DevOps - introduction

Порядок денний

1. Чому DevOps?
2. Принципи DevOps
3. Концепції DevOps
4. Практики DevOps
5. Люди у DevOps
6. Контроль у DevOps
7. Висновок

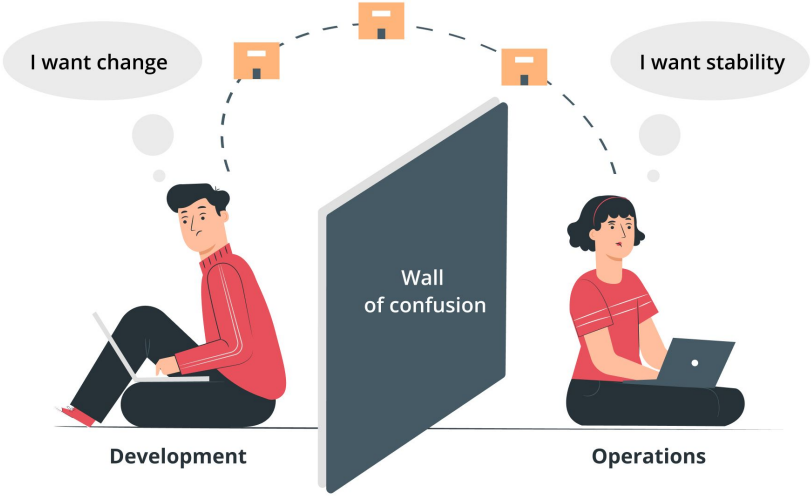
Чому Девопс?

Щоб скоротити час розробки до хвилин!

- Бізнес вимагає швидкого та безперервного постачання.
- Більшість організацій не здатні впроваджувати зміни у продуктивне середовище за хвилини чи години; натомість це займає тижні або місяці. Протилежні цілі між командами розробки (Development) та експлуатації (Operations):
- Конфлікт між гнучкою розробкою (термінові проєкти) та стабільною експлуатацією (підтримка працездатності, уникнення втручань у середовище).

Типові проблеми між командами розробки (Dev) та експлуатації (Ops):

- Відсутність спільних цілей
- Різні менталітети
- Різні інструменти
- Різні середовища
- Накопичення завдань у беклозі (product backlogs)
- "Гра у звинувачення"
- Роз'єднані процеси
- Низька якість зворотного зв’язку

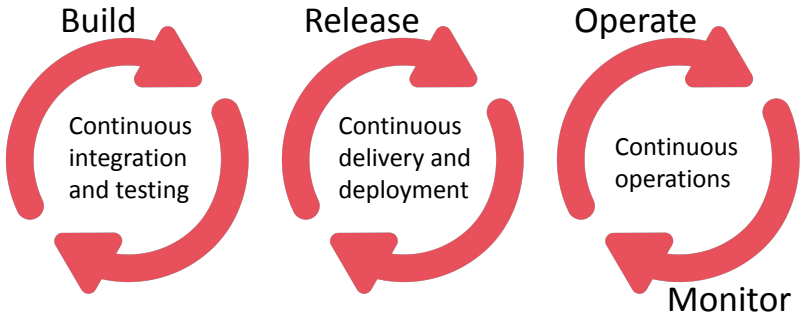


Призводить до негативної спіралі: Усі стають більш зайнятими, робота вимагає більше часу, обмін інформацією уповільнюється, черга завдань стає довшою. Збільшується кількість погоджувальних процесів, що уповільнює виконання завдань і впровадження змін.

Що таке DevOps?

Зруйнуйте стіну між розробкою та операціями:

Безперервна доставка, безперервна інтеграція, тестування та розгортання



«Невеликі команди самостійно впроваджують свої функції, перевіряють їх правильність у середовищах, схожих на виробничі, та швидко, безпечно й надійно розгортають свій код у виробниче середовище»

The DevOps Handbook, by Gene Kim, Jez Humble, Patrick Debois and John Willis, 2016

Що таке DevOps?

«Безперервна доставка — це про передачу графіка випуску в руки бізнесу, а не в руки IT. Впровадження Безперервної доставки означає, що ваше програмне забезпечення завжди готове до розгортання на виробництві протягом усього його життєвого циклу — будь-яка збірка може бути випущена для користувачів натисканням однієї кнопки за допомогою повністю автоматизованого процесу, який займає секунди або хвилини.Це, у свою чергу, залежить від повної автоматизації процесів створення збірки, тестування та розгортання, а також від чудової співпраці між усіма, хто бере участь у доставці — розробниками, тестувальниками, адміністраторами баз даних, системними адміністраторами, користувачами та бізнесом.У світі Безперервної доставки розробники не завершують роботу над функцією, коли передають код тестувальникам або коли функція проходить перевірку якості («QA passed»). Вони завершують її тоді, коли вона працює у виробничому середовищі. Це означає, що більше немає окремих фаз тестування або розгортання, навіть у межах спринту (якщо ви використовуєте Scrum).»

Continuous Delivery, Reliable Software Releases through Build, Test, and Deployment Automation, by Jez Humble and David Farley, 2010

Коротка історія DevOps

- DevOps спирається на перевірені часом концепції, такі як Lean (ощадливе виробництво), управління IT-сервісами (ITSM), гнучка розробка (Agile), теорія обмежень, інженерія стійкості, організації, що навчаються, тощо.
- На конференції Agile у 2008 році в Торонто, Канада, Патрік Дебуа та Ендрю Шафер провели сесію формату “birds of a feather”, присвячену застосуванню принципів Agile до інфраструктури, а не лише до коду застосунків.
- Пізніше, на конференції Velocity у 2009 році, Джон Оллспо та Пол Хаммонд представили знакову доповідь "10 розгортань на день: співпраця розробників і операторів у Flickr".
- Патрік Дебуа не був присутній на конференції, але був настільки вражений ідеями Оллспо та Хаммонда, що організував перший DevOps Days у Генті, Бельгія, в 2009 році, де вперше було введено термін «DevOps».
- У 2010 році Джез Хамбл і Девід Фарлі у своїй книзі *Continuous Delivery* (Безперервна доставка) розширили концепцію до безперервної доставки, яка визначила роль «конвеєра розгортання» для забезпечення постійної готовності коду та інфраструктури до розгортання, а також можливості безпечного розгортання будь-якого коду, що інтегрується до основної гілки.
- У 2013 році Джин Кім та співавтори опублікували роман *Проєкт Фенікс: Роман про IT, DevOps і успіх вашого бізнесу*, який зробив DevOps загальновідомим у галузі.
- У 2016 році Джин Кім, Джез Хамбл, Патрік Дебуа та Джон Вілліс опублікували книгу *The DevOps Handbook* (Довідник DevOps).

Принципи, Концепції, Практики та Люди

ПРИНЦИПИ

1. Потік
2. Зворотній зв’язок
3. Постійне навчання та вдосконалення

КОНЦЕПЦІЇ

1. Продукт
2. Потік створення цінності
3. Слабо пов’язана архітектура
4. Автономні команди

ПРАКТИКИ

1. Безперервна інтеграція
2. Швидке та надійне автоматизоване тестування
3. Безперервне та автоматизоване розгортання/підготовка середовищ
4. Вимірювання та зворотний зв’язок

ЛЮДИ

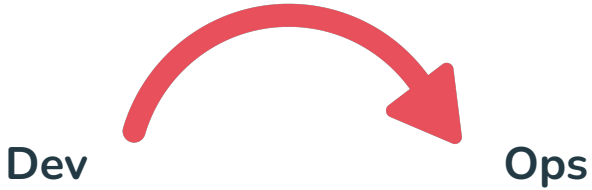
1. Орієнтація на клієнта
2. Відповідальність від початку до кінця
3. Співпраця
4. Навчання та вдосконалення
5. Експерименти та прийняття ризиків

Принципи DevOps

Три принципи DevOps

1. Потік

Забезпечення швидкого потоку роботи зліва направо: від розробки до операцій і до клієнта



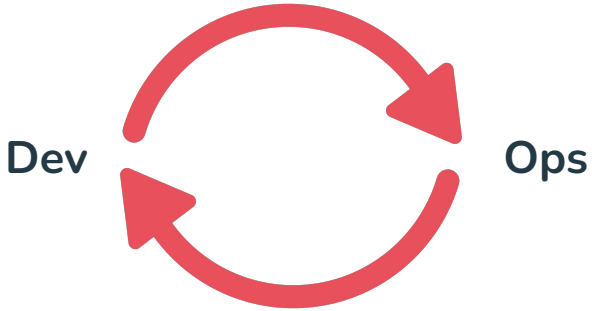
- Зробити роботу видимою за допомогою візуальних дошок
- Обмеження обсягу роботи, що виконується одночасно (WIP)
- Зменшення розмірів партій
- Зменшення кількості передач роботи між командами
- Постійне виявлення та усунення обмежень
- Усунення труднощів і втрат у потоці створення цінності

Три принципи DevOps

2. Зворотній зв’язок

Забезпечення швидкого та постійного потоку зворотного зв’язку на всіх етапах потоку створення цінності

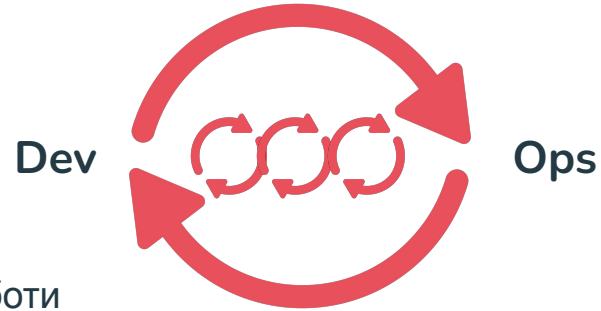
- Встановлення швидких циклів зворотного зв’язку на кожному етапі процесу
- Запровадження постійної телеметрії виробництва
- Наближення якості до джерела
- Оптимізація для центрів обробки роботи



