Лабораторна робота 6. Створення Azure DevOps проекту. Pipeline для мікросервісів. Додавання підтримки Docker в рішення.

Мета. Вивчити процес взаємодії мікросервісів, освоїти побудову додатків з мікросервісною архітектурою в Visual Studio 2019.

Завдання: створити проект Azure DevOps, розробити pipeline для мікросервісів, додати підтримку Docker в рішення.

Порядок виконання

Щоб отримати максимальну користь від рішення для мікросервісу, необхідно прийняти процес, який дозволяє автоматично будувати, тестувати та розгортати мікросервіси автоматизованим способом. Існує так багато різних платформ і технологій, які дозволяють зробити саме це.

Процес безперервної інтеграції або CI має важливе значення для успішного рішення для мікросервісів.

У цьому курсі ми використовуватимо Azure DevOps, який містить різні сервіси, такі як Azure Repos та Azure Pipelines (https://azure.microsoft.com/en-us/services/devops).

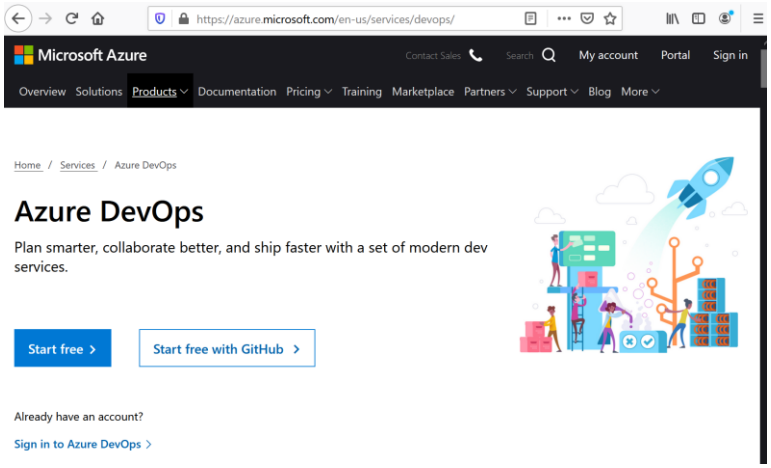

Рис. 1. Сайт Microsoft Azure DevOps

DevOps має таку ілюстрацію.

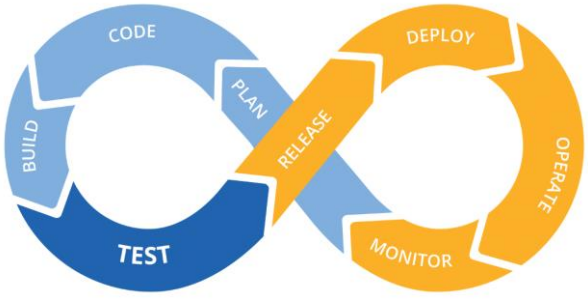

Рис. 2 DevOps

CI/CD розшифровується як Continuous Integration і Continuous Delivery - тобто безперервна інтеграція і безперервна доставка. Спрощено, CI – це синя петля на зображенні, а CD – жовта петля на зображенні.

Слово pipeline означає «конвеєр» (пристрій для безперервного переміщення оброблюваного виробу від одного робітника до іншого).

Що цікаво про Azure DevOps, це те, що він включає в себе безліч вільних функціональних можливостей та час на комп’ютер.

Крім того, ви можете використовувати Azure DevOps для створення та розгортання програмного забезпечення, написаного на будь-якій мові та будь-якій платформі.

Тож почнемо. Рішення, розроблене в попередній лабораторній роботі, розмістіть в репозиторії на GitHub.

Далі зайдіть на сайт Azure DevOps, натисніть Start free. Війти можна з свого корпоративного облікового запису (account).

Після входу введіть ім'я проекту, задайте доступність, систему контролю версій і т.д.

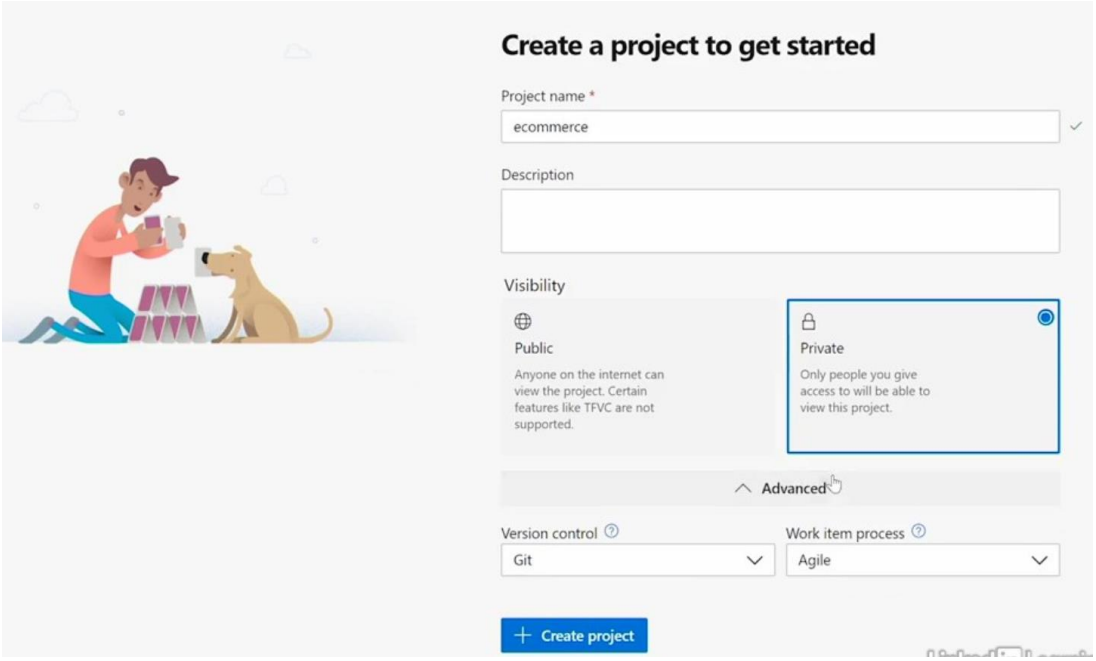

Рис. 3. Створення проекту в Azure DevOps.

Натисніть Create project. З'явиться панель управління проекту.

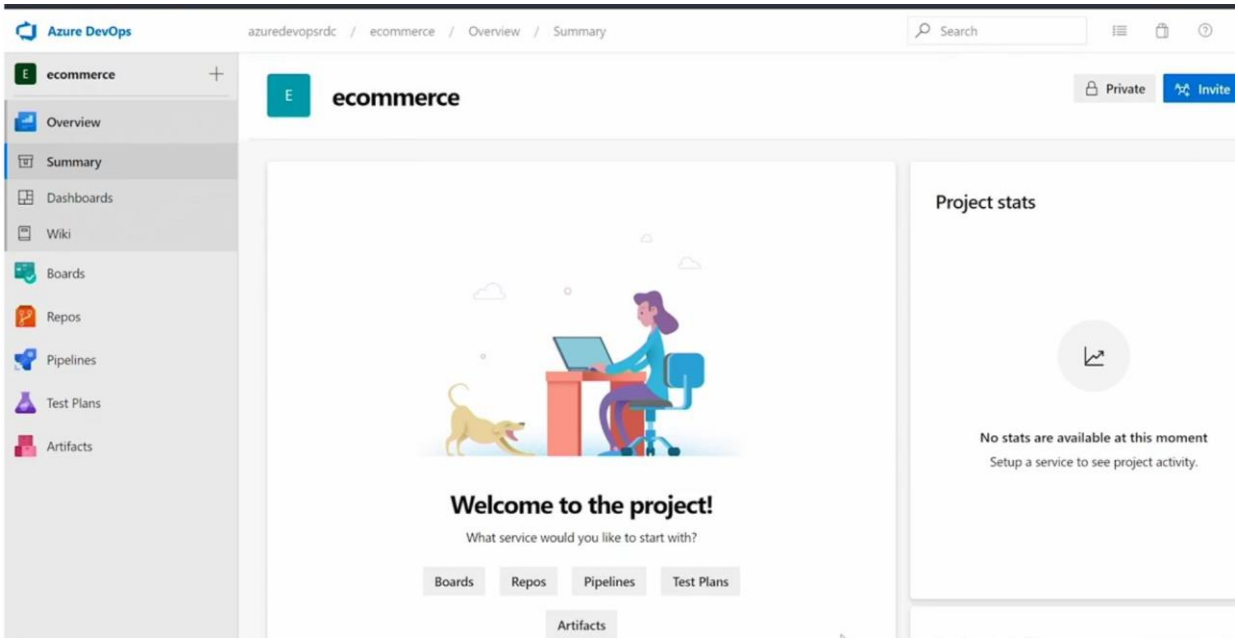

Рис. 4. Панель управління проекту ecommerce.

А тепер, коли проект створено, давайте імпортуємо Repos з GitHub. Вам не потрібно обов'язково мати свій вихідний код у Azure Repos, але зараз наша мета - показати, як репозиторій з вихідним кодом можна інтегрувати з Azure Pipelines, щоб створити конвеєр безперервної інтеграції.

Отже, давайте продовжимо і натиснемо на опцію Repos.

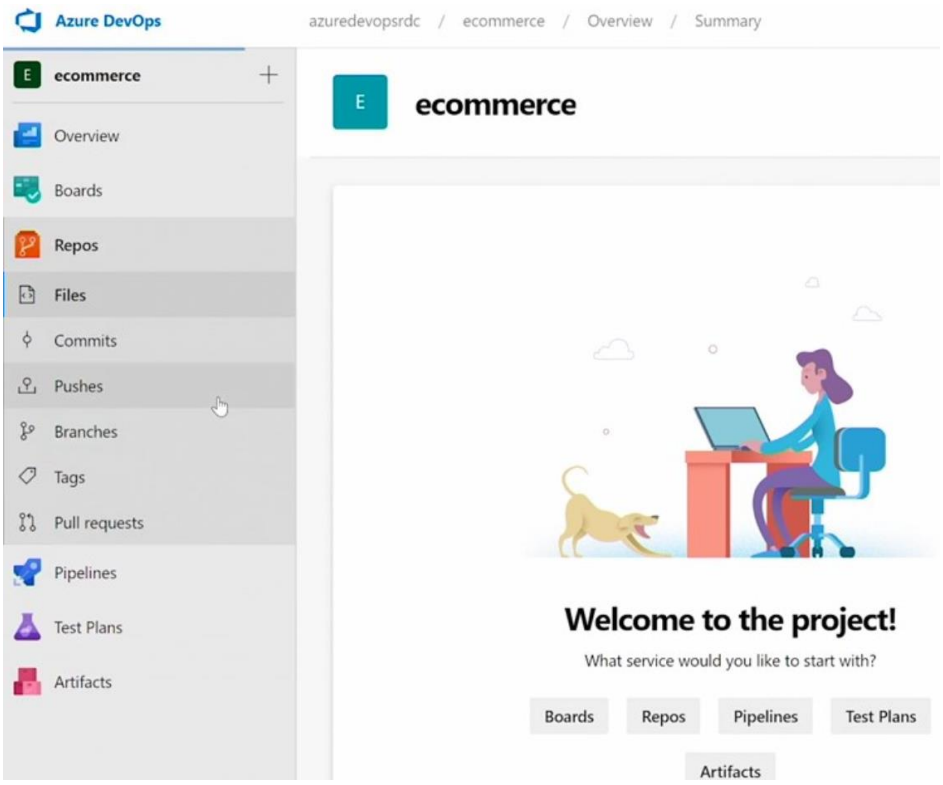

Рис. 5 Розкритий пункт Repos

Натисніть пункт Pushes.

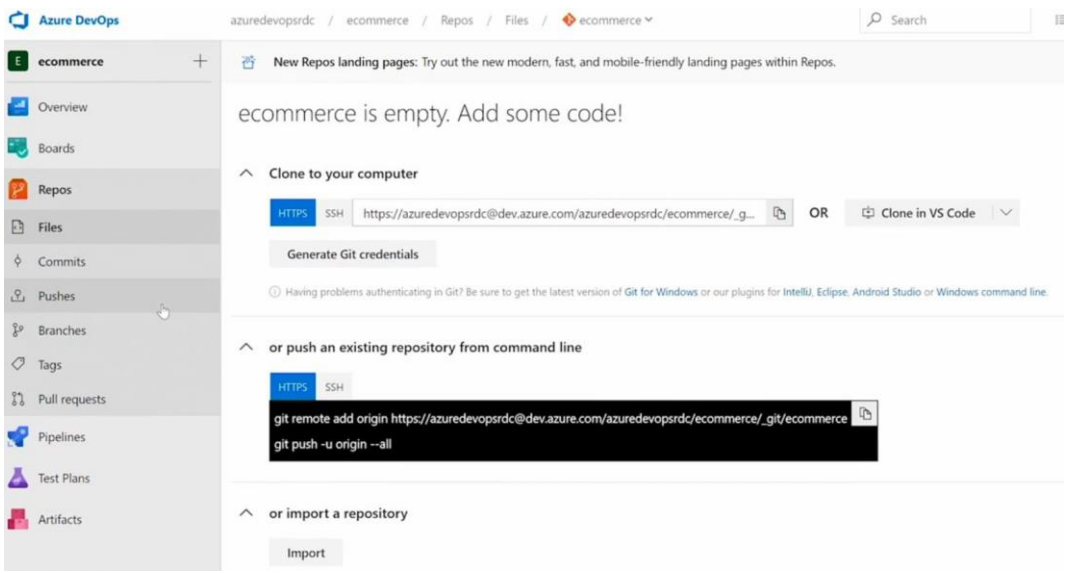

Рис. 5 New Repos landing page
Далі натисніть кнопку Import. З’явиться діалогове вікно.

