1.4 З чого почати

На цьому уроці ми говоримо про DevOps Infinity Loop.

І прямо зараз почнемо будувати фундамент твоєї спеціалізації як інженера DevOps і, якщо простими словами, ми відповімо на питання, яке найчастіше задається в DevOps — "З чого почати?".

Особливо тим, хто тільки познайомився з поняттям циклу SDLC, можна легко загубитися в цьому нескінченному списку термінів, технологій та інструментів. DevOps, GitOps, DevSecOps, MLOps, ClickOps — тут так багато Ops. Адже ти можеш бути світчером (від англійської switch), тобто тим, хто переключається, наприклад, з розробника, сисадміна на тестувальника в DevOps. Студентом-новачком, який тільки починає свій кар'єрний шлях, або навіть досвідченим IT спеціалістом, що розширює своє професійне портфоліо.

Для будь-якого початківця в DevOps важливо з перших кроків визначитися у своїй професійній специфіці. Тобто що тобі ближче? Автоматизація, інфраструктура, системи обробки даних, безпека, або як, наприклад, особисто мені – хмарні рішення та Kubernetes. Сьогодні запит на DevOps сильно перевищує пропозицію. Ринок диктує умови. І найкращою відповіддю на питання "з чого почати", буде знайомство з реальними вимогами на ринку та конкретними кроками твого навчального шляху.

Якщо сильно спростити поняття DevOps, від узагальненого і досить розмитого терміну культура, все зводиться до двох основних якостей — **навички комунікації та володіння інструментами**. Це, так звані на сленгу, софт і хард скіли. У перекладі: персональні навички, не пов'язані з професією, — soft skills та професійні навички — hard skills.

Достатньо всього двох? спитаєте ви. Для швидкого старту достатньо. І зараз я це доведу! Я проаналізував перших 10 вакансій DevOps інженер з відкритих джерел. Організував їх за вимогами та функціями. І ось що в мене вийшло. В 9-ти з 10 випадків вакансія на першому місці включає перелік вимог до інструментів та технологій. Найчастіше це вимога мови програмування, що корелює з етапом Coding: скриптові мови, bnash, часто python, також люблять golang. Тут йдеться про рівень, достатній для нескладних системних утиліт та скриптів.

Далі – це досвід роботи із системами CICD. Наприклад, jenkins або хмарними системами. Ці етапи стосуються Build - Test - Delivery і все це стосується пайплайнів. Іноді DevOps асоціюється саме з цим етапом. Це сильно звужує масштаб та ефективність застосування найкращих практик, по суті, до експлуатації специфічного інструменту. Ще це називають DevOps as a Tools - DevOps як інструмент.

Наступним йде вимога до досвіду застосування методів та технологій деплойменту, тобто розгортання. Етап Deployment найпопулярніший етап серед спеціалістів на сьогодні. В основному завдяки хмарним провайдерам, контейнерам, екосистемі Kubernetes та останнім часом усьому, що пов'язано з безпекою.

Цінується навичка моніторинг та траблшутингу (перекладається, як вирішення проблем). Тут все частіше використовуються системи на базі штучного інтелекту, що не тільки прискорюють пошук джерела проблеми, а й передбачають інциденти на основі аналізу аномалій. Але інженерна думка та досвід завжди спрацьовують в найскладніших ситуаціях. Адже чи може передбачити штучний інтелект до чого призведе апгрейд проду в п'ятничний вечір?

На другому місці вимоги до ваших soft skills, а саме — навичок комунікації. Іноземні мови, вміння спілкуватися з колегами по команді та замовником, презентувати себе як фахівця, прийняття різноманітних культурних, загальноприйнятих стандартів та суспільних принципів у колективі.

У твоєму запасі очікуються щонайменше дві мови. Одна із них іноземна, а друга — мова програмування. Іноземна мова – це твоя комунікація, читання, навчання та обмін інформацією. А ще вивчення мов позитивно впливає на мозкову активність – феномен нейропластичності.

Як показує досвід, найчастіше саме софт скіли, не дивлячись на їхнє друге місце у списку вимог, є основними при ухваленні рішення на скринінгах та співбесідах. Скринінг — це варіант експрес співбесіди, де буквально за кілька хвилин обидві сторони знаходять або не знаходять загального розуміння. Так званий, "match" або збіг вимог замовника до вашого досвіду, технологічного стеку та побажань. Почніть з аналізу вакансій, відгуків та проходження скринінгів прямо сьогодні. Це найефективніший спосіб тренування комунікації. Озброївшись олівцем, вже через пару-трійку інтерв'ю у вас буде готовий список тем для підготовки, питань з технологічних трендів та потрібних інструментів.

А зараз, гадаю, саме час навести пару прикладів, з чого тобі почати на кожному етапі DevOps Infinity Loop, а також поповнимо наш список термінів.

Як ми уже розглядали, на **етапі планування** визначаються архітектура, патерни та інструменти. Це виглядає як список того, що тобі буде добре подивитись-почитати, а згодом і попрактикувати на перших порах.