Три принципи DevOps

3. Постійне навчання та вдосконалення

Створення культури довіри, експериментів, прийняття ризиків та організаційного навчання



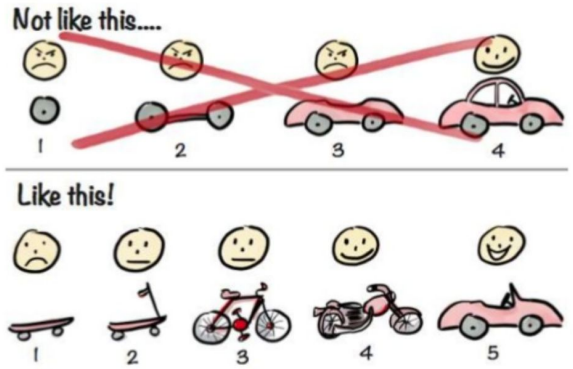
- Забезпечення організаційного навчання
- Інституціоналізація покращення щоденної роботи
- Перетворення локальних відкриттів на глобальні покращення
- Впровадження патернів стійкості у щоденну роботу
- Заохочення лідерів до підтримки культури навчання
- Експерименти, провали - *якщо це складно, роби це частіше*

Концепція DevOps

Продукт

Наша програма — це наш продукт: Прагнемо доставити мінімальний життєздатний продукт (MVP), який має мінімальний обсяг функціональності з цінністю для клієнта, якомога швидше і надійніше

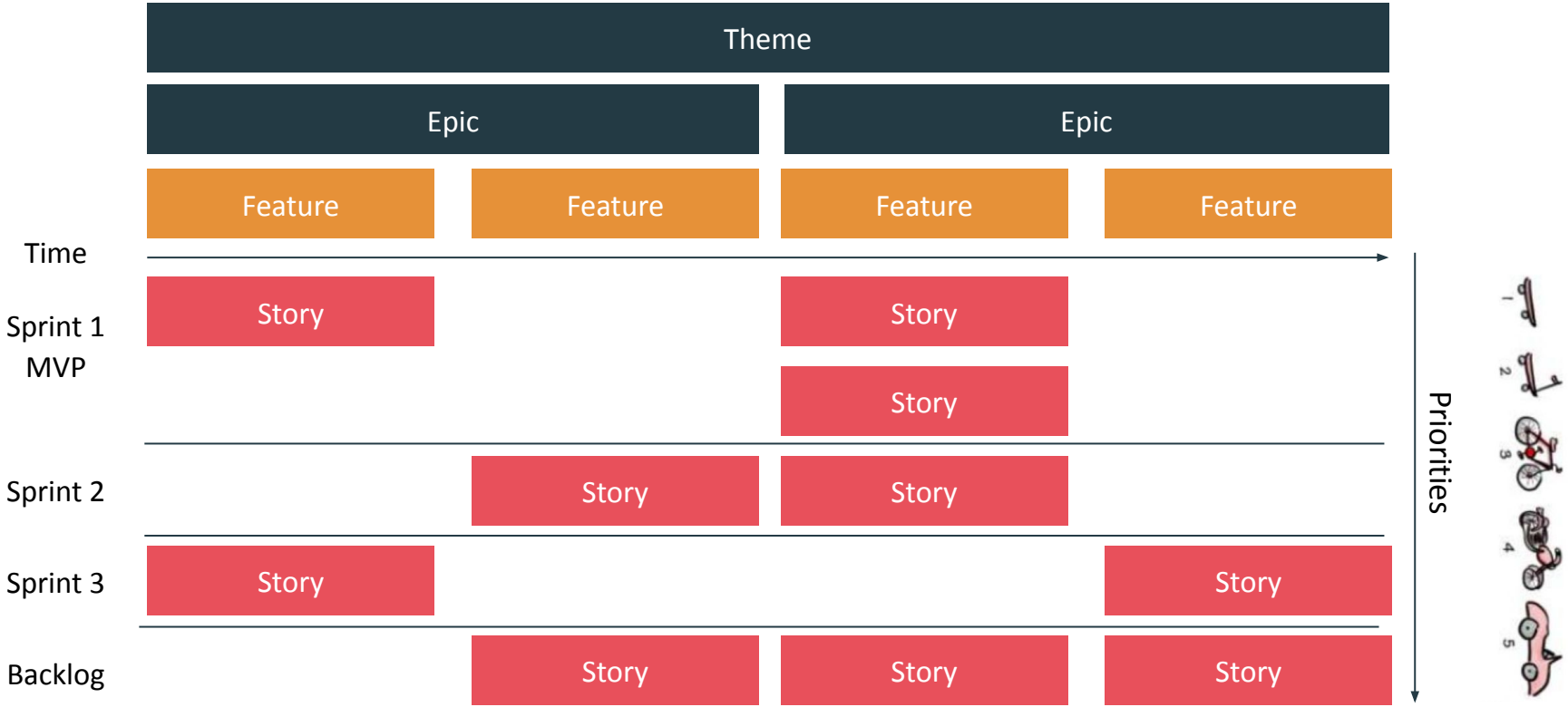
MVP — це найпростіша версія продукту, яка містить лише основні функції, необхідні для задоволення потреб клієнта. Використовується для тестування нових ідей або концепцій на ринку.



Тобто: MVP — це інструмент для швидкого тестування ідей, підтвердження припущень і отримання зворотного зв’язку з мінімальними витратами та ризиками.

The Lean Startup, by Eric Ries, 2011

Продукт


Theme
Epic
Epic
Feature
Feature
Feature
Feature
Time
Sprint 1
MVP
Story
Story
Story
Sprint 2
Story
Story
Sprint 3
Story
Story
Backlog
Story
Story
Story
Priorities

Продукт

- Створення з урахуванням кінцевої мети: Розуміння реальних потреб клієнтів
- Перед побудуванням функції запитуємо себе, “Що ми будуємо і навіщо?”
- Виконання найдешевших та найшвидших експериментів для перевірки функції
- Розглядати кожну функцію як гіпотезу, яку потрібно перевірити
- Використовуватитакі техніки перевірки гіпотез, як розробка, керована гіпотезами, воронки залучення клієнтів, та A/B тестування.

50% of visitors see variation A

50% of visitors see variation B

27% conversion

23 EUR average order size

13% conversion

27 EUR average order size

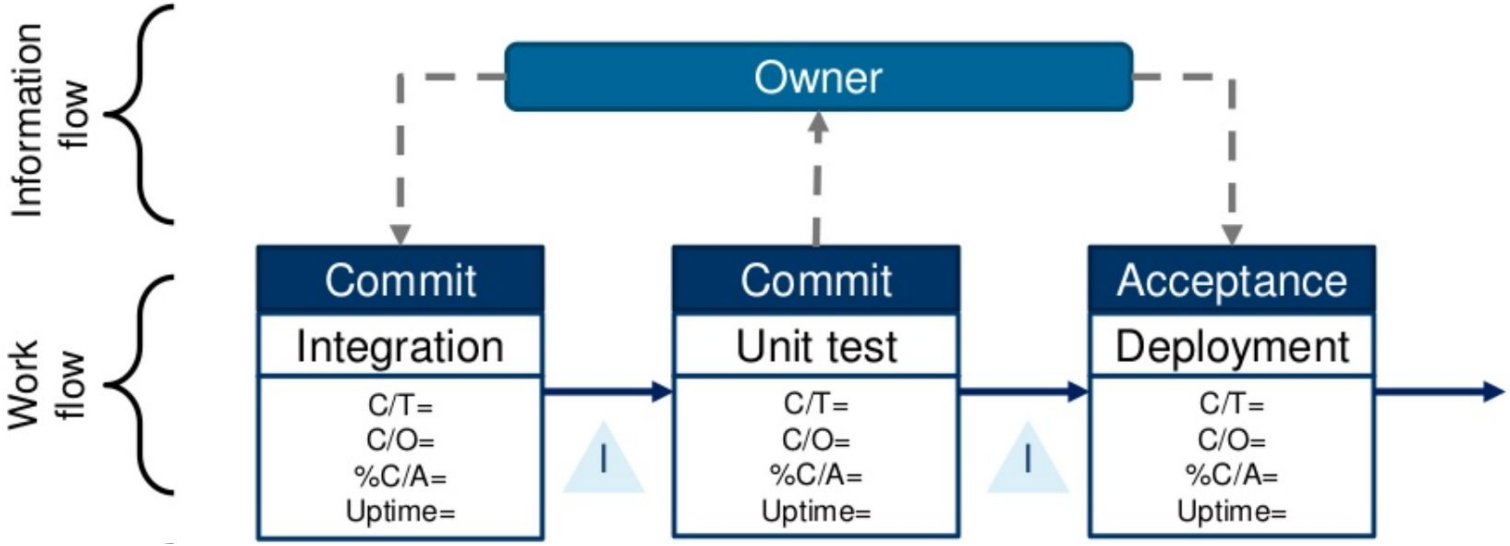
Потік створення цінності

Серія подій, які переводять функцію від розробки коду до впровадження у виробниче

The deployment pipeline, from Continuous Delivery

Автоматичне створення середовищ для Dev, Test і Production на вимогу потрібно, щоб забезпечити ефективну розробку та підтримку програмного забезпечення, необхідно автоматизувати створення середовищ розробки, тестування та виробництва. Цей підхід дозволяє командам створювати необхідні середовища самостійно, швидко та без залучення операційної команди.