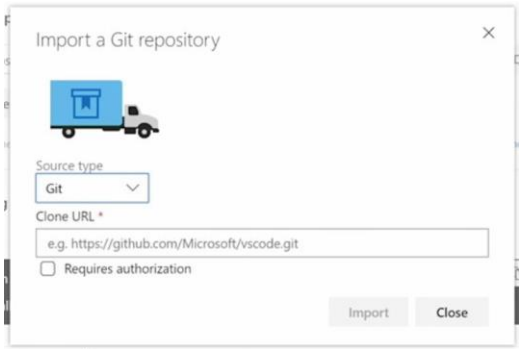

Рис. 6. Вікно імпорту репозитарію.

Тут встановіть адресу репозиторію, який ви хочете імпортувати, і натисніть кнопку Import. Коли з’явиться діалогове вікно Import successful! – натисніть Click here to navigate to code view. З’явиться ваш код.

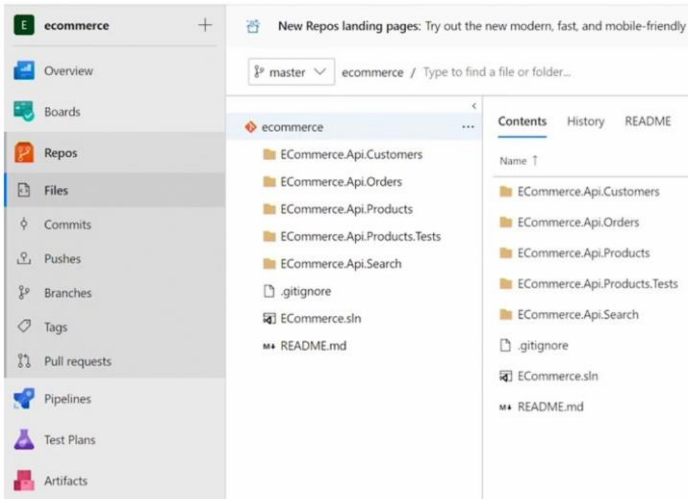

Рис. 7. Код завантажено в Repos.

Тепер, коли у нас є код у Azure Repos, давайте створимо новий конвеєр побудови (pipeline). Конвеєр побудови дозволяє автоматизувати тести та складання (builds) для вашого проекту.

Отже, давайте створимо новий pipeline. Натисніть в панелі управління Pipelines, а потім Builds. З'явиться вікно.

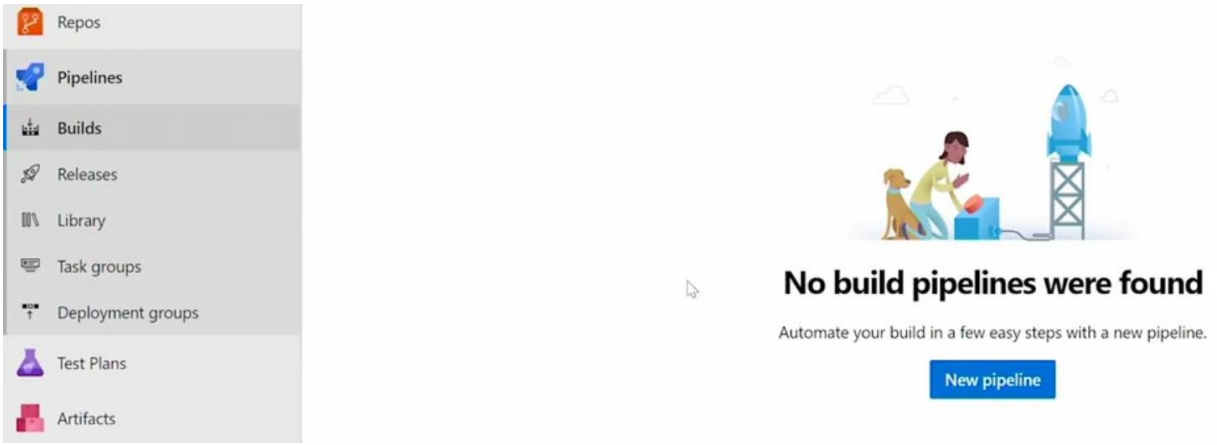

Рис. 8. Вікно повідомлення.

Натисніть кнопку New pipeline.

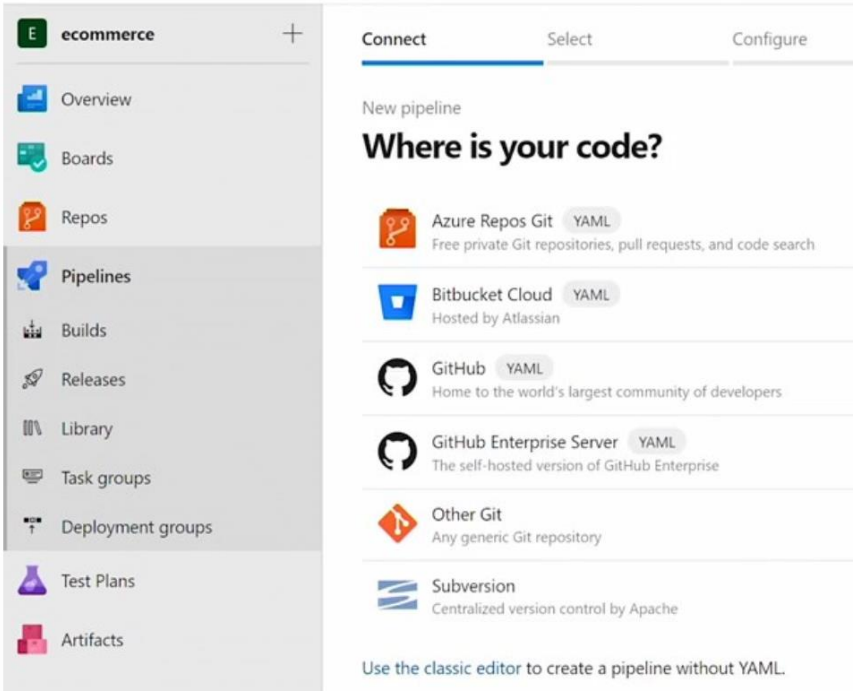

Рис. 9. Вікно Where is your code?

На цій сторінці ми будемо використовувати параметр класичного редактора. Тому натисніть Use the classic editor. З'явиться вікно.

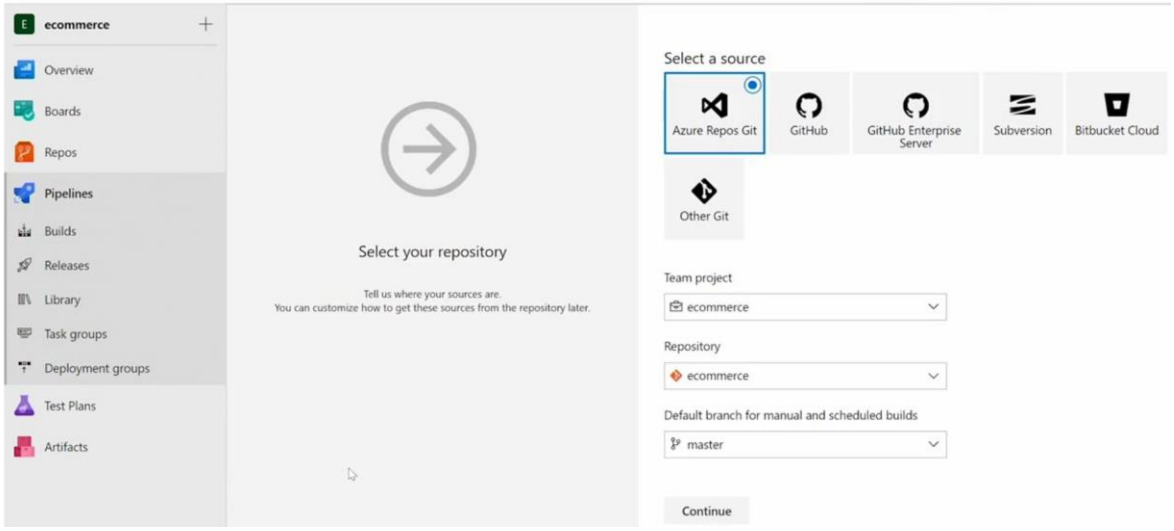

Рис. 10. Вікно вибору репозитарію.

І як ви бачите, існують різні джерела, які ви можете вибрати для свого конвеєра побудови (pipeline). У цьому випадку виберіть Azure Repos Git, тому що саме там знаходиться вихідний код, але ви також можете використовувати й інші джерела. Давайте натиснемо на Continue. З’явиться вікно.

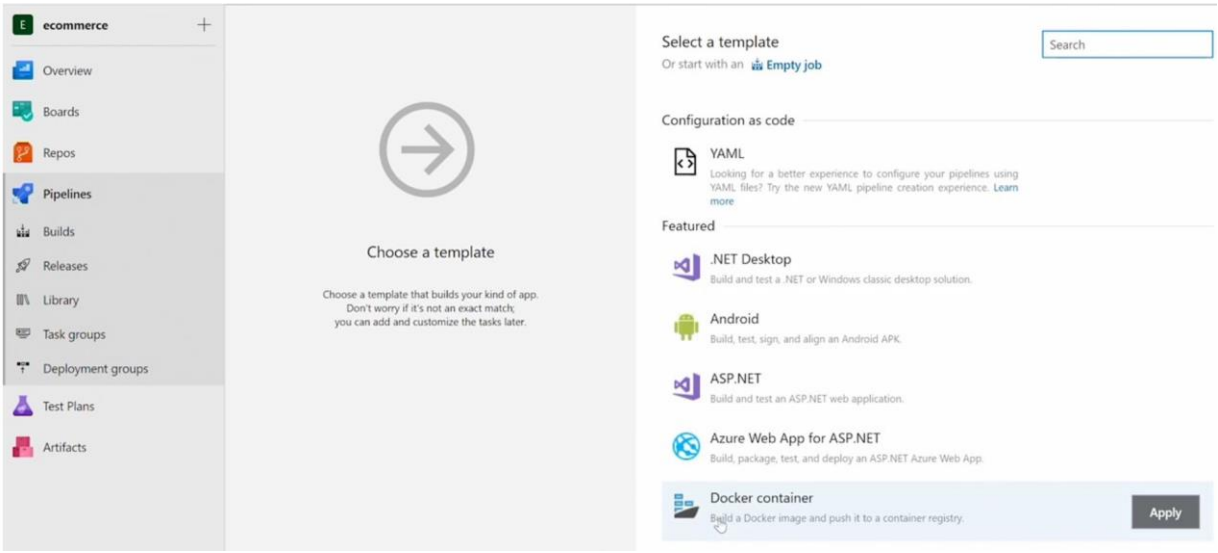

Рис. 11. Вікно вибору шаблона.

Тут ви можете обрати шаблон, який включає всі завдання, які ви хочете виконати у своєму конвеєрі. Давайте шукатимемо .NET Core, оскільки ми створили наші мікросервіси за допомогою .NET Core.

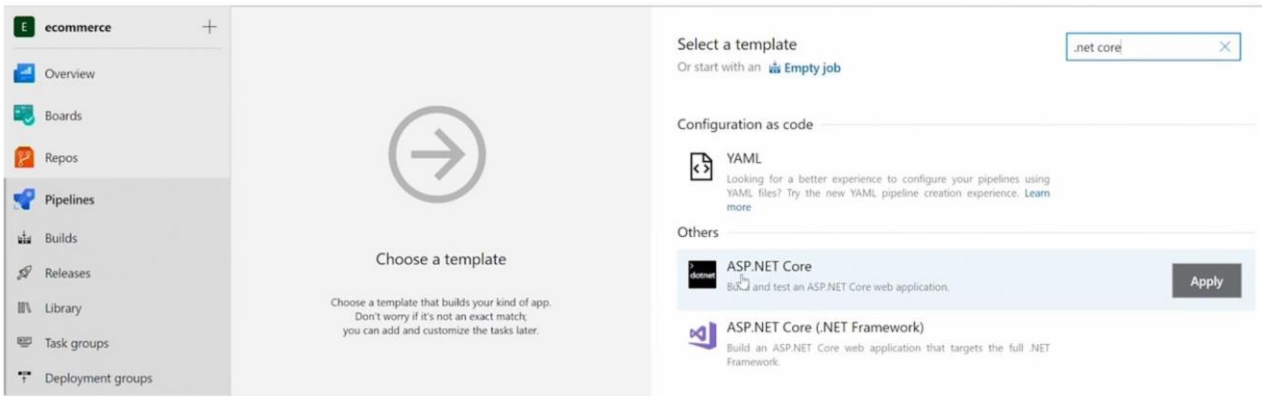

Рис. 12. Пошук .NET Core

Отже, знайдений шаблон є тим, який ми хочемо використовувати. Натисніть Apply. З'явиться вікно.