На етапі Coding DevOps передбачає розуміння базових принципів та підходів розробки софту. Вміння читати, а краще, і писати код, стирає межі та перешкоди в комунікації команди розробників та експлуатації — в принципі, основна причина виникнення DevOps.

Почніть з bash. Інтерпретована або ще мова скриптів. Служить для виконання завдань з автоматизації сценаріїв. Часто використовується у пайплайнах. bash є дуже потужна та глибока мова. І найчастіше завдання на написання простого циклу вже ставить новачка в глухий кут. Тому тренуй скрипти де тільки це можливо – це сильна сторона DevOps.

Якщо bash — це командно-сценарна мова, то Python — це майже всі типи завдань DevOps. Скрипти, бібліотеки та сервіси. Навіть базові знання python, скриптової універсальної мови, додадуть значний "плюс" до твого резюме. Якщо у тебе виникло питання Пайтон або Пітон - звернися до історії створення мови та шоу «Літаючий цирк Монті Пайтона».

Go lang — компілювана, багатопоточна мова, розроблена Google. Чому така популярна серед DevOps? Мова з відкритим кодом, швидкий поріг входу, швидка компіляція та виконання, просте

управління залежностями, підтримка мікросервісів. На Go написаний Docker, Kubernetes, Terraform, GitHub, Etcd та багато інших інструментів екосистеми DevOps.

Перший крок на цьому етапі — почни писати простий код вже сьогодні, а завтра це точно стане у нагоді.

Етап Build. Одвічне спірне питання — хто має писати пайплайн? Розробник, який пише код, добре знає як він повинен збиратися і тестуватися? Або інженер DevOps, який добре знає інструменти побудови пайплайну. Тут варто відзначити, незалежно хто пише пайплайн, базові знання мови проєкту та інструментів автоматизації дає вам особливу перевагу. Почніть з налаштування простого пайплайну — це дасть вам можливість краще розібратися у специфіці продукту. Наприклад, реалізуйте security scanning для пошуку сенсетів даних в коді продукту у відкритому вигляді — це буде значний внесок з боку DevOps.

Етап контролю якості або QA. Ми з вами говорили на минулому занятті про безперервний процес контролю якості на всіх етапах розробки. Почніть колаборацію з командо qa на цьому етапі. Ви можете значно розширити можливості проведення тестів за рахунок ваших знань про інфраструктуру та інструменти. Наприклад, це допоможе в реалізації принципу Fail Fast (швидкого автоматичного виявлення помилок в коді) — що з боку QA значно підвищить якість коду і зменшить кількість допущених помилок за рахунок людського фактора. А розробникам це надасть свободу для експериментів.

Доставка та деплой. Це прямо перевага DevOps. Весь хайп промисловості зосереджений прямо тут. І все завдяки мікросервісам, контейнерам та екосистемі Kubernetes. Тут збудовано

знамениту стіну, через яку розробники перекидають, найчастіше, сирий додаток, не дбаючи про його подальшу долю. А інженери експлуатації на свій страх і ризик намагаються цю програму розгорнути, при цьому не зламавши те, що вже працює. У результаті одні поспішають запустити в продакшн зміни, коли другі, керуючись принципом працює — не чіпай, не поспішають що-небудь змінювати. DevOps просто народився тут. Усі принципи, практики та інструменти спрямовані на виправлення цієї ситуації. Почни з Container Driven Collaboration, або просто почни доставляти продукт контейнерами. Цей процесс зробить перший крок до ефективної колаборації в команді.

Моніторинг та зворотній зв'язок. І ось наш додаток вже у продакшн. Життєвий цикл продовжується. Зрозуміти чи працює саме так, як задумано розробниками та замовником продукту, допомагають метрики. Усуненню несправностей сприяє логування. Про методи та інструменти цих процесів ми детально говоритимемо на практичних заняттях. Зараз варто зауважити, що зворотний зв'язок, як від цих систем, так і безпосередньо від користувачів, обов'язково обробляється і візуалізується. Збираються технічні метрики. На основі технічних створюються бізнес метрики. Життєвий цикл та стан систем стає доступним як для експлуатації, так і для розробників продукту. Зворотній зв'язок виражається у вигляді RFC (запитів на зміни та додавання нової функціональності), багів (помилок у логіці або синтаксисі коду програми) та хотфіксів (гарячих заплаток або швидкого виправлення якоїсь конкретної помилки). Почни з аналізу вже готових логів та метрик клауд провайдера. Збери основні показники на свій перший дашборд та налаштуй повідомлення, наприклад, про недоступність системи.

Як ви вже знаєте, цикл тут замикається. Етап зворотнього зв'язку перетворюється на етап планування. А DevOps, своєю чергою, починає новий цикл. І так до умовної нескінченності,

підвищуючи ефективність, покращуючи якість, доопрацьовуючи методи та перетворюючи новий досвід на кращі практики. У нашому випадку, початківець у DevOps, отримавши інформацію для роздумів, робить висновки. Який саме етап йому цікавіший. На чому варто сфокусуватись. В який бік розвиватися та поглиблювати свої знання. А найголовніше — розуміє з чого почати.