Карта потоку створення цінності



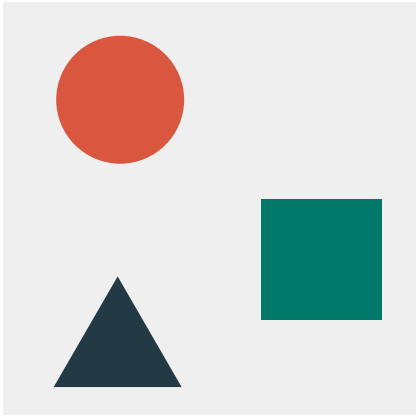
Слабо пов’язана архітектура

Архітектурні принципи DevOps

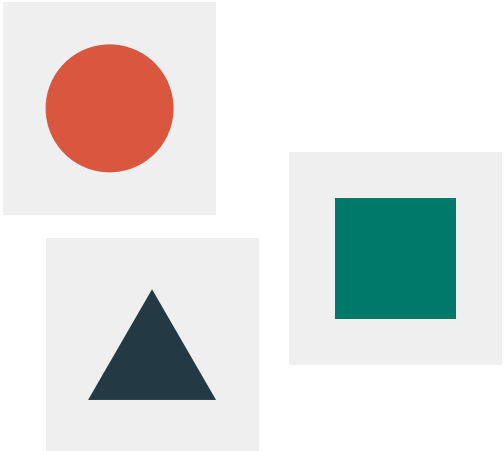
- **Компоненти через сервіси:** Незалежне розгортання, готовність до хмарних платформ, масштабованість.
- **Організація навколо бізнес-можливостей:** Кожен сервіс відповідає за конкретну бізнес-можливість. (Мікросервісна архітектура - MSA)
- **Продукти, а не проєкти:** Принцип “You build it, you run it.” Відповідальність за весь життєвий цикл
- **Розумні кінцеві точки та прості канали:** прості інтерфейси та відсутність центральної маршрутизації або бізнес-логіки, як це часто буває у складних ESB.
- **Децентралізоване управління:** Відсутність стандартизації на одній технології, що сприяє інноваціям.
- **Децентралізоване управління даними:** Транзакції гарантують узгодженість, але створюють часову залежність між сервісами.
- **Автоматизація інфраструктури:** Автоматизовані тести та розгортання.
- **Проєктування з врахуванням відмов:** Кожен сервіс повинен бути готовий до того, що інший сервіс може бути недоступний. Клієнти повинні зберігати працездатність навіть при збої постачальника. Збої виявляються та усуваються швидко.
- **Еволюційний дизайн:** Дизайн сприяє адаптації до нових бізнес-вимог і технологій.

Слабо пов’язана архітектура

Від моноліту до мікросервісів



Monolithic architecture



Microservices architecture

Слабо пов’язана архітектура

Типи архітектури DevOps

	Переваги	Недоліки
Монолітний v1 (Увесь функціонал в одному додатку)	• Простота на початку • Низькі затримки між процесами • Одна кодова база, єдиний блок розгортання • Ефективність використання ресурсів на невеликому масштабі	• Зростання координаційного навантаження • Недостатнє забезпечення модульності • Проблеми з масштабуванням • Розгортання за принципом "все або нічого" • Довгий час збірки
Монолітний v2 (Модель з наборами рівнів: “front end presentation”, “application server”, “database layer”)	• Простота на початку • Легкість виконання запитів із приєднаннями • Єдина схема для розгортання • Ефективність використання ресурсів на малих масштабах	• Схильність до зростання зв’язності • Погана масштабованість і надлишковість • Складність у налаштуванні • Управління схемою за принципом "все або нічого"
Мікросервіс (Модульна, незалежна архітектура з графовими зв'язками vs. Рівнева ізольована архітектура)	• Кожен модуль (юніт) простий • Незалежне масштабування та продуктивність • Незалежне тестування та розгортання • Оптимальне налаштування продуктивності	• Багато співпрацюючих компонентів • Багато невеликих репозиторіїв • Вимоги до складних інструментів і управління залежностями • Мережеві затримки

Infrastructure as Code

Infrastructure as Code (IaC) — це підхід до налаштування та управління інфраструктурою за допомогою програмного коду, що дозволяє автоматично створювати, змінювати і керувати технологічним стеком для застосунків. Замість ручного налаштування обладнання, операційних систем і мережевих ресурсів, IaC використовує програмні інструменти, щоб забезпечити точність, повторюваність і автоматизацію інфраструктурних процесів.

Концепція інфраструктури як коду схожа на сценарії програмування, які використовуються для автоматизації IT-процесів. Однак сценарії в основному використовуються для автоматизації серії статичних кроків, які потрібно багаторазово повторювати на кількох серверах. Інфраструктура як код використовує мову вищого рівня або описову мову для кодування більш універсальних і адаптивних процесів забезпечення та розгортання. Наприклад, можливості інфраструктури як коду, включені в Ansible, IT-інструмент управління та конфігурації, можуть інсталювати сервер MySQL, перевірити, чи MySQL працює належним чином, створити обліковий запис користувача та пароль, налаштувати нову базу даних і видалити непотрібні бази даних

Infrastructure as Code

Процес доставки

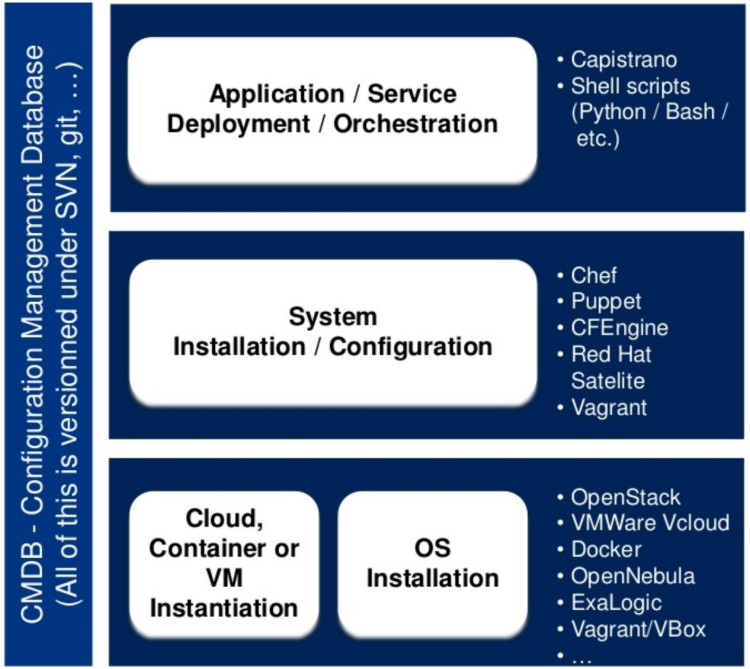
- Розгортання коду додатка та відкат
- Налаштування ресурсів (RDBMS, etc.)
- Запуск додатків
- Об'єднання в кластери

Конфігурація системи

- JVM та сервери додатків
- Middlewares (проміжне програмне забезпечення)
- Конфігурація сервісів (logs, ports, user / groups, etc.
- Реєстрація в системах моніторингу

Установка машини

- Віртуалізація
- Самообслуговувані середовища



Переваги infrastructure as Code

Розробники програмного забезпечення можуть використовувати код для підготовки та розгортання серверів і програм, а не покладатися на системних адміністраторів у середовищі DevOps. Розробник може написати процес інфраструктури як коду, щоб підготувати та розгорнути нову програму для забезпечення якості або експериментального розгортання перед тим, як перейде до операцій для живого розгортання у виробництві.

Завдяки налаштуванням інфраструктури, написаним у вигляді коду, вона може проходити той самий контроль версій, автоматичне тестування та інші етапи безперервної інтеграції та безперервної доставки (CI/CD), які розробники використовують для коду програми. Організація може вибрати поєднання інфраструктури як коду з контейнерами, які абстрагують додаток від інфраструктури на рівні операційної системи. Оскільки ОС і апаратна інфраструктура налаштовуються автоматично, а додаток інкапсулюється поверх них, ці технології виявляються доповнювальними для різноманітних цілей розгортання, таких як тестування, постановка та виробництво.

Незважаючи на свої переваги, інфраструктура як код має потенційні недоліки. Це вимагає додаткових інструментів, таких як система керування конфігураціями, які можуть запровадити криві навчання та можливість помилок. Будь-які помилки можуть швидко поширюватися через сервери, тому важливо стежити за контролем версій і виконувати всебічне передрелізне тестування.

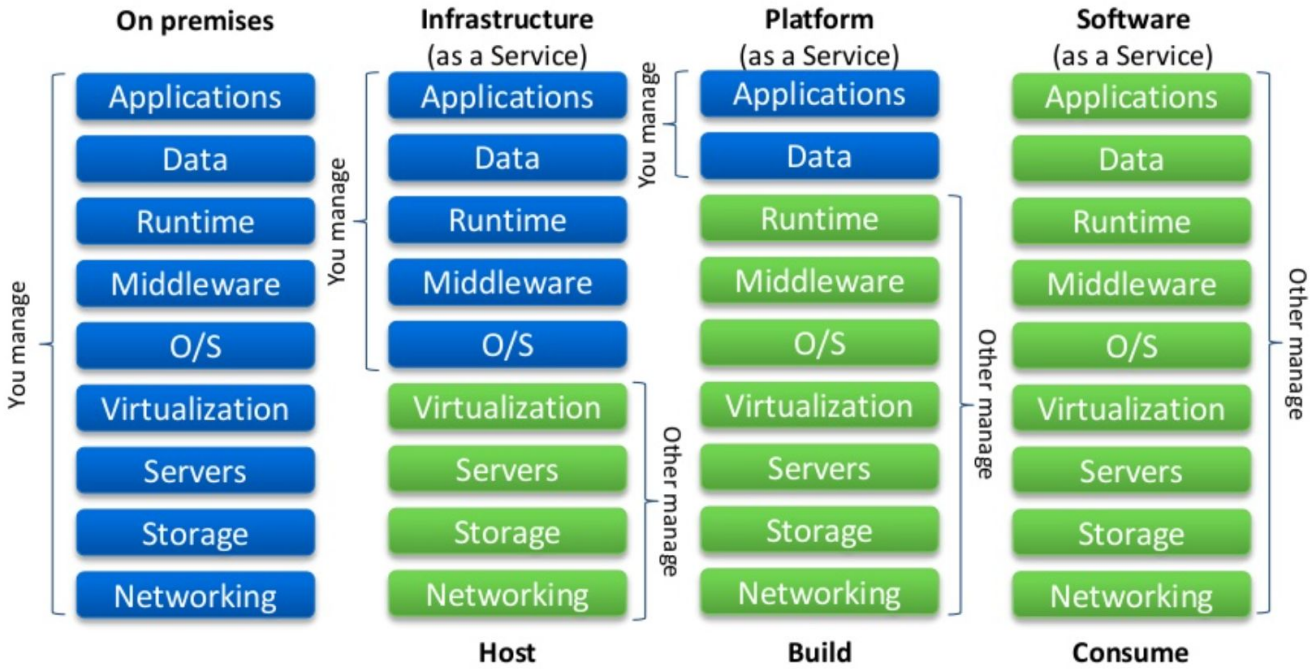
Якщо адміністратори змінюють конфігурації сервера за межами встановленого шаблону інфраструктури як коду, існує ймовірність дрейфу конфігурації. Важливо повністю інтегрувати інфраструктуру як код у системне адміністрування, IT-операції та практики DevOps із добре задокументованими політиками та процедурами.