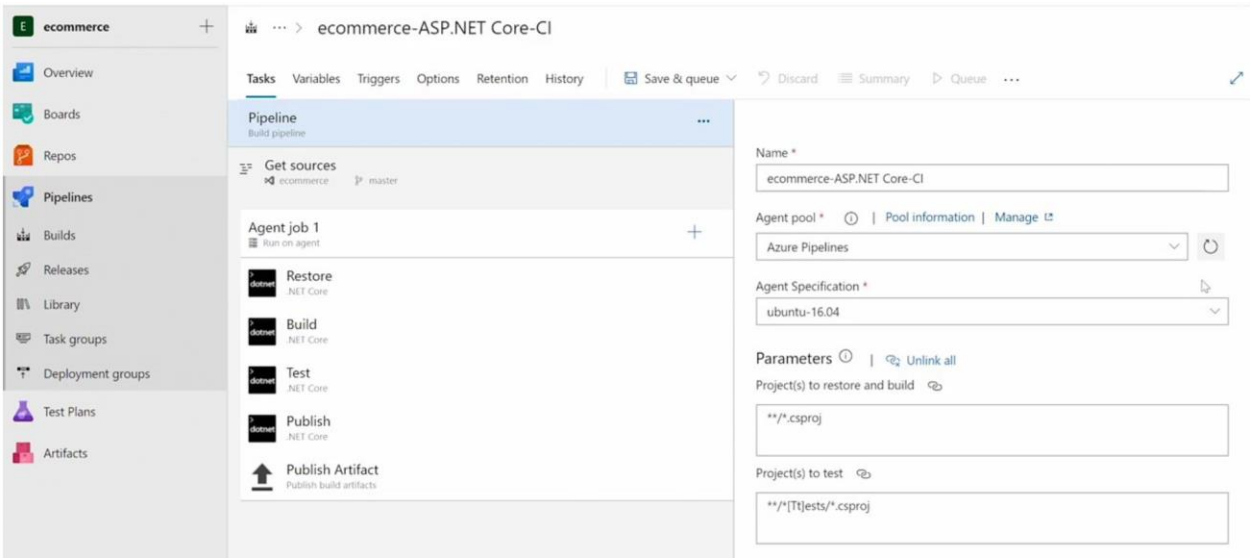

Рис. 13. Вікно pipeline для шаблону .NET Core

І як ви бачите, цей шаблон вже включає деякі кроки, які буде виконано.

У наступному ми налаштуємо цей конвеєр для складання та виконання тестового блоку.

Давайте налаштуємо цей конвеєр. Спочатку нам потрібно вибрати агента, на якому буде збиратися і виконуватися збірку. Оскільки ми використовуємо крос-платформову технологію .NET Core, ми можемо вибрати будь-якого агента (Windows, Linux or Mac OS). Тож залишимо вибрану за замовчуванням платформу Ubuntu-16.04.

Ви можете пам'ятати, що наш unit-тестовий проект має в своїй назві суфікс Tests. У цьому ж розділі можна вказати шаблон імен для тестування проектів.

В вікні в рядку Agent job 1 натисніть «+» (Add a task to Agent job 1) щоб додати нове завдання.

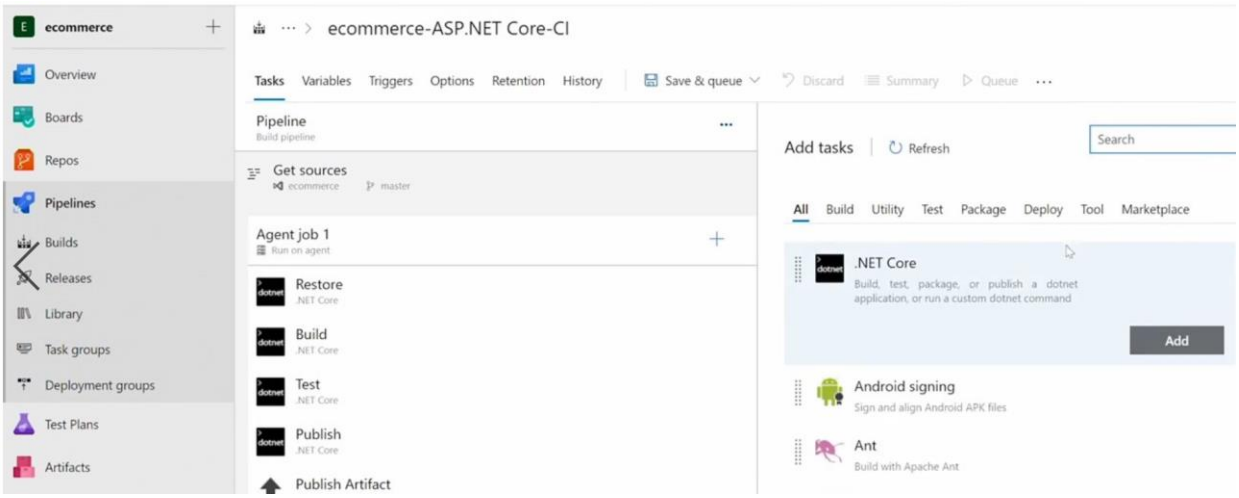

Рис. 14. Додавання нового завдання.

Ми маємо переконатися, що потрібна версія .NET Core встановлена в агенті, який ми обрали перед запуском тесту. З цієї причини давайте додамо нове завдання в конвеєр встановити правильний SDK в агенті.

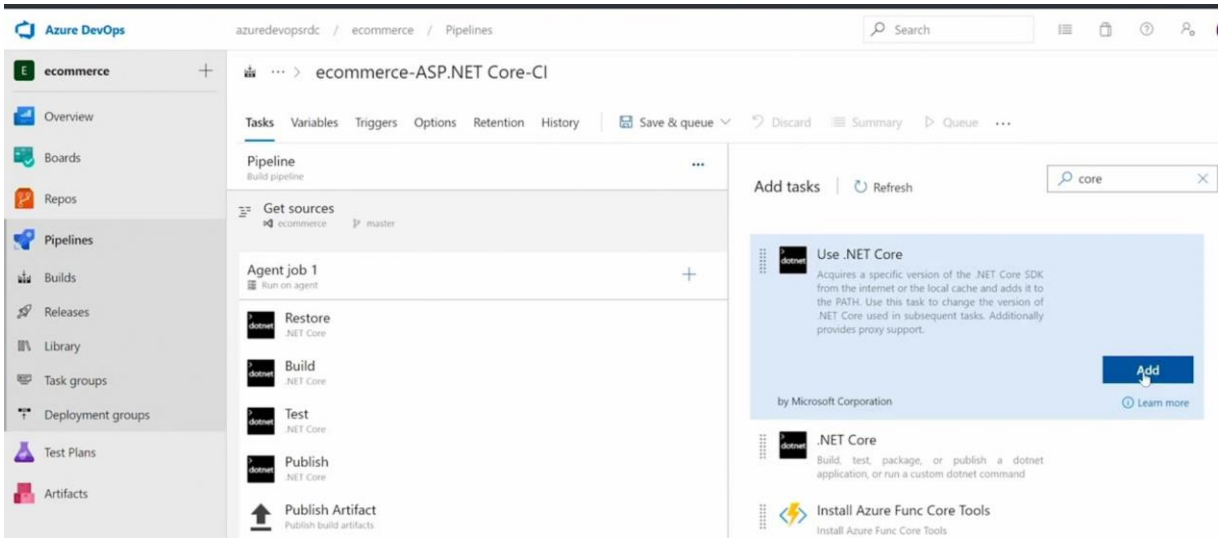

Рис. 15. Додавання в конвеєр Use .NET Core.

Тож додамо Use .NET Core і піднімемо (drag and drop) це завдання вверх.

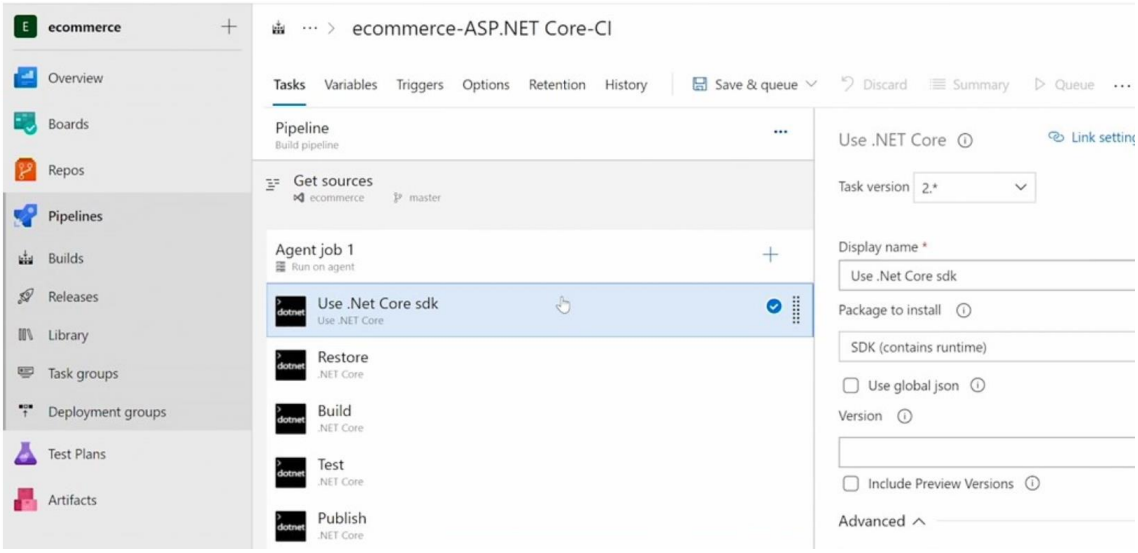

Рис. 16. Переміщення завдання вверх.

Більше того: задайте мінімальну версію .NET Core 3.1.x

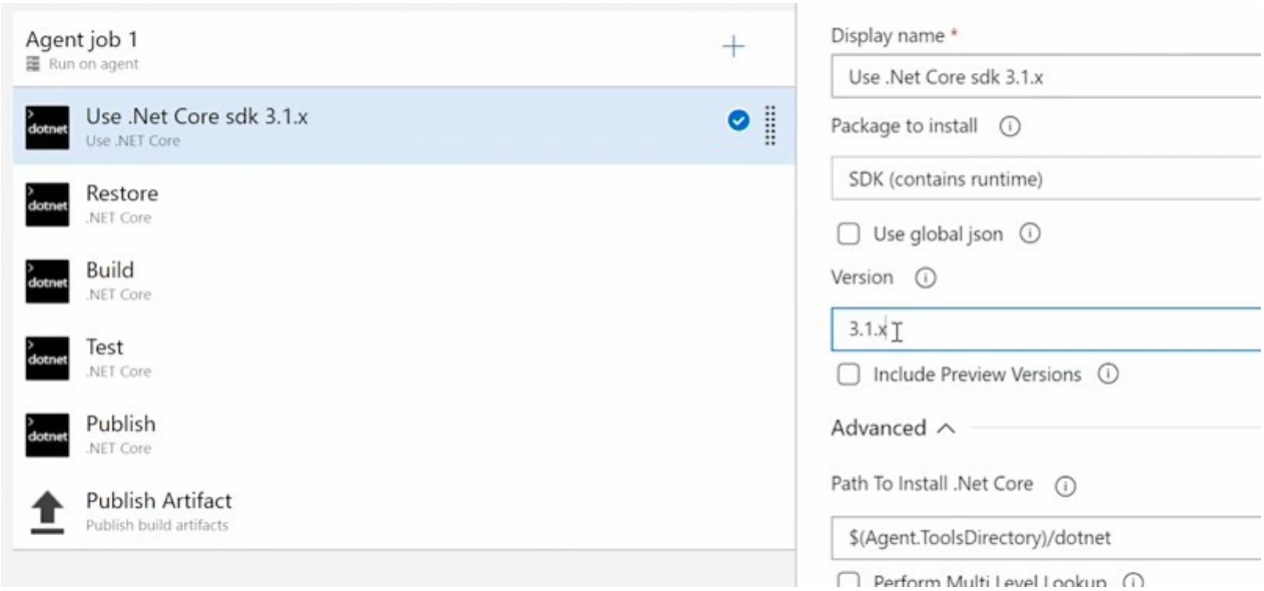

Рис. 17. Визначення мінімальну версію .NET Core

Конвеєр (pipeline) - це послідовність дій, які будуть виконуватись одна за одною. Отже, після встановлення SDK конвеєр відновить усі пакети NuGet, потім збиратиме програми, потім виконує всі unit тести, а потім виконує команду Publish. Нарешті, збирається опублікувати артефакти. Артефакт - це компонент вашої програми, що розгортається.

