення, демонстрували практичне застосування методів DevOps та розробили рекомендації для їх оптимізації. Ми виявили, що впровадження практик DevOps сприяє підвищенню ефективності, швидкості та якості розробки програмного забезпечення, а також покращує співпрацю між командами та забезпечує більш стабільну та безпечну роботу продуктів.

Основні завдання були успішно виконані в ході дослідження, включали огляд теоретичних основ DevOps, оцінку ефективності впровадження DevOps, аналіз переваг та недоліків їх застосування, розробку практичних прикладів застосування методів DevOps для автоматизації процесів, моніторингу та логування, а також впровадження практик безпеки, та розробку рекомендацій щодо оптимізації процесу розробки з використанням DevOps.

Перспективи подальших досліджень у цій області включають розширення застосування DevOps у різних галузях та типах проектів, включаючи великі корпорації та стартапи, аналіз інтеграції DevOps з іншими практиками, такими як Agile, Lean та ITIL, для досягнення ще більшої ефективності, вивчення нових інструментів та методик автоматизації для подальшої оптимізації процесів розробки та експлуатації програмного забезпечення, розробка та аналіз підходів до інтеграції практик безпеки у процеси DevOps для створення більш безпечних програмних продуктів та розробка методик вимірювання та оцінки ефективності впровадження DevOps у різних організаційних та проектних контекстах.

У підсумку, дослідження підтвердило, що впровадження DevOps є стратегічно важливим кроком для організацій, які прагнуть оптимізувати свої процеси розробки програмного забезпечення. Рекомендації, надані в цій роботі, слугують практичним посібником для досягнення вищої ефективності, гнучкості та безпеки у розробці. Однак, для успішного застосування DevOps необхідно

враховувати не тільки технологічні аспекти, а й культурні зміни в організації. Подальші дослідження в цій галузі можуть сприяти розробці нових методів та інструментів для покращення процесів розробки та експлуатації програмного забезпечення, а також допомогти організаціям ефективно адаптуватися до швидко змінюваного технологічного ландшафту.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ramtin J. What is DevOps? [Електронний ресурс] / J. Ramtin, B. A. Nauman. – 2016. – Режим доступу до ресурсу: https://www.researchgate.net/publication/308857081_What_is_DevOps_A_Systematic_

2. Michael H. DEVOPS FOR DEVELOPERS [Електронний ресурс] / Huttermann Michael. – 2012. – Режим доступу до ресурсу: http://huettermann.net/devops/.

3. Len B. DevOps: A Software Architect\'s Perspective [Електронний ресурс] / B. Len, W. Ingo. – 2015. – Режим доступу до ресурсу: https://books.google.com.ua/books/about/DevOps.html?id=fcwkCQAAQBAJ&redir_esc=y.

4. DevOps: Concepts, practices, tools, benefits and challenges [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://opus.lib.uts.edu.au/rest/bitstreams/f161f4aa-8a29-4240-bbba-900887cb56aa/retrieve

5. Liming Z. DevOps and Its Practices [Електронний ресурс] / Zhu Liming. – 2016. – Режим доступу до ресурсу: https://www.researchgate.net/publication/301639252_DevOps_and_Its_Practices.

6. Gene K. The DevOps Handbook [Електронний ресурс] / Kim Gene. – 2016. – Режим доступу до ресурсу: https://books.google.com.ua/books/about/The_DevOps_Handbook.html?id=ui8hDgAAQBAJ&redir_esc=y.

7. Akshaya H. L. A Basic Introduction to DevOps Tools [Електронний ресурс] / Akshaya. – 2015. – Режим доступу до ресурсу: https://www.ijcsit.com/docs/Volume%206/vol6issue03/ijcsit2015060382.pdf.

8. Welder L. Building a Collaborative Culture: A Grounded Theory of Well Succeeded DevOps Adoption in Practice [Електронний ресурс] / L. Welder, P. Gustavo. – 2018. – Режим доступу до ресурсу: https://www.researchgate.net/publication/327687778_Building_a_Collaborative_Culture_A_Grounded_Theory_of_Well_Succeeded_DevOps_Adoption_in_Practice.