Слабозв'язана архітектура

Переваги

- Автоматизація побудови та конфігурації середовища за допомогою віртуалізованих середовищ, процесів створення середовища, які починаються з «голого металу», інструментів керування конфігурацією «інфраструктура як код», автоматизованих інструментів конфігурації операційної системи, збирання середовища з набору віртуальних образів або контейнерів, розкручування нові середовища в загальнодоступній хмарі тощо.
- Прийміть концепцію «все як код». У традиційному операційному просторі дедалі більше компонентів можна визначити за допомогою коду, наприклад, програмно визначені мережі. Визначайте дедалі більше артефактів за допомогою коду.
- Create our single repository of truth for the entire system. Put all application source files and configurations as well as all production artefacts in version control. Version control is for everyone in our value stream, including QA, Operations, Infosec, as well as developers
- Створіть наше єдине сховище правди для всієї системи. Помістіть усі вихідні файли програми та конфігурації, а також усі артефакти виробництва в контроль версій. Контроль версій призначений для всіх у нашому потоці створення цінностей, включаючи QA, Operations, Infosec, а також розробників
- Створюйте слабозв’язану архітектуру з чітко визначеними інтерфейсами, які забезпечують з’єднання модулів один з одним для підвищення продуктивності та безпеки.

Слабозв'язана архітектура



Автономні команди

«Команда — це невелика кількість людей із доповнювальними навичками, які віддані спільній меті, набору цілей ефективності та підходу, за який вони несуть спільну відповідальність».

The Wisdom of Teams, by Katzenbach & Smith, 1993

Багатофункціональні автономні команди DevOps:

- складаються з представників усіх дисциплін, які повністю відповідають за розробку та розгортання IT-послуги
- повністю уповноважені та самодостатні для розробки, створення, тестування, розгортання та запуску програмного забезпечення
- працюють автономно та незалежно один від одного, щоб забезпечити безперервний потік змін програмного забезпечення
- мають наскрізну відповідальність. Немає передачі відповідальності та звітності
- володіють всім необхідним досвідом, щоб взяти на себе повну відповідальність
- потрібні члени команди з T-профілем і додатковими навичками
- надавають підтримку продуктам до кінця терміну експлуатації.
- створюють стабільне середовище для ітерацій та вдосконалення

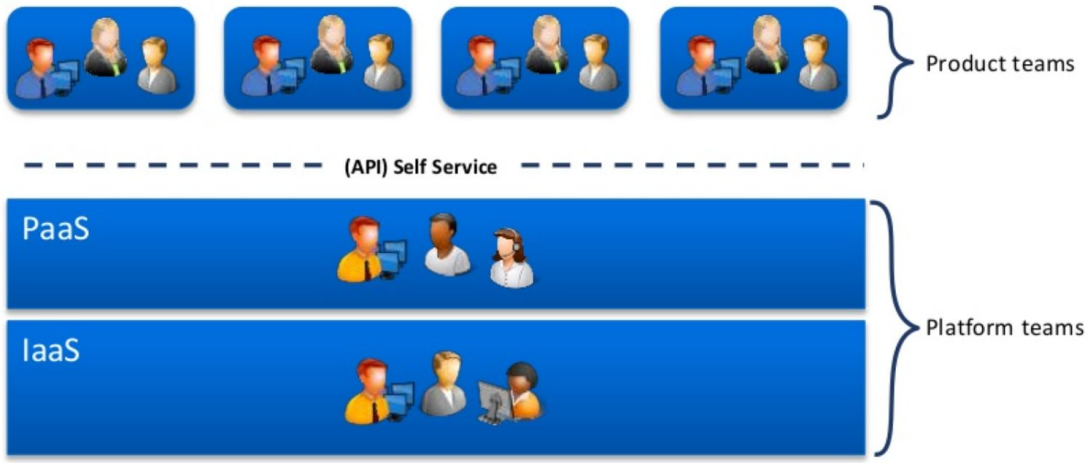
Автономні команди

Переваги

- Орієнтація на цінності, а не на культуру
- Ділитися історіями та досвідом
- Які проблеми ми намагаємось вирішити?
- Чи вирішуємо ми правильні проблеми?
- Чи має наша команда необхідні знання та досвід?

Автономні команди

Створення умов, у яких невеликі продуктові команди можуть працювати **автономно**, **безпечним способом** і **архітектурно розділено**, є ключем до підвищення продуктивності та гнучкості. Використання самообслуговуваних платформ на основі спільного досвіду команд **операцій (Operations)** та **інформаційної безпеки (Information Security)** дозволяє досягти цього.



Автономні команди

Відокремлення команд продукту та платформи:

- Інтерфейси між командами продукту та командами платформи чітко визначені через інтерфейси прикладного програмування (API).
- Команди платформи пропонують багатий набір стандартизованих можливостей самообслуговування для команд продуктів, таких як журналювання, моніторинг, розгортання, резервне копіювання, відновлення та багато іншого. Такий набір можливостей/сервісів дозволяє групам продуктів повністю керувати своїми (програмними) продуктами.
- Команди продукту є клієнтами команд платформи. Вони використовують продукти команди Platform. Ці продукти можна використовувати як повністю автоматизовані сервіси.
- Команди платформи несуть відповідальність за якість своїх продуктів платформи, як-от доступність і продуктивність. Команди продукту несуть відповідальність за якість своїх (додатків) продуктів, наприклад доступність і продуктивність.
- Системні якості продуктів платформи можна відстежувати та керувати ними незалежно від доступності продуктів додатків, які використовують продукти платформи.

Автономні команди

«Команд має бути стільки, скільки можна нагодувати двома піцами»

Закон Конвея стверджує, що «організації, які розробляють системи...обмежені створювати проекти, які є копіями комунікаційних структур цих організацій.... Чим більша організація, тим меншою гнучкістю вона володіє і тим більш вираженим є феномен».

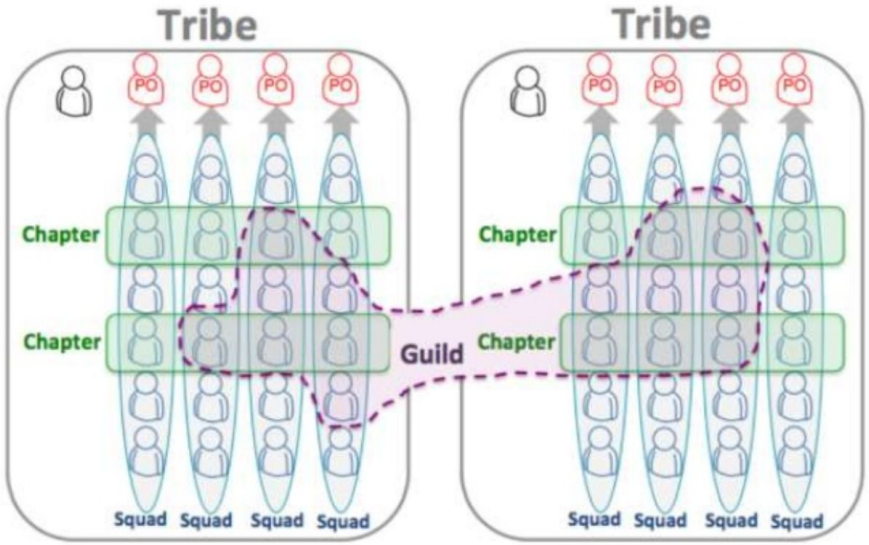
Source: Conway, Melvin E., 1968

Орієнтовані на ринок організації, як правило, є однорідними, складаються з кількох міжфункціональних дисциплін (наприклад, маркетингу, інженерії тощо), що часто призводить до потенційного звільнення в організації. Саме так працює багато організацій, які використовують DevOps. Ринково-орієнтовані команди відповідають не лише за розробку функцій, але й за тестування, захист, розгортання та підтримку своїх послуг у виробництві, від концепції ідеї до виходу на пенсію. Ці команди створені як міжфункціональні та незалежні