Тож давайте змінимо ці завдання публікації. Тиснемо Publish. Тому ми знімаємо прапорці з цих веб-проектів, оскільки ми не маємо веб-проектів. У нас немає веб-додатків всередині нашого рішення, а лише мікросервіси.

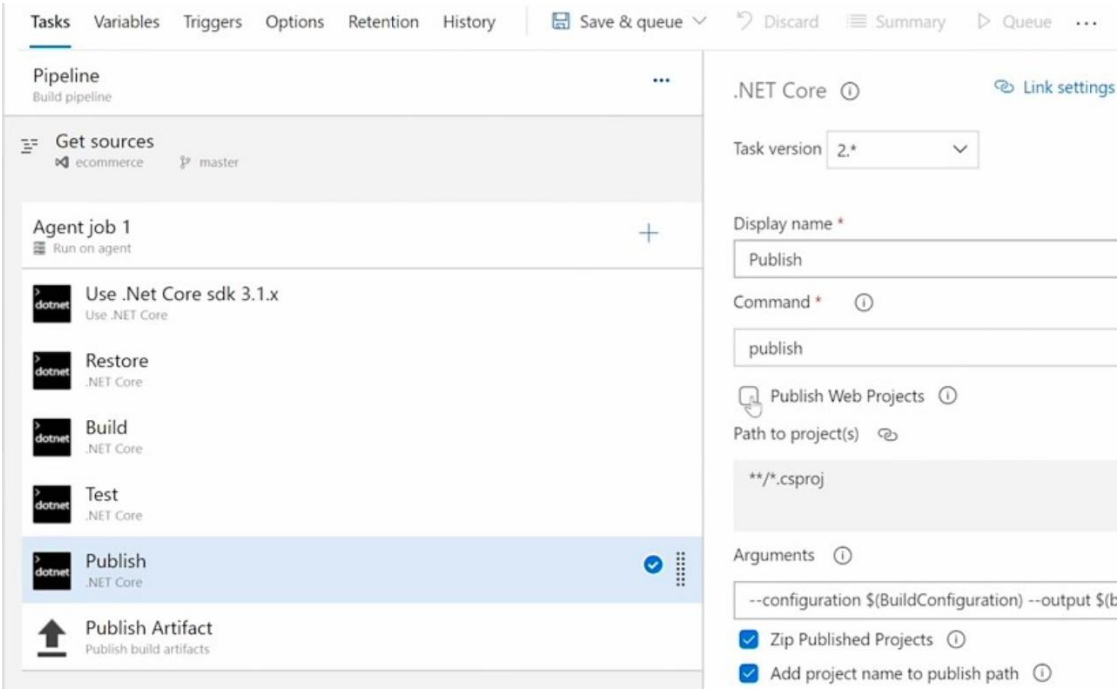

Рис. 18. Зняття прапорця Publish Web Projects.

Далі натисніть Save $ queue і потім ще раз Save $ queue. З’явиться вікно.

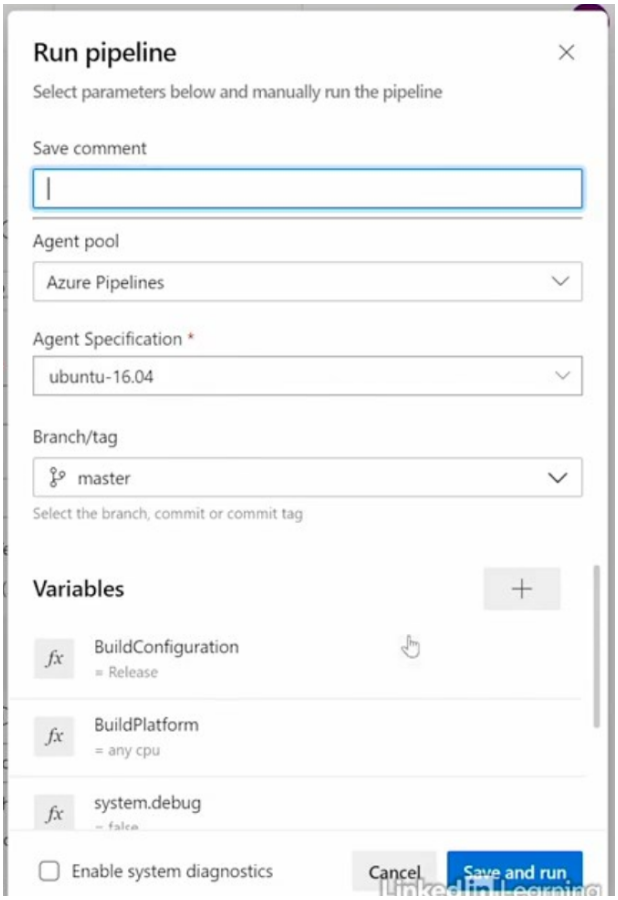

Рис. 19. Вікно Run pipeline.

Натисніть кнопку Save and run. Тепер pipeline виконує всі завдання в заданій послідовності. Через деякий час всі завдання закінчені, і ми можемо побачити результати побудови pipeline. У цьому випадку це було успішним.

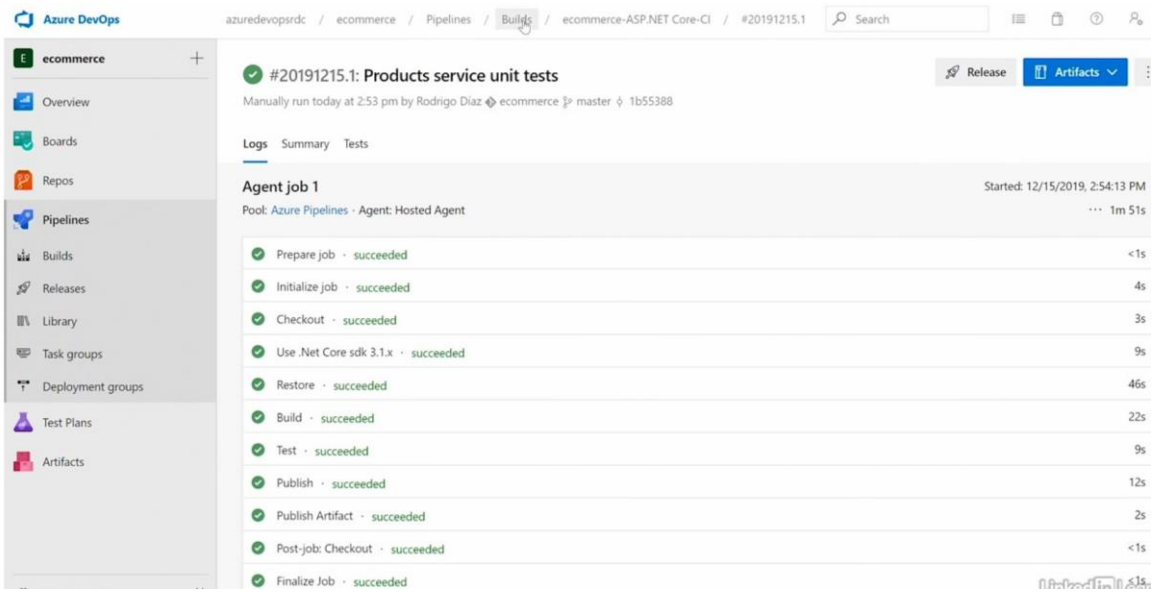

Рис. 20. Результати виконання конвеєру.

Перш ніж закінчити цей розділ, давайте розглянемо, як ми можемо включити постійну інтеграцію (CI – continuous integration). Натисніть в верхніх вкладках Builds.

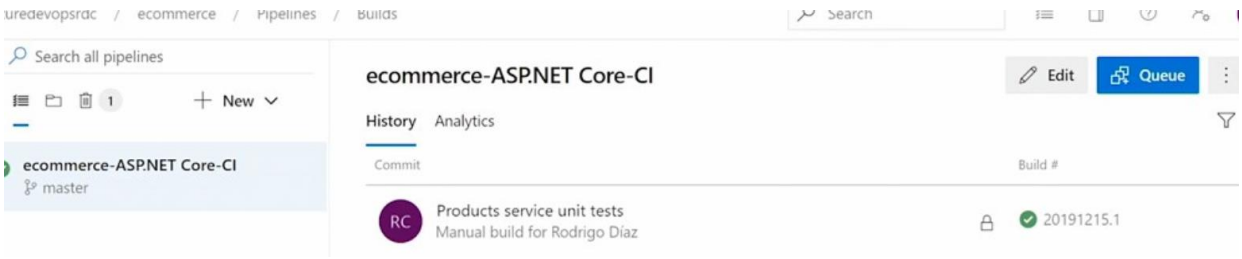

Рис. 21. Вкладка Builds

Давайте натиснемо на кнопку редагування (Edit) і в вікні, що з’явилось виберемо опцію Triggers.

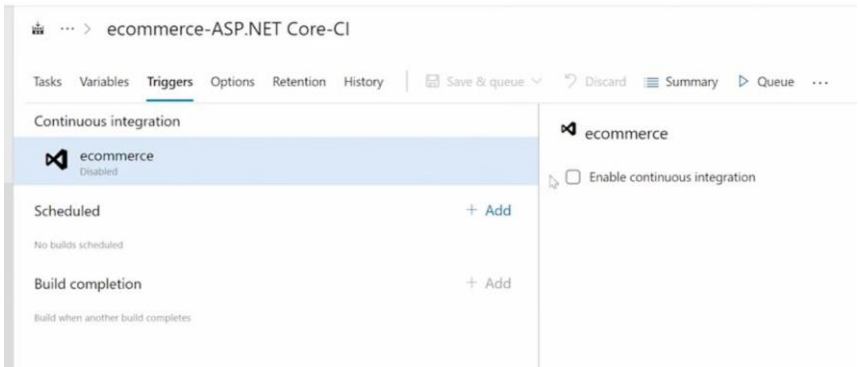

Рис. 22. Опція Triggers

І тут у варіанті Triggers ви можете знайти прапорець Enable continuous integration (Увімкнути безперервну інтеграцію), де ви можете включити ці функціональні можливості.

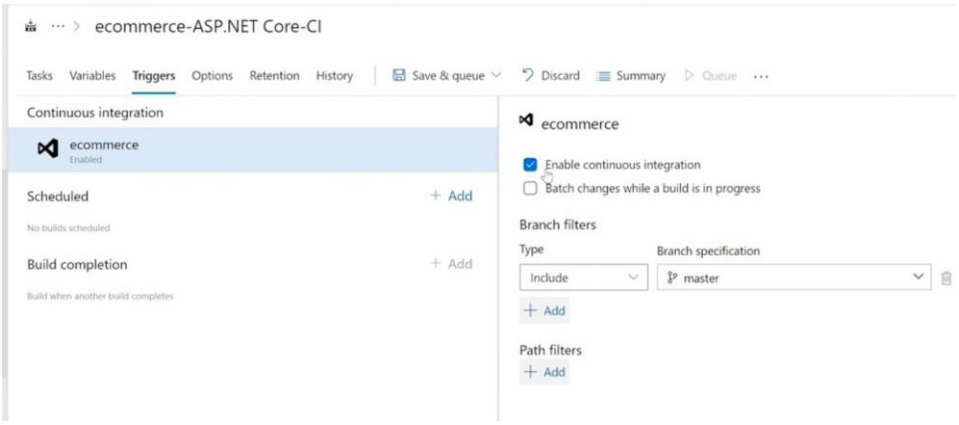

Рис. 23. Увімкнення безперервної інтеграції

Тож знов натисніть Save $ queue і потім Save і ще раз Save.

І давайте перевіримо безперервну інтеграцію лише внісши зміни, наприклад і файл README нашого рішення. Тож натисніть Repos, виберіть серед вихідного коду файл README, додайте довільний рядок, натисніть Commit, а потім в діалоговому вікні – ще раз натисніть Commit.

І якщо ми повернемось до конвеєру побудови (натиснувши Pipelines, Builds), ми можемо побачити, що друге виконання автоматично виконувалося функціоналом безперервної інтеграції.

