1.5 Типи сучасних команд

Для високої ефективності девопс в організації потрібні ґрунтовні зміни. Розробники починають брати участь в експлуатації. Прикладом може бути впровадження в продукті health checks інфраструктури та liveness probes продукту. Простими словами — це коли перевірка доступності сервісу інтегрована в коді. Також гарним прикладом будет інтеграція Application Performance та Real User Monitoring. Процес включення подібних фіч додає розуміння, залученість та зацікавленість розробника в успішному розгортанні та моніторингу свого коду.

Оперейшен інженер в цій топології посилено прокачує софт скіли на ком'юнікейшн — йому має бути комфортно працювати в парі з розробником. Напевно, тут проходить тонка грань між анти-паттерном та ефективністю. Все пов'язане на вмінні будувати комунікацію у команді. До прикладу, я сьогодні працюю в проекті, де софт скіли відіграють найважливішу роль.

Приклад досить нетривіального, але цікавого рішення. Зазвичай тести відбуваються на синтетичному трафіку. Іншими словами — трафіку, спеціально згенерованому для тестів. Що буде, якщо ми зробимо одностороннє віддзеркалення продуктивного трафіку на тестове середовище? Простіше кажучи, завернем копію трафіку, що приходить від реальних клієнтів, на тестове середовище. Тут продукт може бути протестований набагато якісніше. На живому трафіку ми отримаємо всю його непередбачуваність, реальні затримки, помилки та інше. Як саме це робиться, ми розглянемо на практичних заняттях. Такого роду інтеграції долучають до комунікації всіх членів команди. Це однозначно прокачує софт скіли та вимагає поділяти відповідальність.

Повний взаємний розподіл експлуатаційних обов'язків — оперейшен інтегрований у продакт девелопмент. Так працюють Netflix та Facebook, розробляючи один основний веб продукт. Усі учасники

високо сфокусовані тільки на одній меті — на якісному продукті. В книзі “Наснага” за авторством Петті Мак-Корд колишньої директорки з персоналу Netflix “Бізнес за правилами Netflix” — описує тип з високою потенційною ефективністю. Компанія пройшла складний шлях та сьогодні це найвищі зарплати на ринку, вільний робочий графік та безстрокова відпустка.

"Оперейшн — це інфраструктура як сервіс" і "Девопс як сервіс". Це організації з лімітованим досвідом експлуатації та низькою швидкістю змін. На допомогу їм приходять хмарні провайдери або компанії, що надають повний спектр інфраструктурних послуг.

Якщо експертизи не вистачає — замовник звертається за девопс сервісом. Визначає оперейшен фічі, які потрібно імплементувати. Відповідно, їм допомагають розгорнути тестове оточення, автоматизувати інфраструктуру та CI/CD процеси. Також, можливий обмін досвідом девопс практик та інструментів із спеціалістами замовника. Думаю, більшість наших слухачів так чи інакше будуть залучені в подібну схему. Це загалом яскравий приклад для України — більшість проєктів це аутсорс і аутстаф, саме такого виду послуг.

Потенційну ефективність данного типу визначено як середня. Аутстаф, тобто співробітники поза штатом компанії замовника, вочевидь, не настільки сильно вмотивовані, що може викликати меншу залученість в розвиток девопс в організації в цілому.

DevOps євангелісти. В організації з великим розривом процесів в командах запрошують євангеліста. Місія — зблизити розробників та оперейшенс. Ці спеціалісти покликані допомогти розвіяти побоювання. Перше — вони позиціонуються як аут-оф бізнес. Для команди експлуатації презентуються божевільні ідеї типу стендапів та канбану. А розробників посвячують у "брудні подробиці" лоад балансерів, нетвору, Kubernetes тощо. Люди починають помічати зростаючу ефективність співпраці та приходить ясність цілей та впевненість у змінах. Потенціал ефективності визначено як середній, оскільки знову

ж таки, людський фактор, готовність команд та організації до змін впливає на результат.

Як відомо, Ірландія — це не лише кліфи, унікальна природа північного острова, але також — це центр зосередження європейських офісів медіа гігантів: Apple, Facebook, Twitter та LinkedIn, Intel, Microsoft, Google. Безперечно, ми всі знайомі з першою та другою книгами Маквелі від Google — **Сайт Реабіліті інжиніринг**. Про те як розробники ведуть свій код до продакшен та беруть участь в он-колах тобто чергуваннях. Як набираються команди з софтвар інженерів і як будуються взаємини, які скіли обов'язкові. В даному випадку, SRE — це компетентний софтвар інженер: він так само впевнено читає як і пише код. Плюс софт скіли комунікації. Але в першу чергу, це — компанії та команди з високо розвиненим інжинірингом та зрілою організаційною культурою. Сайт реабіліті Інженер може відмовити в деплої фічі та попросити розробника виправити код. Розробник надає підтвердження, що його код придатний до продакшн. Це можуть бути логи, тести, проби продуктивного трафіку тощо. Звичайно, ці зміни не з найлегших в організації.

І якщо ж комунікацію між розробниками та експлуатацією вже якось налагодити вдалося, то розрив у володінні інструментарієм зі зрозумілих причин не досягти. Не кожний розробник вивчатиме експлуатацію Kubernetes, оскільки він зайнятий розробкою. Так само як інженер експлуатації не занурюється в якийсь фреймворк програмування. Кожен зайнятий своєю справою. На базі цього і з'являються нові сервіси і підходи, покликані хоч якось покращити ситуацію.

Прикладом може стати **container-driven колаборація**. Компромісний технологічний варіант уникнути спілкування з розробниками. Коли артефактом, за домовленістю, визначається контейнер, готовий до запуску. Розробник піклується про інкапсулювання деплойменту та рантайм конфігурації до продукту у самому контейнері. Він готовий до

запуску. Розробник передає вже зібраний незмінний контейнер експлуатації, а ті, у свою чергу, просто безстрашно його запускають — швидко та ефективно. Якщо артефактом є контейнер, а між сервісами дотримуються контракт та зворотна сумісність — все має запрацювати. JFDI — принцип неупередженого чи безстрашного деплоя: Просто запусти це! — just fairless deploy it.

Отже, коли ми маємо досить нетривіальну ситуацію з погляду розробки та експлуатації і продукт та інфраструктура досить специфічні. Обидві сторони не мають повного уявлення про технології та підходи один одного. Тому ми тут реалізуємо container-driven підхід. Розробники пишуть код, збирають його та упаковують у контейнер. Артефакт буде контейнер імідж. Інфраструктура має одну вимогу — це має бути арм архітектура. Це так званий пункт контракту.

Розробники враховують це на етапі збирання та компіляції. Додатком до основного коду програми з боку розробки стане Dockerfile з інструкціями білда контейнера іміджу. А helm пакет з інструкціями для деплойменту на Куберенетіс додадуть інженери експлуатації.

У результаті, автоматизації CICD, система розгортається на інфраструктурі без необхідності додаткових узгоджень між сторонами — головне дотримання контракту. Ми можемо запакувати в контейнер все, що завгодно, як кажуть, контейнер стерпить все. Найважливішим тут залишається культура відносин тих, хто упаковує контейнери і тих, хто їх розгортає.