Автономні команди

Команди команд

Масштабування за допомогою племен, загонів, орденів і гільдій



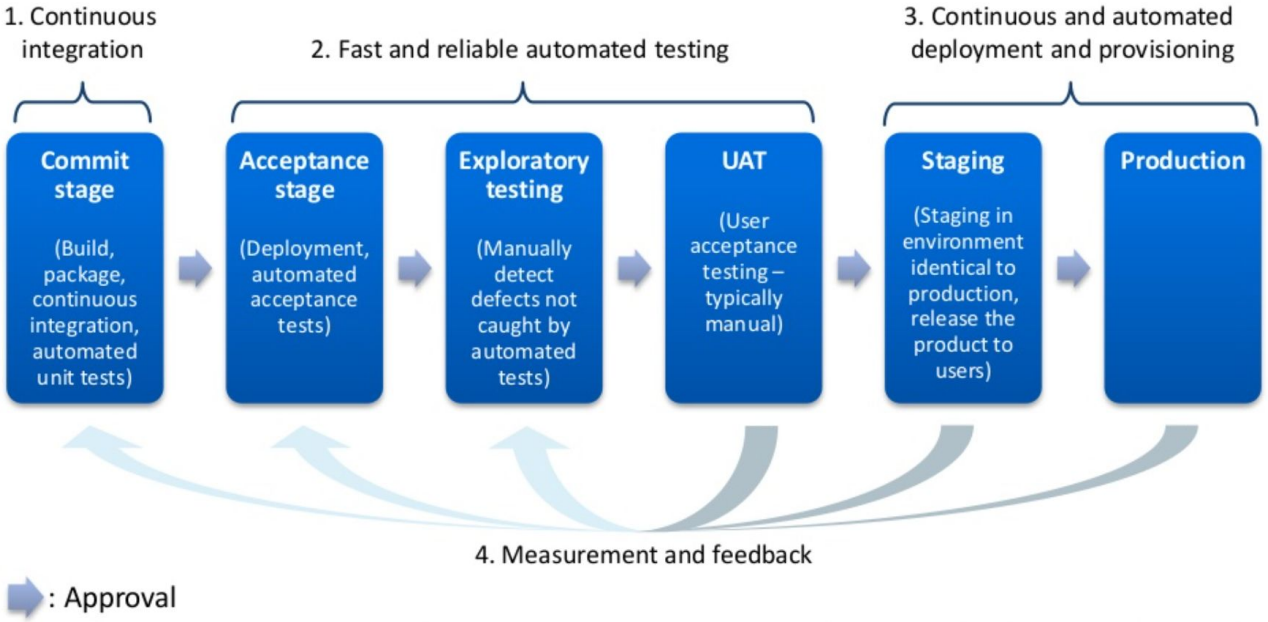
Автономні команди

Вимірювання продуктивності команди

Area	Squad 1	Squad 2	Squad 3	Squad 4	Squad 5
Product owner					
Agile coach					
Influencing work					
Easy to release					
Process that fits team					
A mission					
Organization support					

Практики DevOps

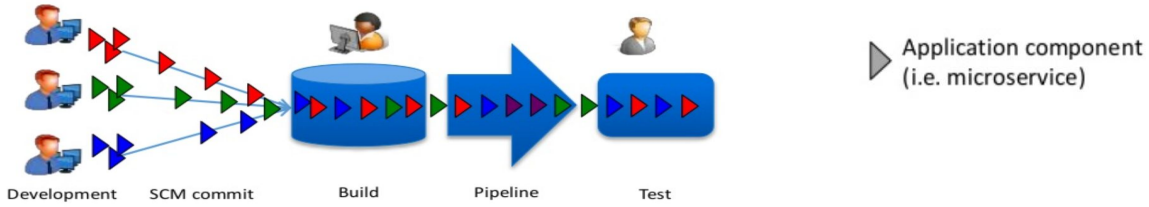
Практики DevOps



Безперервна інтеграція

Безперервна інтеграція (CI) це практика, яка полягає у розробці програмного забезпечення, об’єднання всіх робочих копій розробників у спільну магістраль кілька разів на день.

- Після того, як член команди вносить ряд змін коду в програмне забезпечення керування конфігурацією, зміни можуть бути автоматично об’єднані, проаналізовані, скомпільовані, модульно протестовані та зібрані автоматично.
- Автоматизований процес збірки може створити новий пакет розгортання та опублікувати його в каналі. Таким чином можна реалізувати безперервний потік від фіксації коду до перевіреного пакета розгортання.
- Автоматизований процес збірки важливий для стратегії швидкої відмови. Наприклад, якщо виникають конфлікти злиття коду, збірка має завершитися помилкою, щоб члени команди могли виправити конфлікти.



Безперервна інтеграція

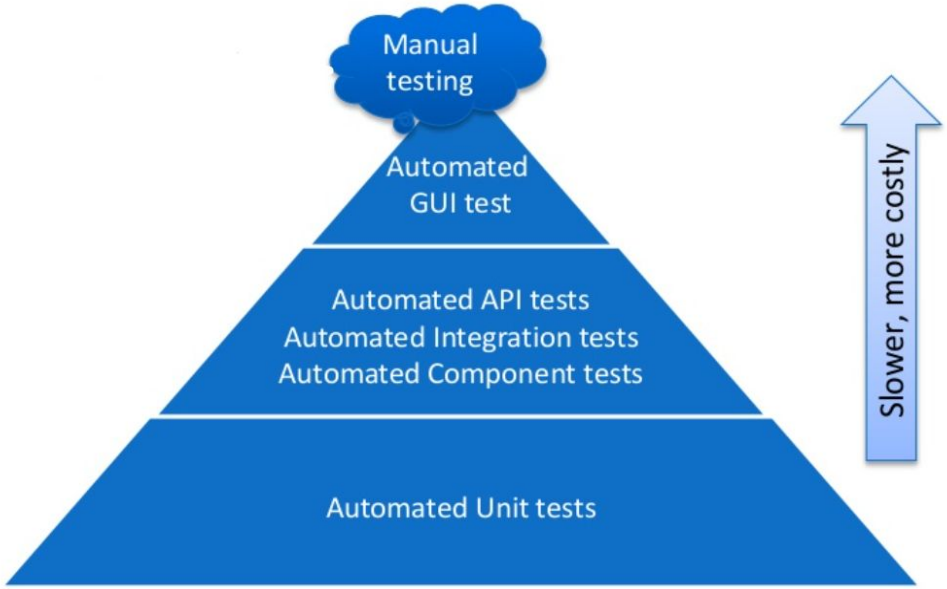
Переваги

- Використання методів розробки на основі транка, коли всі розробники перевіряють свій код у транк принаймні раз на день.
- Часте прикріплення коду до каналу означає, що всі автоматизовані тести можна запускати в системі програмного забезпечення в цілому, і сповіщення отримуються, коли зміна порушує роботу іншої частини програми або втручається в роботу іншого розробника.
- Дисципліна щоденного коду змушує розробників розбивати роботу на дрібніші фрагменти, зберігаючи при цьому стовбур у робочому та доступному для випуску стані

Швидке та надійне автоматизоване тестування

Створіть швидкий і надійний набір автоматизованих перевірочних тестів:

- Модульні тести
- Тести прийняття
- Інтеграційні тести



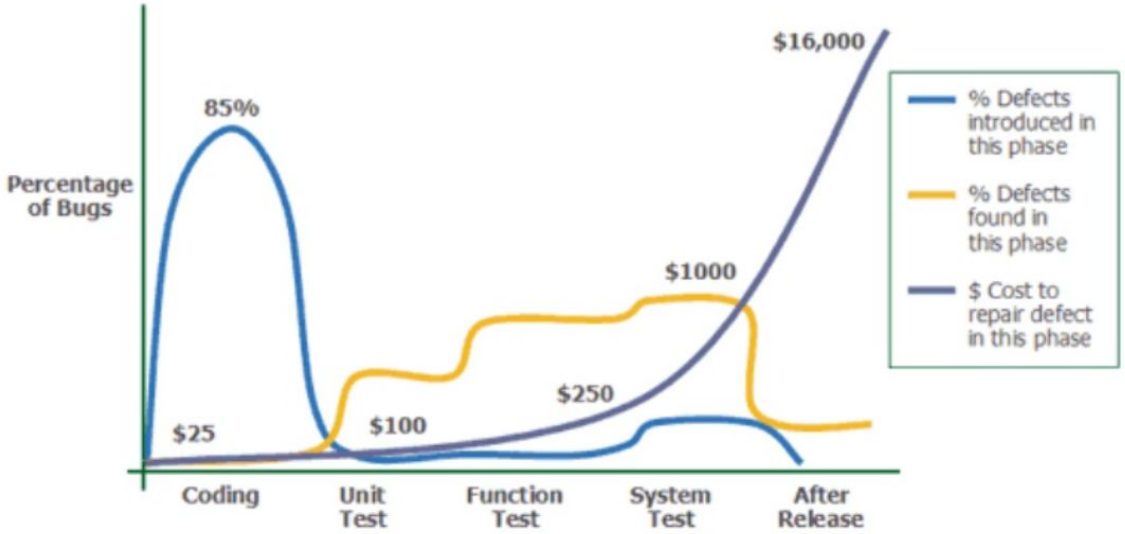
Швидке та надійне автоматизоване тестування