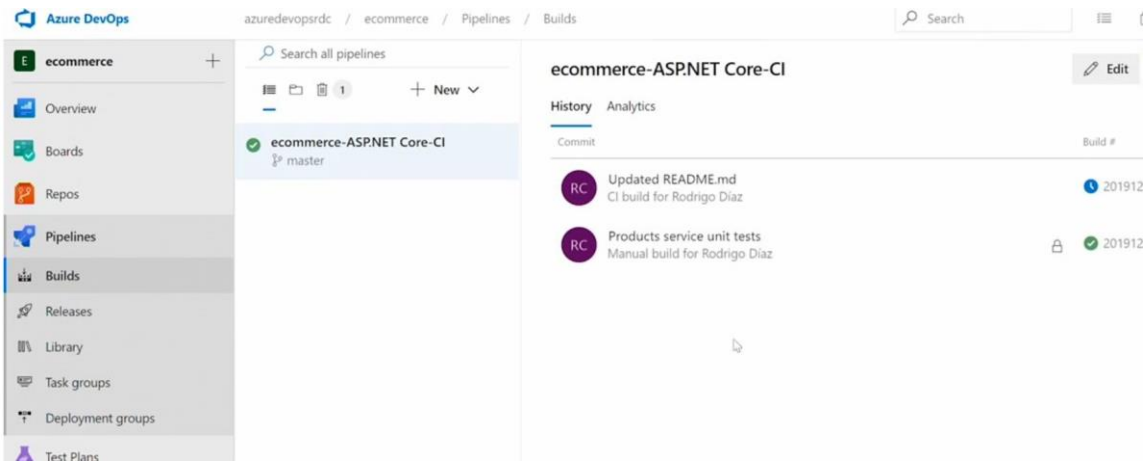

Рис. 24. Перевірка роботи безперервної інтеграції.

Можливість автоматичного складання коду та виконання unit тестів - лише частина рішення. Ви можете створити конвеєри для розгортання (deploy) готових компонентів. Ви навіть можете автоматично запустити конвеєри, коли з'явиться новий артефакт збірки.

Щоб створити новий конвеєр випуску (release pipeline), вам потрібно вибрати параметр Releases під пунктом меню Pipelines і потім натиснути кнопку New pipeline. З’явиться вікно.

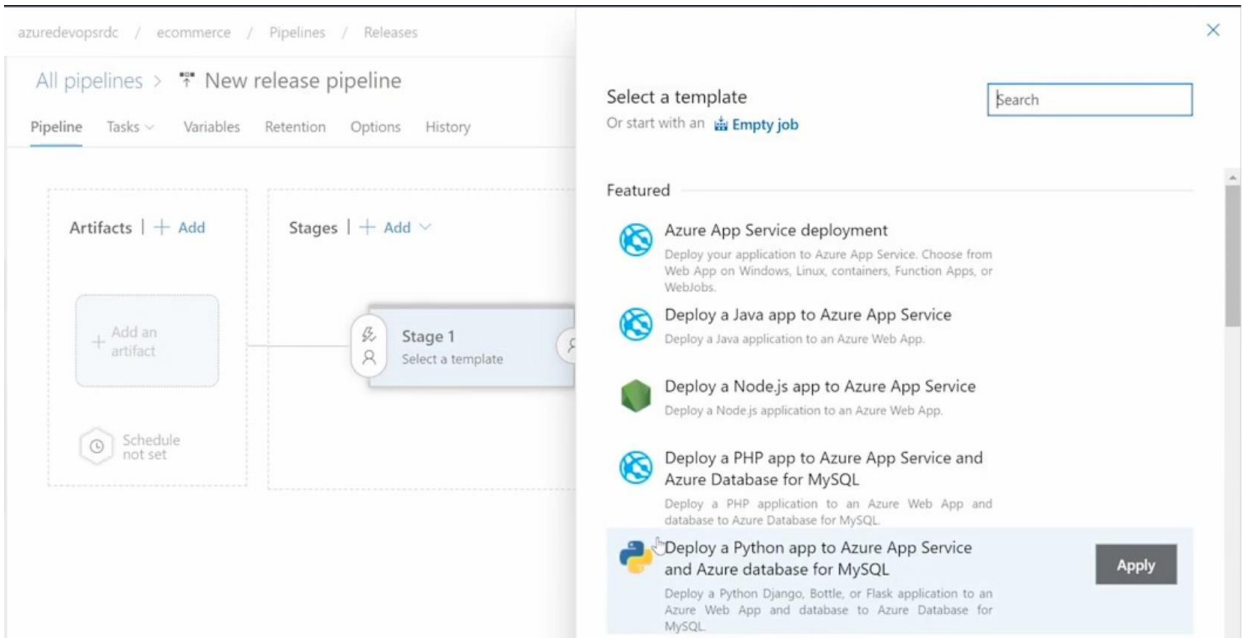

Рис. 25. Вибір шаблону для release pipeline

І тут, ви можете почати з використання шаблону або ви можете почати з порожнього завдання. Існують десятки шаблонів для загальних сценаріїв та технологій розгортання. Виберіть, наприклад, Empty job. З’явиться діалог.

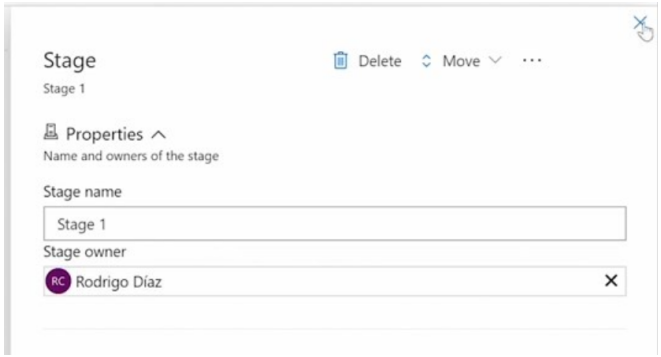

Рис. 26. Початковий діалог для пустого шаблона.

Закрийте поки що цей діалог. В нижньому вікні в блоці Artifacts натисніть Add an artifact. Ви можете вибрати різні види артефактів.

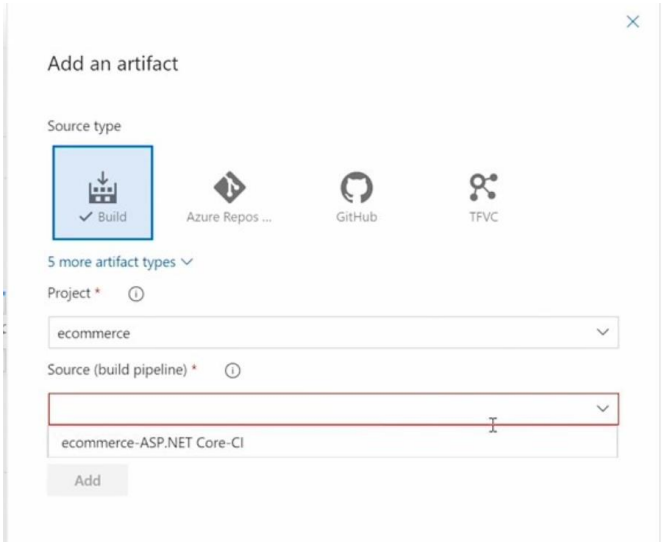

Рис. 27. Діалог додавання артефакту

Наприклад, ви можете взяти той, який ми раніше створили в конвеєрі збірки, як артефакт цього release pipeline.

Таким же чином ви могли використовувати Repos як артефакт. Це стане в нагоді, коли вам потрібні лише ***файли сценаріїв*** з інструкціями щодо розгортання. Саме так ми будемо робити з файлами Docker у наступному розділі. А поки що закрийте цей діалог.

Якщо ви обрали шаблон або вирішили почати з порожнього завдання, ви можете налаштувати конвеєр на сторінці визначення випуску. В нижньому вікні натисніть на Stage 1.

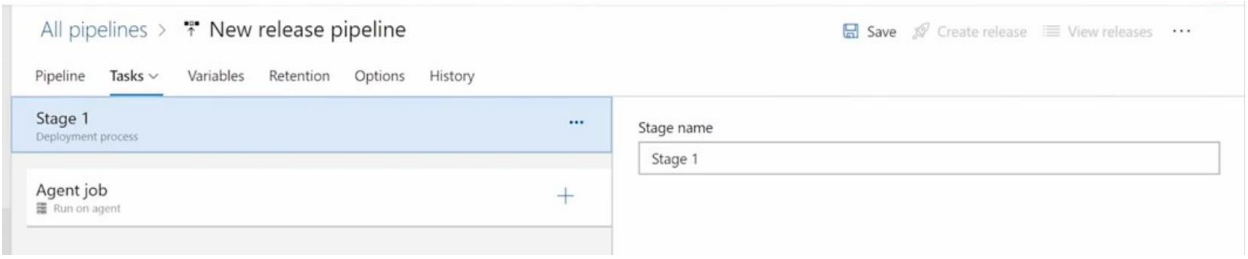

Рис. 28. Pipeline release definition page

На цій сторінці ви можете налаштувати конвеєр. Дуже схоже на те, що ми робили в *конвеєрі побудови*, ви можете зробити тут в *конвеєрі виконання*, вказавши набір завдань для виконання в послідовності. Натисніть «+», і ви побачите безліч елементів для додавання в вашу послідовність.

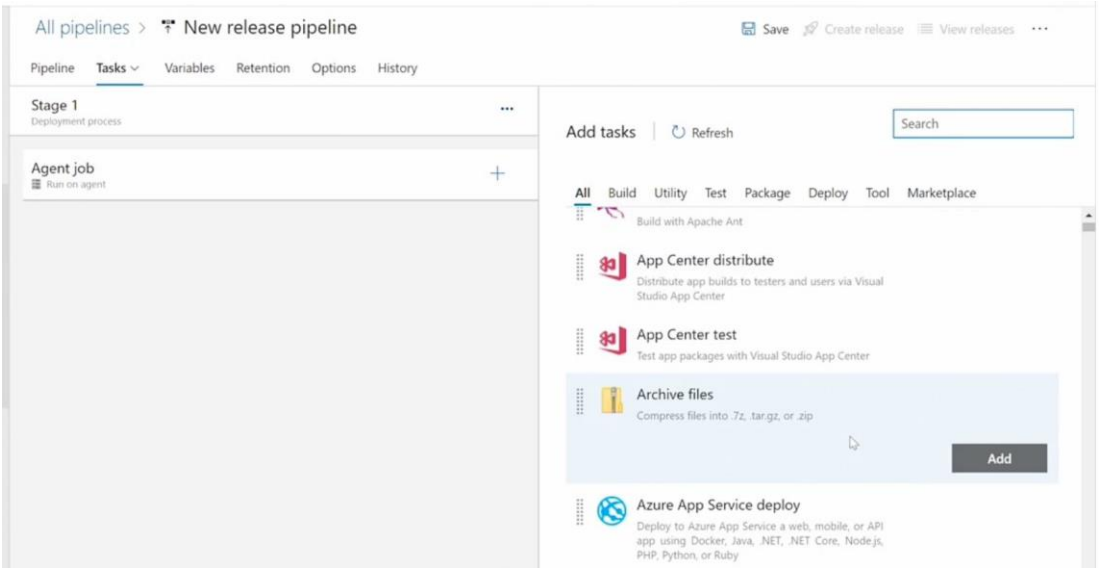

Рис. 29. Список можливих завдань для *конвеєра виконання*

Як бачите, є сотні завдань, що попередньо збудовані (pre-build), які ми можемо додати до конвеєра. Це залежатиме від ваших вимог до розгортання.

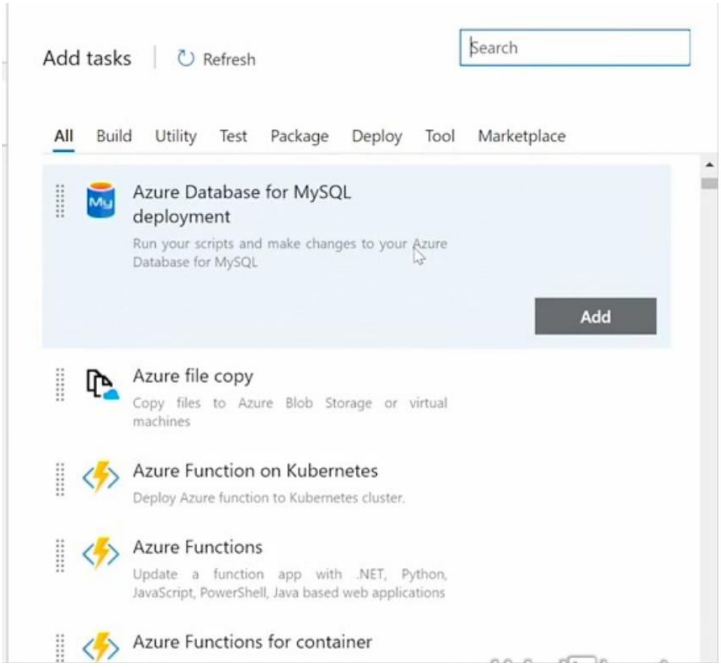

Рис. 30. Можливі варіанти розгортання

А поки що, переглянувши можливі варіанти, закриємо цей діалог.