9. Azure DevOps Azure DevOps integration [Електронний ресурс] – Режим доступу до ресурсу: https://www.getint.io/integrations/azure-devops-azure-devops-

integration?gad_source=1&gclid=CjwKCAjw-O6zBhASEiwAOHeGxTUz_apg9q2fhHoTRnGWo99yLCaxN5mOPC07M_Ok_fX10YDRPGrBlBoClWUQAvD_BwE.

10. Thomas S. Agile Software Development: Best Practices for Large Software Development Projects [Електронний ресурс] / Stober Thomas. – 2010. – Режим доступу до ресурсу: https://www.researchgate.net/publication/293305412_Agile_software_development_Best_practices_for_large_software_development_projects.

11. Безпека життєдіяльності, основи охорони праці [Електронний ресурс] – Режим доступу до ресурсу: https://dl.tntu.edu.ua/content.php?cid=299141.

12. Постанова ВР № 2695-XII від 14.10.92 [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12#Text

ДОДАТКИ

ДОДАТОК А

Лістинги коду

Лістинг коду 1 – Приклад автоматизації неперервної інтеграції з Jenkins

```
groovy pipeline {
agent any stages {
stage('Build') { steps {
echo 'Building...'
// Команди для збірки проекту
}
}
stage('Test') { steps {
echo 'Testing...'
// Команди для запуску тестів
}
}
stage('Deploy') { steps {
echo 'Deploying...'
// Команди для розгортання проекту
}
}
}
```

Лістинг коду 2 – Приклад конфігурації інфраструктури як код з Terraform

```
hcl
resource "aws_instance" "example" {
ami             = "ami-
0c55b159cbfafe1f0" instance_type =
"t2.micro"

tags = {
Name = "ExampleInstance"
}
}
```

Лістинг коду 3 – Приклад автоматизації розгортання з Docker та Docker Compose

```
yaml version: '3' services:
web:
image: "my-web-app:latest" ports: - "80:80"
db:
image: "mysql:5.7" environment:
MYSQL_ROOT_PASSWORD: "example"
```

Лістинг коду 3.4 – Приклад моніторингу за допомогою Prometheus

```
yaml
# Приклад конфігурації для Prometheus global:
scrape_interval: 15s

scrape_configs:
- job_name: 'my-application' static_configs:
- targets: ['localhost:8080']
```

Лістинг коду 3.5 – Приклад логування з Elastic Stack

```
json
{
"message": "Приклад логування", "level": "INFO",
"timestamp": "2021-01-01T12:00:00Z"
}
```

Лістинг коду 4 – Приклад використання Grafana для візуалізації метрик

```
ini
# Приклад конфігурації панелі в Grafana [dashboard]
title = "Моя панель" uid = "my-dashboard"
tags = ["performance", "metrics"] timezone = "browser"

# Визначення віджета для відображення часового ряду [widget "cpu_usage"]
type = "graph" datasource = "Prometheus"
title = "Використання CPU" span = 6

# Конфігурація запиту до джерела даних [widget "cpu_usage".series]
name = "CPU Usage"
query = "rate(process_cpu_seconds_total{job='my-app'}[5m])"
```

Лістинг коду 5 – Приклад використання OWASP ZAP

```
bash
# Запуск сканування вразливостей для веб-додатку
zap-cli quick-scan --self-contained --start-options '-config api.disablekey=true' http://example.com
```

Лістинг коду 6 – Приклад використання Git Hooks

```
bash #!/bin/sh
# pre-commit hook для перевірки коду
# Перевірка коду за допомогою лінтера lint_result=$(eslint .)
if [ "$lint_result" != "" ]; then
echo "Помилки лінтера виявлені, коміт скасовано:" echo "$lint_result"
exit 1
fi
exit 0
yaml
```

Лістинг коду 7 – Приклад використання Snyk

```
# Приклад конфігурації GitHub Actions для інтеграції Snyk name: Security Test on: [push] jobs:
snyk:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v2
- name: Run Snyk to check for vulnerabilities uses: snyk/actions/node@master
env:
SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
```

ДОДАТОК Б

Диск із кваліфікаційною роботою бакалавра