Переваги

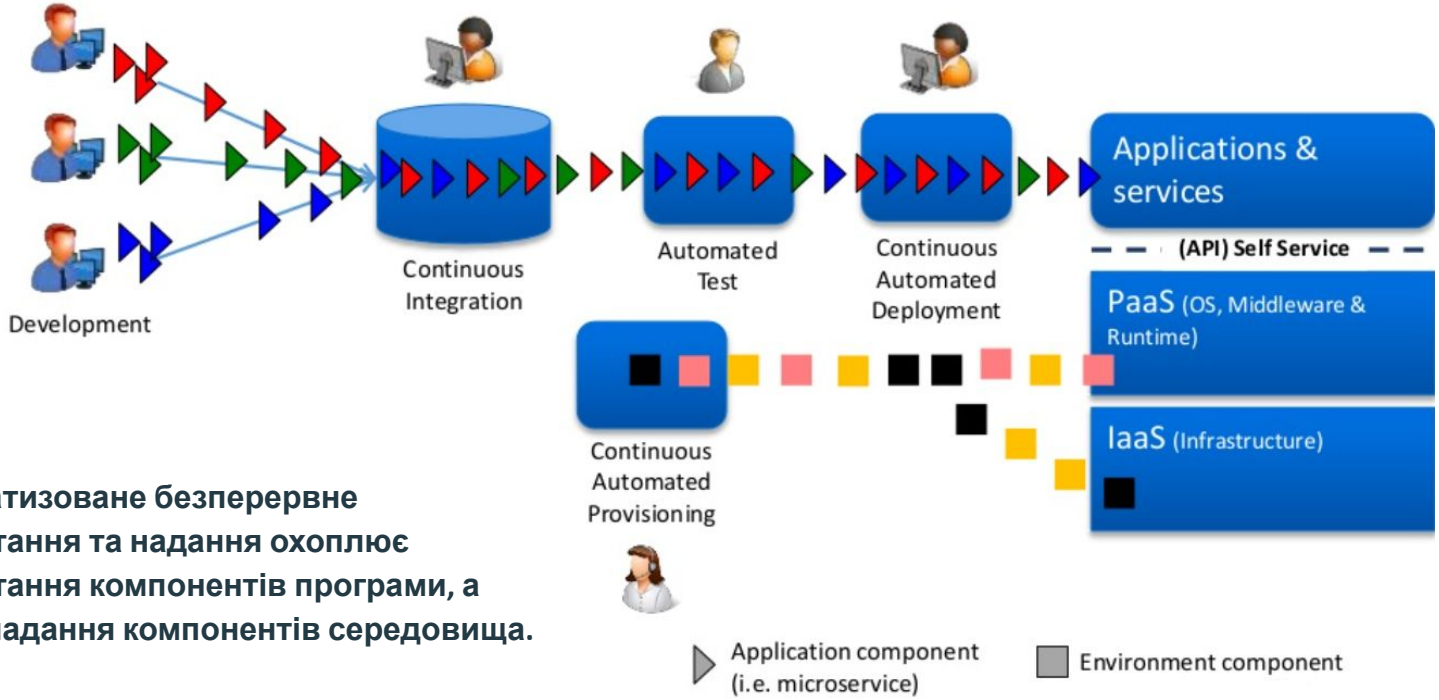
- Виявляйте помилки на ранніх етапах розробки
- Забезпечуйте швидке виконання тестів (in parallel, if necessary)
- Пишіть тести перед кодом ("Test-driven development")
- Автоматизуйте якомога більше ручних тестів Інтегруйте тестування продуктивності в набір тестів
- Інтегруйте тестування нефункціональних вимог у набір тестів
- Зупиняйте процеси при збоях

Швидке та надійне автоматизоване тестування

Виявлення помилок якомога раніше під час тестування завжди окупається пізніше.



Автоматизоване безперервне розгортання та надання



Автоматизоване безперервне розгортання та надання охоплює розгортання компонентів програми, а також надання компонентів середовища.

Автоматизоване безперервне розгортання та надання

Автоматизоване безперервне розгортання автоматично запускає компоненти додатків у виробництво після проходження всіх тестів.

Розгортання програми передбачає:

- Встановлення додатків
- Оновлення додатків
- Налаштування ресурсів
- Налаштування middleware-компонентів
- Запуск/зупинка компонентів
- Налаштування встановленого додатка
- Конфігурація систем (балансувальники, маршрутизатори)
- Перевірка компонентів

Автоматизоване безперервне розгортання та надання

Автоматизоване забезпечення визначається як повністю автоматизоване розгортання та обслуговування компонентів середовища.

- Компоненти середовища програми – це цільові контейнери програми для розгортання, наприклад сервер бази даних або сервер виконання програми.
- Зміни інфраструктури розглядаються як код (Infrastructure as Code)
- Замість того, щоб надавати та керувати компонентами середовища вручну, команди DevOps повинні мати можливість отримувати нові компоненти середовища на вимогу, повністю автоматизовано.
- Нові середовища доставляються протягом хвилин/годин замість тижнів/місяців, отже, велика швидкість
- Контроль, відстеження системних змін, відсутність ручних змін і єдине центральне джерело правди

Автоматизоване безперервне розгортання та надання

Хороші практики

- Стандартизуйте платформи та зменшіть кількість варіацій
- Автоматизуйте процеси розгортання та підготовки, використовуючи той самий механізм розгортання для кожного середовища.
- Увімкніть автоматичне розгортання самообслуговування для збирання, тестування та розгортання.
- Автоматизуйте якомога більше ручних кроків, наприклад:
    - Упаковка коду способами, придатними для розгортання
    - Створення попередньо налаштованих образів або контейнерів віртуальних машин
    - Автоматизація розгортання та налаштування проміжного ПЗ
    - Копіювання пакетів або файлів на робочі сервери
    - Перезапуск серверів, програм або служб
    - Генерація файлів конфігурації з шаблонів
    - Виконання автоматизованих димових тестів, щоб переконатися, що система працює та правильно налаштована
    - Запуск процедур тестування
    - Створення сценаріїв і автоматизація міграції бази даних

Автоматизоване безперервне розгортання та надання

Відокремте розгортання від випусків

- *Розгортання — це встановлення певної версії програмного забезпечення в певному середовищі*
- *Випуск — це коли ми робимо функцію (або набір функцій) доступною для всіх наших клієнтів або сегмента клієнтів*

Автоматизоване безперервне розгортання та надання

Шаблони випуску

Шаблони випусків на основі середовища: це те, де ми розгортаємо два або більше середовищ, але лише одне середовище отримує реальний трафік клієнтів

- Найпростіший візерунок називається синьо-зелене розгортання. У цьому шаблоні ми маємо два робочих середовища: синє та зелене. У будь-який час лише один із них обслуговує клієнтський трафік
- Патерн випуску canary автоматизує процес випуску, який поступово переходить до більших і критичніших середовищ, оскільки ми підтверджуємо, що код працює, як розроблено. Коли здається, що щось йде не так, ми повертаємось назад; інакше ми розгортаємо наступне середовище.
- Кластерна імунна система розширює шаблон релізу canary, пов'язуючи нашу систему моніторингу виробництва з нашим процесом випуску та автоматизуючи відкат коду, коли продуктивність робочої системи, спрямована на користувача, виходить за межі попередньо визначеного очікуваного діапазону.

Шаблони випусків на основі програми: тут ми змінюємо нашу програму, щоб ми могли вибірково випускати та показувати певні функції програми шляхом невеликих змін у конфігурації

- Запровадити перемикачі функцій, які надають нам механізм вибіркового ввімкнення та вимкнення функцій, не вимагаючи розгортання робочого коду, напр. щоб увімкнути темні запуски.

Вимірювання та зворотній зв'язок

Телеметрію можна визначити як автоматизований процес зв’язку, за допомогою якого вимірювання та інші дані збираються у віддалених точках і згодом передаються до приймального обладнання для моніторингу.

Поширена виробнича телеметрія як у коді, так і в виробничому середовищі гарантує швидке виявлення та виправлення проблем:

- У додатках
- У середовищі
- У конвеєрі розгортання

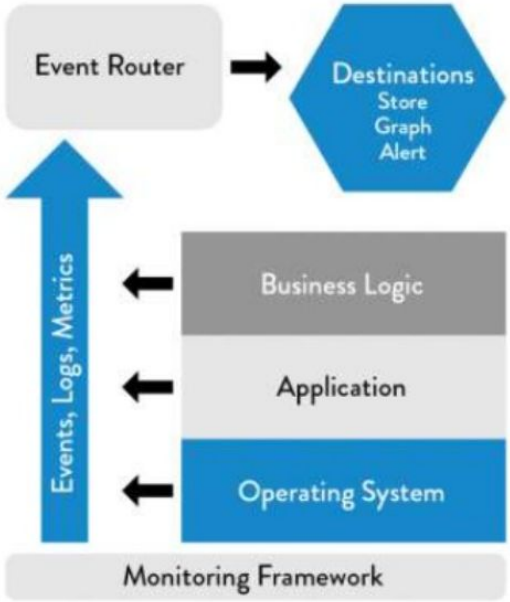
Ключові показники DevOps

- Термін виконання (запит до виконання)
- Час процесу (від початку роботи до виконання)
- Відсоток виконання та точності (%C/A)

Вимірювання та зворотній зв'язок

Структура моніторингу

- Збирайте дані на рівні бізнес-логіки, програми та середовища
- На кожному з цих рівнів створіть телеметрію у вигляді подій, журналів і показників.
- Створіть маршрутизатор подій, відповідальний за зберігання подій і показників
- Ця можливість дозволяє візуалізувати, трендувати, сповіщати про виявлення аномалій тощо



Вимірювання та зворотній зв'язок

Хороші практики

Створіть телеметрію, щоб бачити та вирішувати проблеми

- Створіть централізовану інфраструктуру телеметрії
- Створюйте телеметрію журналу додатків, яка допомагає у виробництві. Різні рівні реєстрації, деякі з яких також можуть ініціювати сповіщення, як-от Debug, Info, Warning, Error, Fatal
- Створіть необхідну інфраструктуру та бібліотеки, щоб будь-кому, хто займається розробкою чи експлуатацією, було якомога легше створювати телеметрію для будь-якої функціональності, яку вони створюють
- Створити доступ самообслуговування до телеметричних та інформаційних випромінювачів

Аналізуйте телеметрію, щоб краще передбачати проблеми та досягати цілей

- Використовуйте середні значення, стандартне відхилення та методи виявлення аномалій для виявлення потенційних проблем.