Додавання підтримки Docker до рішення мікросервісів.

Хоча контейнери не є вимогою в архітектурі мікросервісів, вони так досконало доповнюють один одного. Контейнери дають можливість самостійності та ізоляції процесів, які шукають мікросервіси.

Коли ми говоримо про контейнери, то більшу частину часу ми маємо на увазі контейнери Docker. Докер - це і технологія, і компанія.

Хоча контейнери не є чимся новим, вони вже стільки років завойовують занадто велику популярність і тягу в технічному співтоваристві. Ми можемо запускати контейнери Docker у хмарі, на локальних серверах і навіть у невеликих пристроях IoT.

Ця технологія контейнерів тісно інтегрована з інструментами для розробки коду Visual Studio та Visual Studio Code, що дуже корисно, коли нам потрібно налагоджувати локально.

Контейнери засновані на контейнерних образах (container images). Образ - це шаблон, який включає все необхідне для запуску програми, наприклад, необхідний runtime, системні бібліотеки та налаштування. Цей взаємозв'язок схожий на клас та об’єкт. Іншими словами, контейнер - це екземпляр зображення. Докер-контейнери повинні працювати в Докер-хаусі або в контейнерних оркестраторах, таких як Kubernetes, Service Fabric або DC / OS. Рішення, яку платформу ви повинні використовувати, залежатиме від сценарію та проекту, над яким ви працюєте.

У цьому розділі ми додамо підтримку Docker для всіх мікрослужб у рішенні. Крім того, ми збираємося додати підтримку Docker Compose, щоб мати змогу виконувати та запускати цілу групу контейнерів у цілому.

Почнемо з клацання правою кнопкою миші на кожному проекті у Visual Studio (починаючи, наприклад, з Customers) та вибору опції додавання підтримки для оркестровки контейнерів (Add, Container Orchestrator Support).

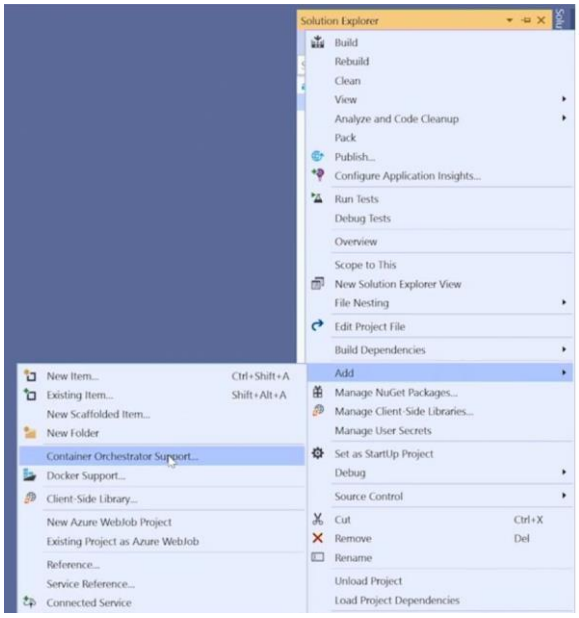

Рис. 31. Додавання підтримки Container Orchestrator

В вікні, що з’явиться, виберіть Docker Compose.

Рис. 32. Варіанти Container Orchestrator

Тепер ми будемо використовувати контейнери Windows у цьому курсі.

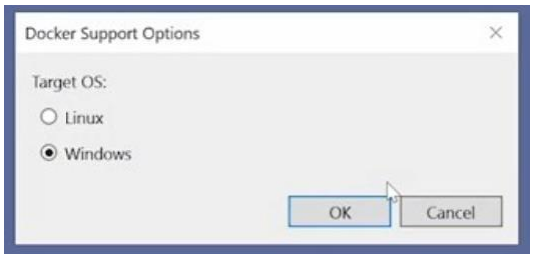

Рис. 33. Вибір контейнерів Windows

Причиною цього вибору є те, що ми будемо використовувати Azure Service Fabric як container Orchestrator Platform, а підтримка цього оркестратора є тільки в Windows. Але багато авторів настійно рекомендують використовувати контейнери Linux замість Windows, оскільки вони більш легкі та ефективні.

Коли ви натиснете OK в діалоговому вікні, буде створений новий файл Docker всередині мікросервісного проекту. Давайте відкриємо проект Customers, і, як бачите, ось файл Dockerfile.

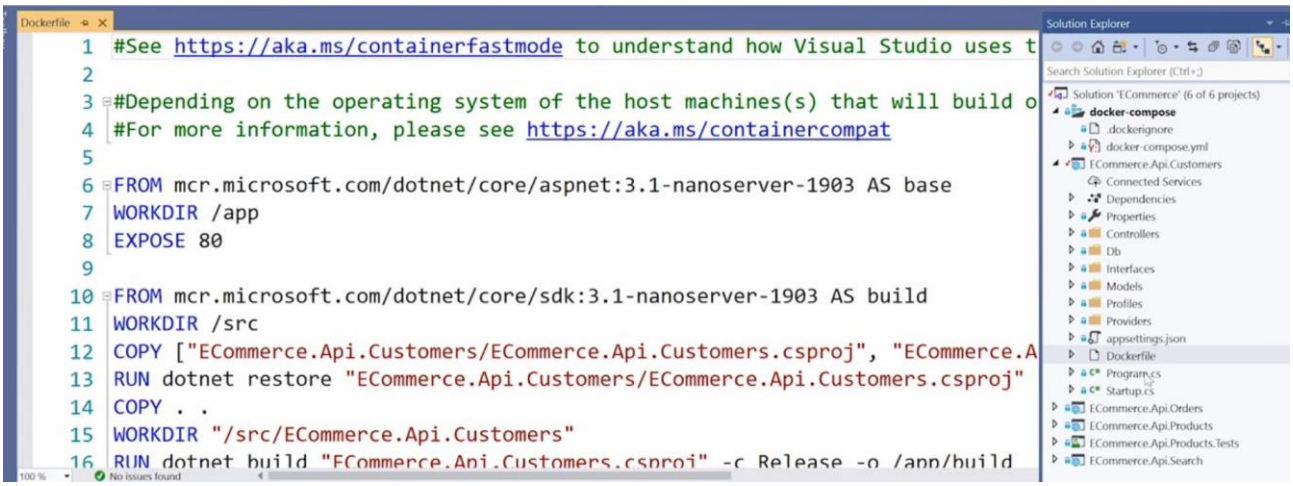

Рис. 34. Файл Dockerfile в проекті Customers

Також в рішенні був створений новий проект Docker Compose. У цьому проекті ви знайдете файли які стосуються Docker Compose.

Цей файл Dockerfile (Рис. 34) - це сценарій, який описує, як створити образ (image) Docker з компільованим сервісом як його вмістом. Враховуйте, що Visual Studio пропонує певну версію зображення на основі вашої поточної конфігурації та фреймворку. У нашому випадку пропонується використовувати зображення Windows Nano Server 1903 як основу (дивись рядки 6 та 10 на Рис. 34). Але давайте змінимо їх на 1803 лише для сумісності з Service Fabric.

Тепер, як це було сказано, повторіть цю ж дію для кожного мікросервісу (залишились Orders, Products і Search).

Якщо ми відкриємо проект Docker Compose і відкриємо файл yml, ми побачимо, що кожен мікросервіс тут зареєстровано як один елемент всередині файлу.

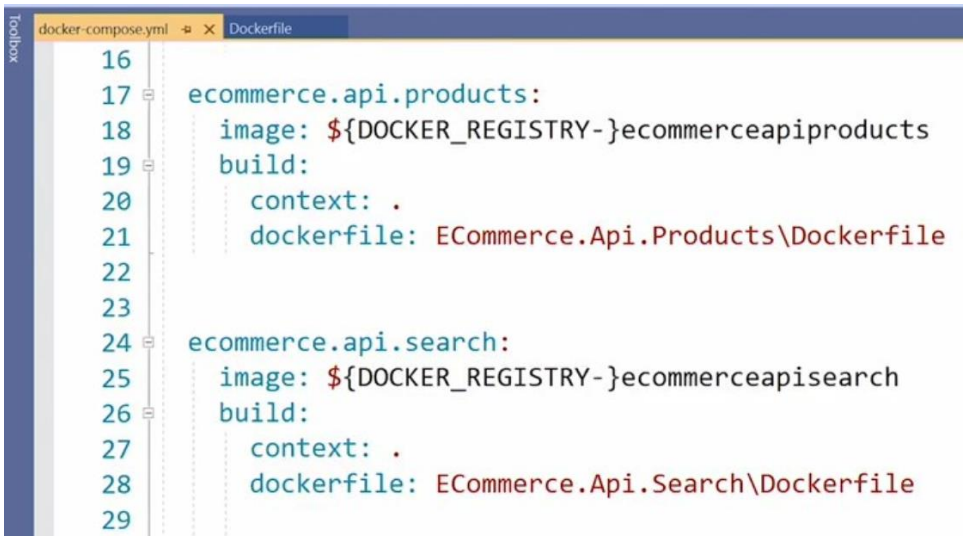

Рис. 35. Файл docker-compose.yml

Тепер ми готові налаштувати та протестувати мікросервіси.

Налагодження та налаштування Docker Compose в рішенні мікросервісів.

Давайте продовжувати та розширювати проект Docker Compose.

Мета файлу docker-compose.yml - включити всі налаштування, які не змінюються між розгортаннями. Цей файл використовує файл docker-compose.override.yml. У нашому випадку ми збираємося змінити цей файл, редагуючи деякі додаткові параметри, такі як порти.

Давайте змінимо налаштування портів, щоб виставити кожен контейнер іншому зовнішньому порту. Почнемо із зміни порту для мікросервісу Products. Ми будемо використовувати порт 6001, а для мікросервіс Customers ми будемо використовувати порт 6002, для Orders – 6003, і нарешті для Search – 6004 (Рис. 36).

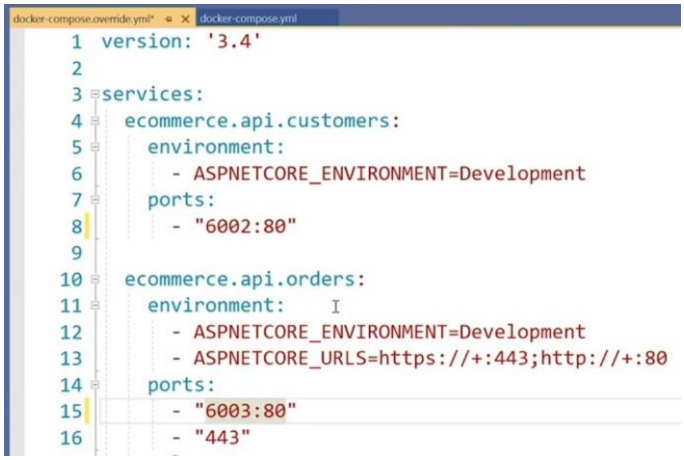

Рис. 36 Зміна портів для мікросервісів у файлі docker-compose.override.yml

Як ви бачите, у нас в мікросервісі Orders є інші елементи конфігурації (Рис. 37).

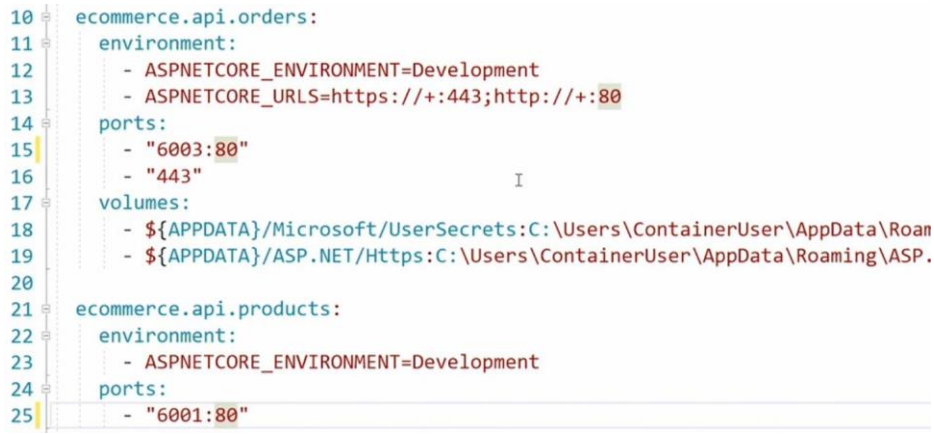

Рис. 37. Файл docker-compose.override.yml

Це тому, що, коли ми створювали проект мікросервісу Orders, ми пропустили змінити конфігурацію на HTTPS, яка зараз увімкнена. Visual Studio виявила це і додала належні розділи до файлу YAML, але ми не будемо використовувати їх у цьому курсі, тому цілком безпечно видалити цей елемент volumes, тож видаліть цей елемент (рядки 17-19), додатковий порт 443 (рядок 16) і додаткову змінну оточення (рядок 13).