Увімкніть зворотний зв’язок, щоб відділи розробки й експлуатації могли безпечно розгортати код

- Використовуйте телеметрію, щоб зробити розгортання безпечнішим
- Нехай розробники слідкують за роботою. Одним із найпотужніших методів взаємодії та дизайну взаємодії з користувачем (UX) є контекстний запит. Це коли команда продукту спостерігає, як клієнт використовує додаток у своєму природному середовищі, часто працюючи за своїм столом.
- Попросіть розробників спочатку самостійно керувати службою виробництва

Вимірювання та зворотній зв'язок

Відгук людини

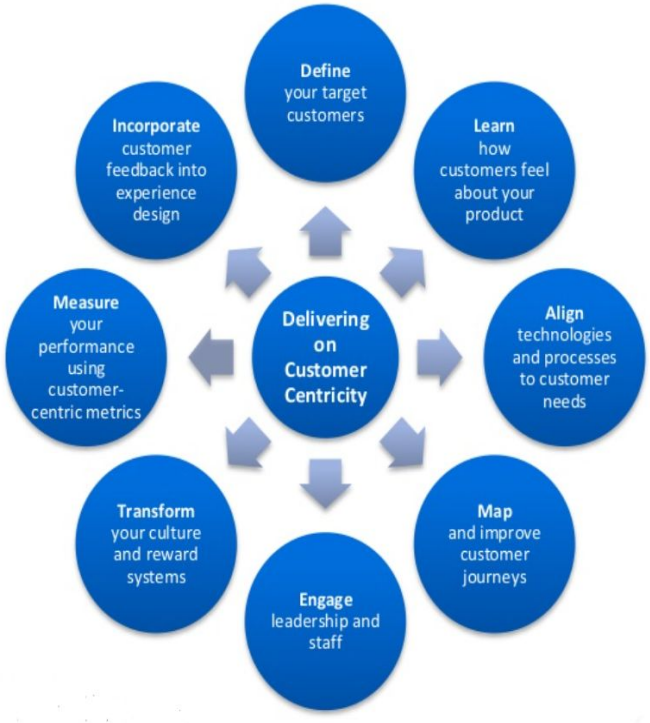
Ключовим інструментом, який повинні використовувати та стимулювати лідери, є зворотний зв’язок з людьми. Надання та отримання зворотного зв’язку є основою для вдосконалення та розвитку командної роботи. Як для керівників, так і для членів команди життєво важливо вчитися та практикуватися давати та отримувати зворотній зв’язок у шанобливій манері.

Give feedback	Receive feedback
Describe specific observations	Listen without interruption
Explain what it does to you	Avoid discussions or excuses
Wait and listen to clarifying questions	Check if there is clear understanding
Give concrete suggestions OR recognition / incentive	Recognize other person’s position Thank him/her Determine whether the feedback is applied

Люди у DevOps

Орієнтований на клієнта

Вкрай важливо мати короткі цикли зворотного зв’язку з реальними клієнтами та кінцевими користувачами. Тому вся діяльність, пов’язана зі створенням IT-продуктів і послуг, повинна зосереджуватися навколо клієнтів.



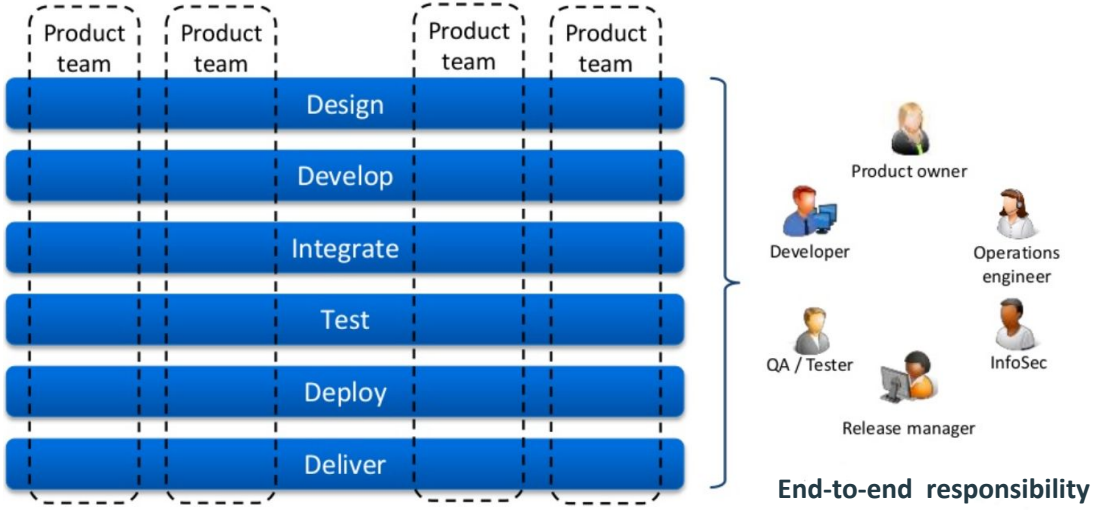
Орієнтований на клієнта

Хороші практики

- Почніть з уявлення кінцевої мети: під час ретроспектив запитуйте в людей «Чому?»
- Почніть із визначення цілей клієнта (голос клієнта). Загальне розуміння думки клієнта (VoC) є важливим, оскільки на наступному етапі ви разом із командою визначите, які дії справді додають цьому VoC, а які – ні.
- Заохочуйте клієнтів відвідувати демонстрації, впроваджуйте відгуки користувачів у розкадровку, дозволяйте клієнтам писати про продукт і відповідати, організовуйте людей навколо продукту, заохочуйте команду писати блоги про продукт (ви створюєте його; ви керуєте ним).

Наскрізна відповідальність

В організації DevOps команди організовані вертикально, щоб вони могли нести повну відповідальність за продукти та послуги, які вони надають. Наскрізна відповідальність означає, що команда несе відповідальність за якість і кількість послуг, які вона надає своїм клієнтам.



Наскрізна відповідальність

Хороші практики

Caring about the end-to-end responsibility might be the most crucial ingredient for DevOps. When people care and have the required skills, knowledge, and resources, they can and will collaborate to live up to their responsibility. If they care, they will learn, adapt, improve, and provide great services and value.

- Усі члени команди несуть відповідальність за повний продукт, який включає повний цикл доставки, а також експлуатацію/надання підтримки клієнтів протягом життєвого циклу продукту.
- *Якість закладена від початку створення команд до звільнення. Це серце кожної діяльності. Це ніколи не ставить під загрозу. Ми цінуємо повну прозорість.*
- Заохочуйте команду використовувати свої навички так, як вони знають найкраще; уникати зрізання кутів; практика автоматизації (тестування, розгортання та забезпечення), безперервне вдосконалення та прозорість (монітори).
- Не пояснюйте, «як» робити, не запитуйте, «що» потрібно, відкрито вирішуйте питання про збиття з рейок, дозвольте людям зрозуміти, як це робити, винагороджуйте за відповідальність, винагороджуйте за невдачу, створіть прозорість у тому, що кожен робить.

Співпраця

Сенс співпраці полягає в тому, щоб працювати разом для досягнення мети. Це центральна тема для команд DevOps. Співпраця означає спільні організаційні цілі, емпатію та навчання між різними людьми

Співпраця

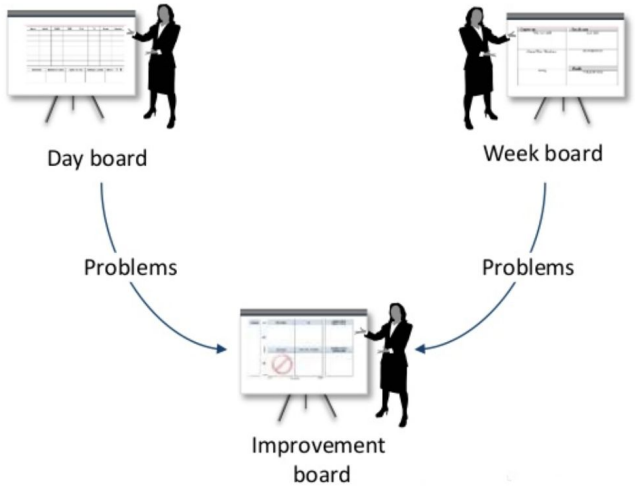
Хороші практики

Візуальне керування є одним із найпотужніших інструментів для стимулювання співпраці та гарантує, що підводні камені будуть виявлені. Інструмент допомагає переконатися, що робота, виконана командою, постійно відображається на дошках. зокрема:

- Визначте роботу та перешкоди.
- Передайте важливу інформацію.
- Покажіть, як виконати завдання.
- Демонструйте планування та пріоритети.

У своїй основній формі візуальне управління включ
три дошки, як показано на малюнку

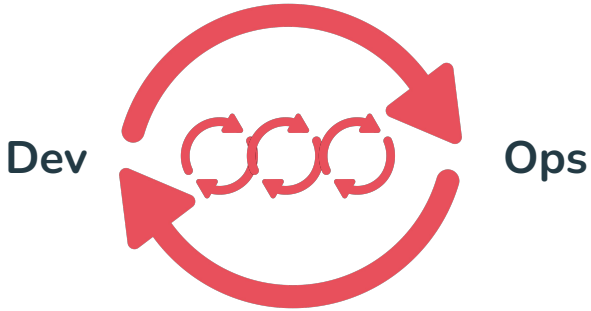
.



Навчання та вдосконалення

Коли ми працюємо в складній системі, нам неможливо передбачити всі результати дій, які ми виконуємо.

Щоб ми могли безпечно працювати в складних системах, наші організації повинні ставати все кращими в самодіагностиці та самовдосконаленні, а також повинні мати навички виявлення проблем, їх вирішення та примноження ефектів, роблячи рішення доступними для всієї організації.



Навчання та вдосконалення

Хороші практики

Постійне вдосконалення стосується вирішення проблем:

- Бачити та пріоритизувати проблеми
  - Виявляти проблеми
  - Приймати проблеми як частину повсякденного життя
  - Ініціювати дії для визначення проблем, які потребують негайного вирішення
- Розв'язання проблем
  - Інвестуйте час та інші ресурси
  - Зрозумійте корінні причини проблем
  - Повністю усуньте проблеми
- Обмін уроками, отриманими з досвіду
  - Плануйте аналіз подій без звинувачень після інцидентів. Ключове питання, на яке варто звернути увагу: «Чому це рішення здавалося мені правильним у той момент, коли я його приймав?»
  - Діліться отриманими уроками з іншими членами IT-організації, щоб вони також могли отримати з цього користь.
- Знижуйте толерантність до інцидентів, щоб виявляти все слабші сигнали про збої, посилюючи слабкі сигнали про проблеми.
- Оскільки ми піклуємося про якість, ми навіть навмисно вводимо збої в наше робоче середовище, щоб у контрольованих умовах вивчити, як наша система може зазнати невдачі.

Навчання та вдосконалення

Хороші практики

- Створіть єдине, спільне сховище вихідного коду для всієї організації
- **Використовуйте чати та чат-ботів для автоматизації та накопичення знань організації.** Інтегруйте інструменти автоматизації в чати, щоб створювати прозорість і документувати процеси. Це допоможе фіксувати знання та прискорювати комунікацію.
- **Виділіть час для організаційного навчання та вдосконалення.** Плануйте регулярні активності, як-от «весняні чи осінні прибирання», тижні перевірки та перегляду завдань, хакатони або виділення 20% часу для інновацій.
- Дозвольте кожному навчати та навчатися
- Розвивайте внутрішніх консультантів і наставників для поширення практик
- Формуйте середовище, де кожен постійно навчається, використовуючи науковий метод для перевірки гіпотез і уникнення припущень.

Навчання та вдосконалення

Хороші практики

- Автоматизуйте стандартизовані процеси у програмному забезпеченні для повторного використання
- Проектуйте для операцій через кодифіковані нефункціональні вимоги
- Поширюйте знання, використовуючи автоматизовані тести як документацію
- Включайте повторно використовувані операційні користувацькі історії в розробку
- Перетворюйте локальні відкриття на глобальні покращення

Експерименти та ризики

Експериментування — це перевірка гіпотези. Іншими словами, спроба чогось нового на основі потреби називається експериментуванням. Команди DevOps повинні мати сміливість експериментувати, приймаючи можливість невдачі. Вони вміють швидко виявляти помилки, робити наступний крок або повернутися на кілька кроків назад під тиском часу, щоб забезпечити якість із самого початку.

- Забезпечте зацікавленість клієнтів
- Визначте та надайте мінімально життєздатний продукт (MVP)
- Зосередьтеся на меті «цінність клієнта доставляється з першого разу прямо в потоці»
- Робіть маленькі кроки і не проводьте великих експериментів

НЕМАЄ ІННОВАЦІЙ БЕЗ ЕКСПЕРИМЕНТУ

Експерименти та ризики

Хороші практики

- Створіть середовище, у якому люди можуть працювати на максимум своїх можливостей, де вони відчувають натхнення, бажають бути присутніми, почуваються бажаними та заохочуються мислити нестандартно.
- Виділіть час, використовуйте середовище миттєвої пісочниці, усуньте перешкоди, підтримуйте ідеї, забезпечуйте безпечні умови для невдач. Відзначайте навіть невдачі!
- Запровадьте «ігрові дні» для відпрацювання сценаріїв невдач. Мета «ігрового дня» полягає в тому, щоб допомогти командам моделювати та відпрацьовувати аварійні ситуації, надаючи їм можливість практикуватися в управлінні кризами.
- Вводьте збої у виробниче середовище, щоб підвищити стійкість системи та сприяти навчанню. Наприклад, Netflix створив сміливий і нестандартний сервіс під назвою **Chaos Monkey**, який імітує відмови серверів, постійно й випадково "відключаючи" виробничі сервери. Це дозволяє командам перевіряти готовність системи до збоїв і вдосконалювати її надійність.
- Наймайте людей із відповідними амбіціями, відходьте від моделі функціональних назв, підтримуйте експерименти, підтримуйте автоматизацію ручних завдань, не зосереджуйтесь лише на використанні.
- Забезпечте хорошу каву, створіть приємне та відкрите середовище, інвестуйте в додаткові об’єкти, як-от ігри, проводьте конкурси чи організовуйте напої наприкінці тижня, дозволяйте командам модифікувати/пристосувати офіс відповідно до своїх потреб.

Елементи керування DevOps

Управління змінами

Традиційні механізми контролю змін можуть призводити до небажаних наслідків: Вони можуть збільшувати час виконання та зменшувати силу та оперативність зворотного зв’язку з процесу впровадження.

Сміливо усувайте бюрократичні процеси

Сприяйте координації та плануванню змін: Навіть у архітектурі з низьким ступенем зв’язності, коли багато команд виконують сотні незалежних розгортань щодня, можуть виникати ризики взаємного впливу змін. Щоб зменшити ці ризики, ми можемо використовувати чат-кімнати, щоб повідомляти про зміни та завчасно знаходити можливі конфлікти.

Ефективна політика управління змінами повинна враховувати, що різні типи змін пов'язані з різними ризиками, і кожен тип змін потребує різного підходу. Ці процеси визначені в **ITIL**, який розподіляє зміни на три категорії:

- Стандартні зміни
- Нормальні зміни
- Термінові зміни

Управління змінами

Увімкніть парне програмування, щоб покращити всі наші зміни. В одній загальній моделі поєднання один інженер виконує роль драйвера, людини, яка фактично пише код, тоді як інший інженер діє як навігатор, спостерігач або покажчик, особа, яка перевіряє роботу під час її виконання. Інший шаблон парного програмування підсилює розробку, керовану тестуванням (TDD), завдяки тому, що один інженер пише автоматизований тест, а інший інженер реалізує код.

Запровадьте **рецензування змін колегами**. Замість того, щоб вимагати затвердження від зовнішніх органів перед впровадженням, вимагайте від інженерів отримання відгуків на їхні зміни від колег.

- Усі зміни (до коду, середовища тощо) повинні бути перевірені кимось із команди перед комітом в основну гілку.
- Усі члени команди повинні слідкувати за стрімом комітів своїх колег, щоб виявляти потенційні конфлікти та аналізувати їх.
- Чітко визначте зміни, які вважаються високоризикованими (наприклад, зміни в базі даних, модулі безпеки, такі як автентифікація). Такі зміни повинні проходити додаткове рецензування експертами з відповідної тематики.
- Якщо вплив зміни важко зрозуміти навіть після кількох прочитань, вона повинна бути розділена на кілька менших змін. Кожна частина має бути зрозумілою з першого погляду.

Інформаційна безпека

Замість перевірки безпеки продукту в кінці процесу ми будемо створювати та інтегрувати засоби контролю безпеки в щоденну роботу команд розробки та операцій. Таким чином, безпека стане частиною роботи кожного співробітника щодня.

- Зробіть безпеку частиною роботи кожного
  - Інтегруйте безпеку в демонстрації ітерації розробки
  - Інтегруйте безпеку в відстеження дефектів і аутоаналіз
- Інтеграція превентивних засобів контролю в загальне сховище вихідного коду
- Інтеграція безпеки в процес розгортання (deployment pipeline)
- Інтеграція безпеки з телеметрією для покращення виявлення та відновлення
  - Створення телеметрії безпеки у додатках
  - Створення телеметрії безпеки в середовищі
- Забезпечте безпеку програми, середовища та конвеєра розгортання
- Зменшіть залежність від розподілу обов’язків

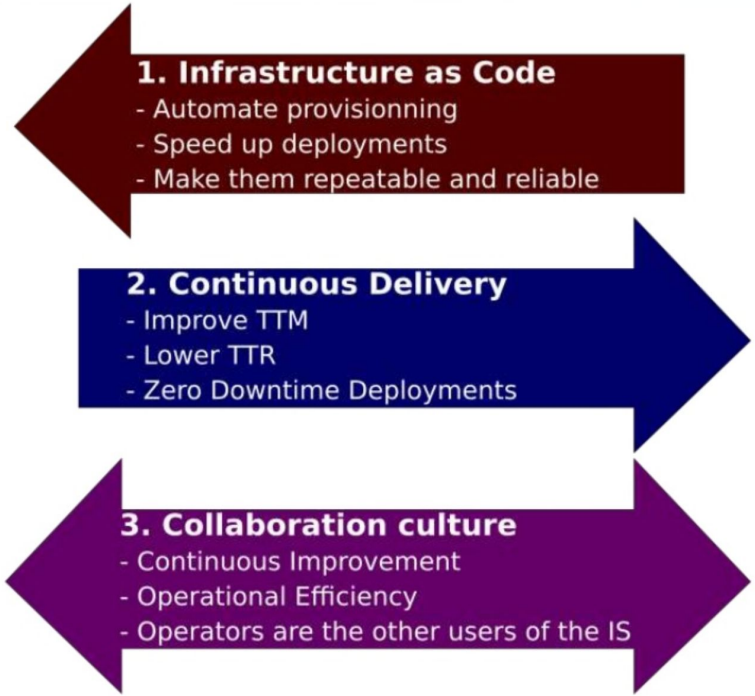
Розподіл обов'язків

Зменшіть залежність від розподілу обов’язків

- Протягом десятиліть **розділення обов'язків** використовувалося як один із основних методів контролю для зниження ризиків шахрайства чи помилок у процесі розробки програмного забезпечення.
- Розподіл обов’язків часто сповільнює та зменшує зворотній зв’язок, який інженери отримують щодо своєї роботи. Це заважає інженерам брати повну відповідальність за якість своєї роботи та зменшує здатність фірми створювати організаційне навчання. Отже, де це можливо, нам слід уникати використання розподілу обов’язків як засобу контролю.
- Замість цього ми повинні вибрати такі елементи керування, як парне програмування, безперервна перевірка реєстрації коду та перегляд коду. Ці засоби контролю можуть дати необхідну впевненість щодо якості роботи.

Висновок

Висновок


1. Infrastructure as Code
- Automate provisionning
- Speed up deployments
- Make them repeatable and reliable
2. Continuous Delivery
- Improve TTM
- Lower TTR
- Zero Downtime Deployments
3. Collaboration culture
- Continuous Improvement
- Operational Efficiency
- Operators are the other users of the IS

Q&A