Далі поверніться до проекту Orders та відкрийте файл JSON налаштувань запуску launchSettings.json, і видаліть цей пункт SSL-порту (рядок 7, Рис. 38).

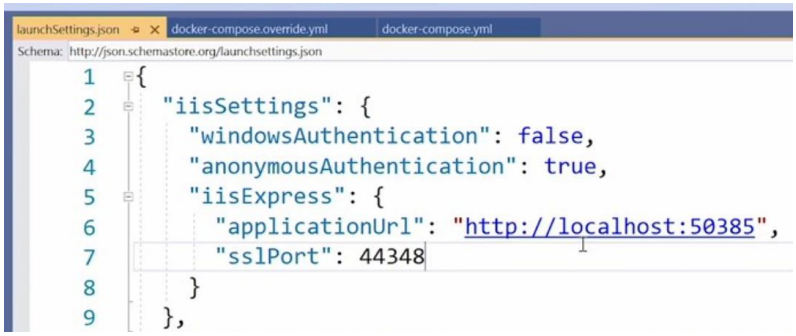

Рис. 38. Файл launchSettings.json

Як ми вже говорили у попередньому розділі, ми будемо використовувати образ Windows 1803, - проблема полягає в тому, що образ не включає останню версію .NET Core 3.1, і якщо ви спробуєте запустити Docker Compose, він надішле помилку з проханням встановити .NET Core.

Мета цього розділу - показати, як легко налагоджувати мікросервіси **на локальному рівні**, тому давайте змінимо в файлі Dockerfile для мікросервіса Customers (і інших мікросервісів) версію образу на 1903, це лише тимчасово, тому що ми будемо використовувати 1803 з Service Fabric.

Гаразд, ми готові протестувати це, натискаємо кнопку Docker Compose і давайте відкриємо Postman.

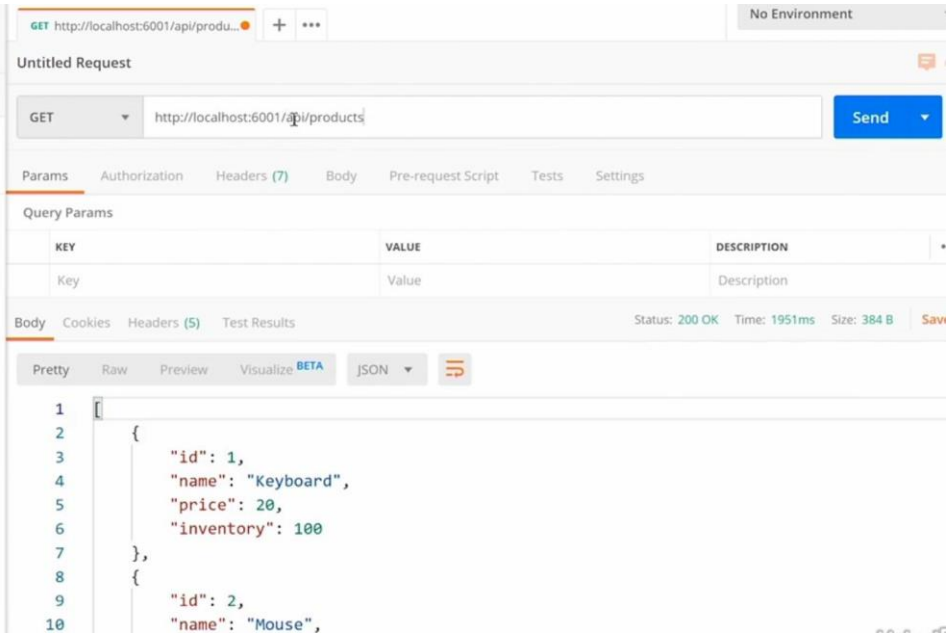

Рис. 39. Тестування мікросервісу Products

Ось ми в додатку Postman, тому давайте виконаємо, наприклад, локальний хост мікросервіса Products, який працює на порту 6001. І, як ви бачите (Рис. 39), він працює.

Мікросервіси Customers (localhost:6002/api/customers) і Orders (localhost:6003/api/orders/1) теж працюють (перевірте).

А що буде з мікросервісом Search? Зробіть запит.

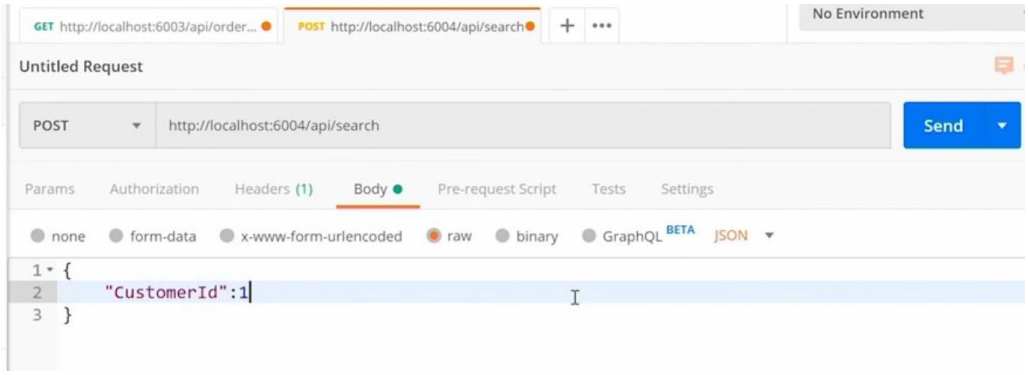

Рис. 40. Запит для мікросервіса Search

І, як ви побачите, він не працює!

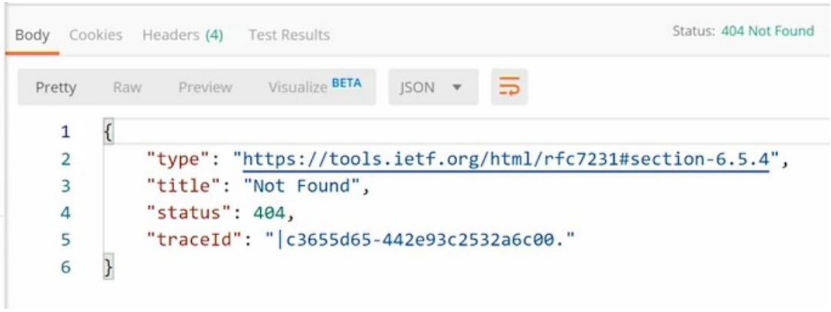

Рис. 41. Помилка в мікросервісі Search

Давайте з’ясуємо, чому мікросервіс Search не працює. Як ви можете згадати, мікросервісу Search потрібно знати, де розташовано інші мікросервіси, ми налаштували це у файлі JSON appsettings.json для мікросервісу Search. Відкрийте цей файл.

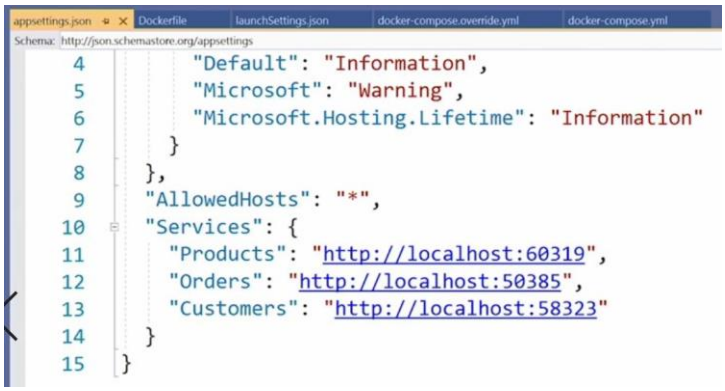

Рис. 42. Старі налаштування в мікросервісі Search

Тому ми замінимо цю конфігурацію такою, яку ми будемо встановлювати, використовуючи змінні середовища. Інфраструктура конфігурації .NET Core підтримує різних провайдерів, один з них - саме змінні середовища, тому для

цього повернемося до файлу docker-compose.override.yml та додамо наступні змінні середовища до мікросервісу Search, першу - для мікросервісу Products (рядок 26), другу - для мікросервісу Customers (рядок 27) і третю – для мікросервісу Orders (рядок 28).

Рис. 43. Зміни в файлі docker-compose.override.yml

Далі знов натисніть кнопку Docker Compose і поверніться до Postman. Повторіть запит до мікросервісу Search, і ви отримаєте результат.

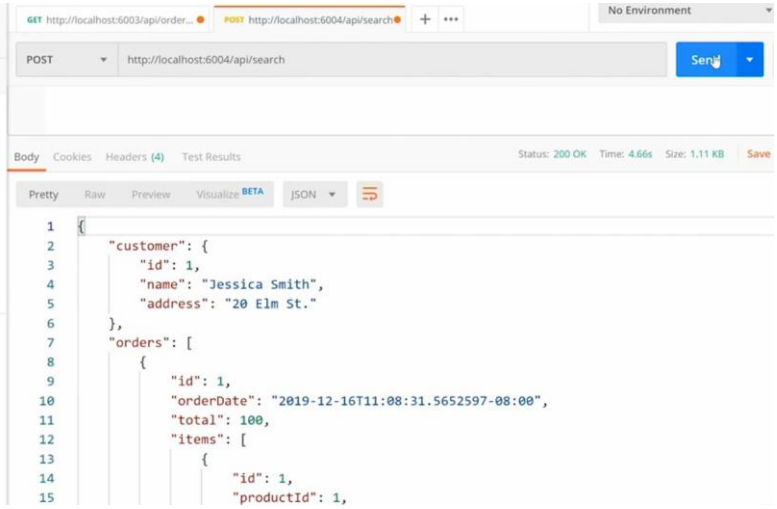

Рис. 44. Коректна робота мікросервісу Search

Тепер, коли ми впевнились, що всі мікросервіси працюють з контейнерами Docker локально, зробимо налаштування для розміщення мікросервісів в хмарі з оркестратором Service Fabric. Зупиніть відладку.

Тож для сумісності з Service Fabric поверніть в файлах Dockerfile всіх мікросервісів номер образу 1803 для .NET Core.

Внесіть ці зміни на ваш репозиторій, тому що ми будемо використовувати їх для створення та перенесення (push) їх docker images до реєстру контейнерів (container registry).

Публікація контейнерних образів для рішення мікросервісів.

Поверніться в наш попередній проект **ecommerce** в Azure DevOps. Оновіть репозиторій Repos новим вмістом, який включає підтримку контейнерів Docker (гілка docker).

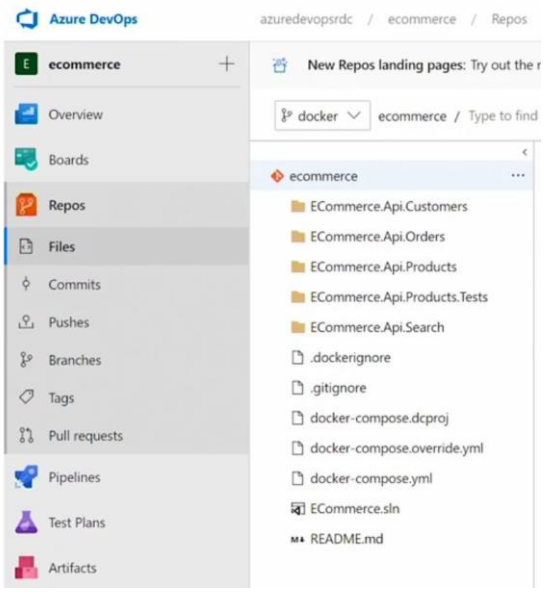

Рис. 45. Оновлений проект в Azure DevOps

Завдяки файлам Docker і Docker Compose, ми готові створити конвеєр випуску (release pipeline), який створює образи Docker. Крім того, конвеєр повинен заштовхувати (push) ці образи до реєстру контейнерів. Є різні реєстри контейнерів, які ми могли б використовувати. Один з них - це **Azure Container Registry**, який є прекрасним варіантом, якщо ви плануєте приватно розміщувати свої мікросервіси в Azure Cloud. Інший варіант - **Docker Hub** - це добре перевірений і дуже відомий реєстр контейнерів, яким керує сам Docker. Docker Hub підтримує як приватні, так і загальнодоступні образи. У цьому розділі ми збираємось переслати образи до Docker Hub, тому що ми хочемо, щоб ви мали доступ до цих образів для довідкових цілей.

Отже, почнемо зі створення нового конвеєра випуску.

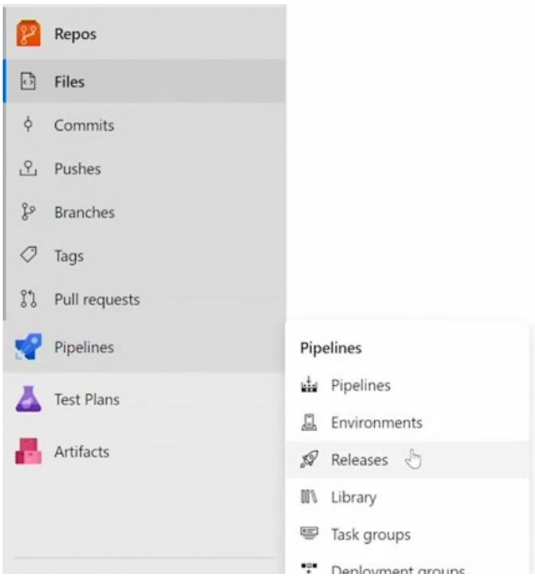

Рис. 46. Створення нового release pipeline

На сторінці, що з’явилась, натисніть кнопку New pipeline. В діалоговому вікні Select a template натисніть Empty job. Далі закрийте діалог.

Натисніть Add an artifact і виберіть Repos.

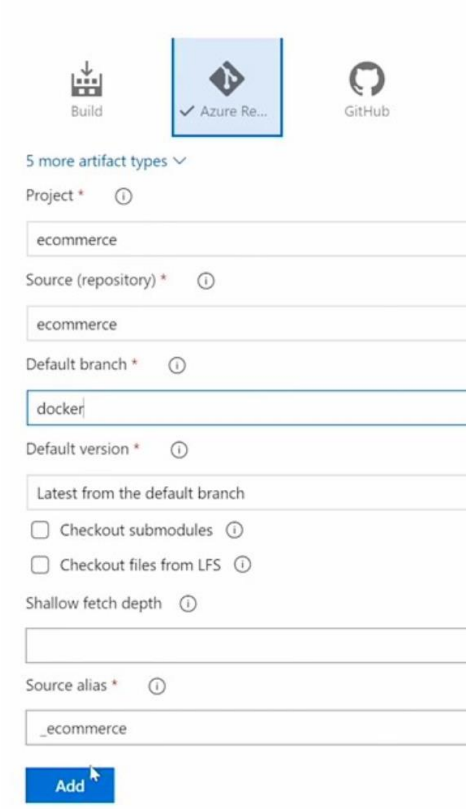

Рис. 47. Вікно додавання артефакту.

Тут обрано проект ecommerce, репозиторій ecommerce, гілку (branch) docker в репозиторії. Натисніть Add.

Далі натисніть Stage 1. З’явиться нова сторінка.

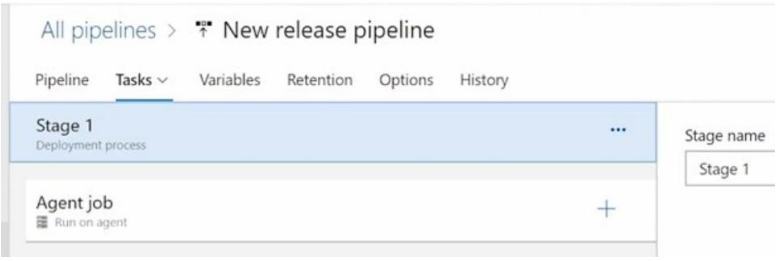

Рис. 48. Сторінка New release pipeline

Перше, що ми будемо налаштовувати - це агента. Пам'ятаймо, що ми будемо використовувати версію Windows 1803, тому ми виберем її.

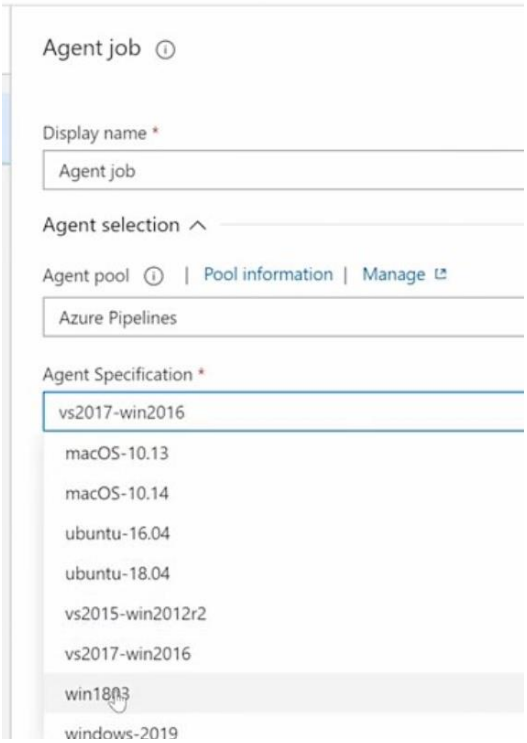

Рис. 49. Вибір win 1803

Далі в рядку Agent job натисніть “+”.

І ми збираємось додати нове завдання для створення цих образів. В вікні Add tasks в рядку пошуку наберіть docker і додайте Docker Compose.

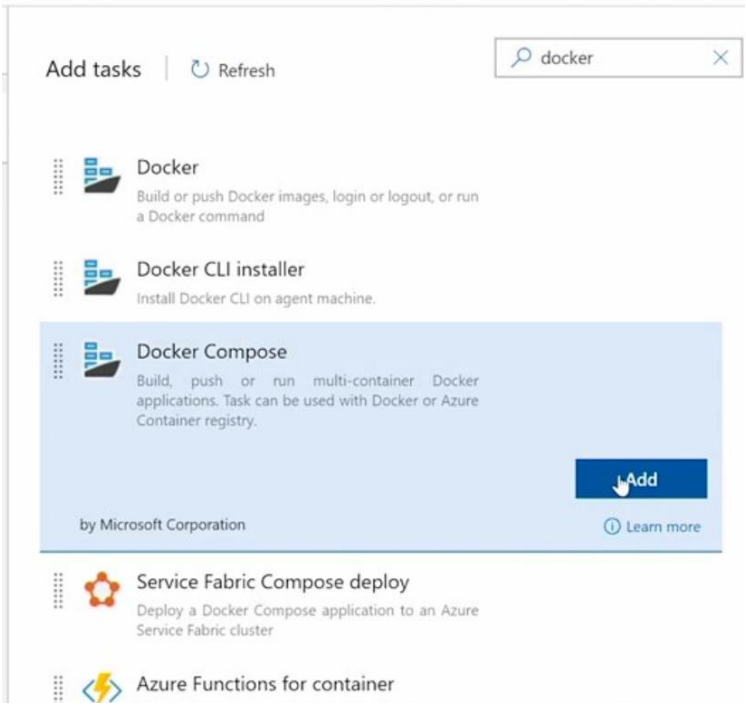

Рис. 50. Додавання завдання Docker Compose

Після натискання Add зліва в списку з’явиться завдання Run a Docker Compose command. Виділіть це завдання.

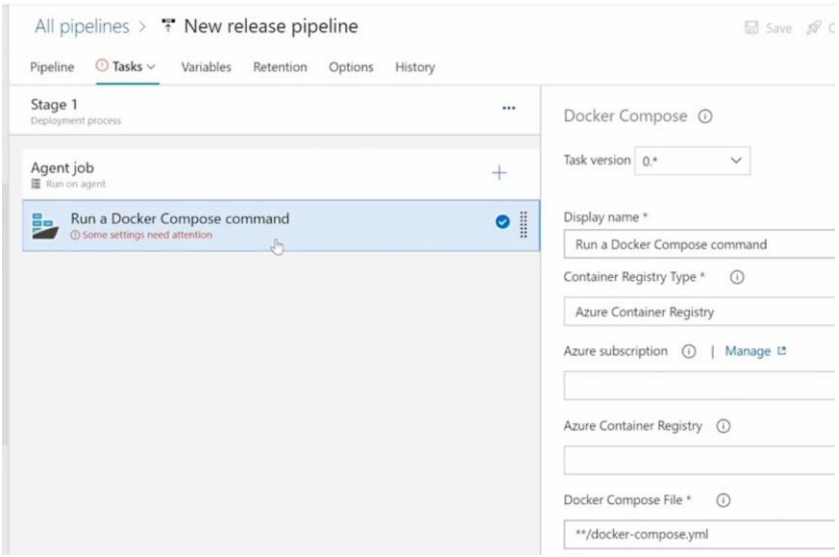

Рис. 51. Завдання Docker Compose

І перше, що ми змінимо в цьому завданні - це тип реєстру контейнерів. Ми хочемо використати Docker Hub. Виберіть в списку Container Registry.

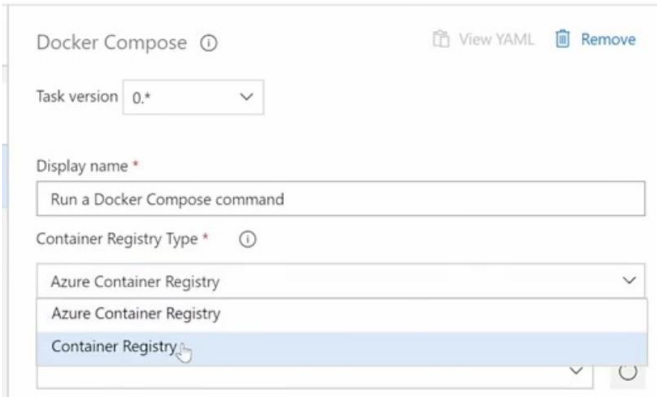

Рис. 52. Вибір Container Registry

Далі бачимо, що при розкритті списку Docker Registry Service Connection з’являється “No result found”.

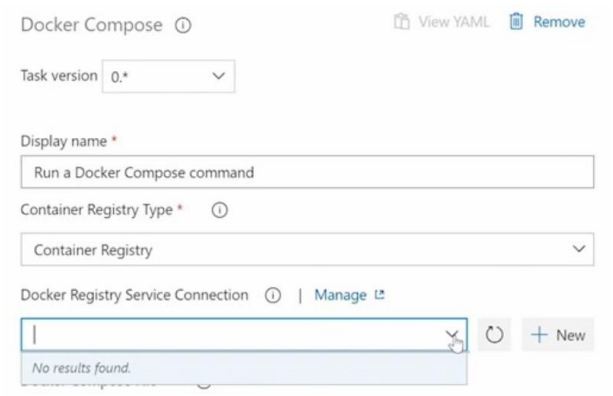

Рис. 53. Поле Docker Registry Service Connection

Тож натисніть “+ New”. З’явиться вікно.

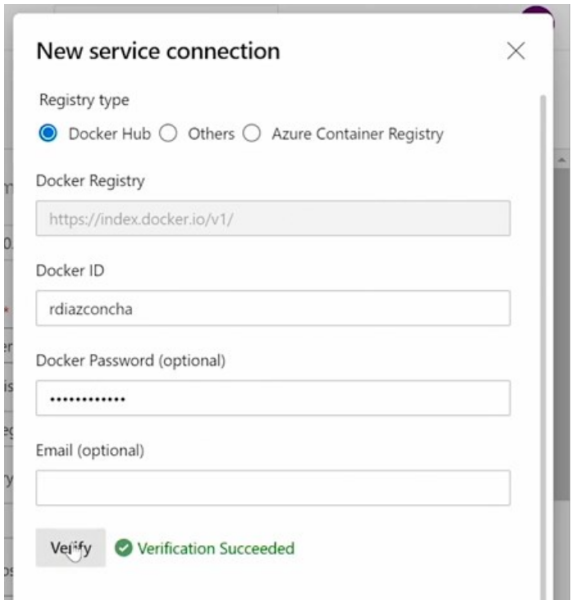

Рис. 54. Додавання нового з’єднання

Виберіть Docker Hub, введіть Docker ID, необов’язковий пароль і натисніть Verify. Далі введіть ім’я нового з’єднання і натисніть кнопку Verify and Save.

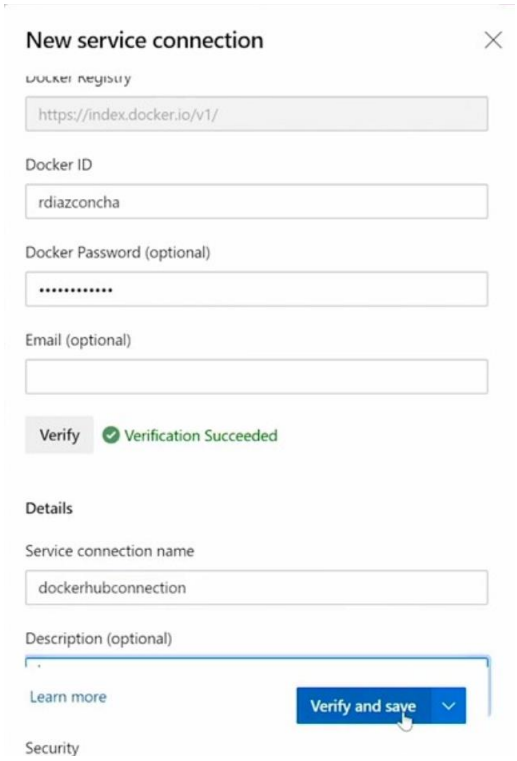

Рис. 55. Заповнене вікно New service connection

Після цього в списку Docker Registry Service Connection з’явиться наше з’єднання.

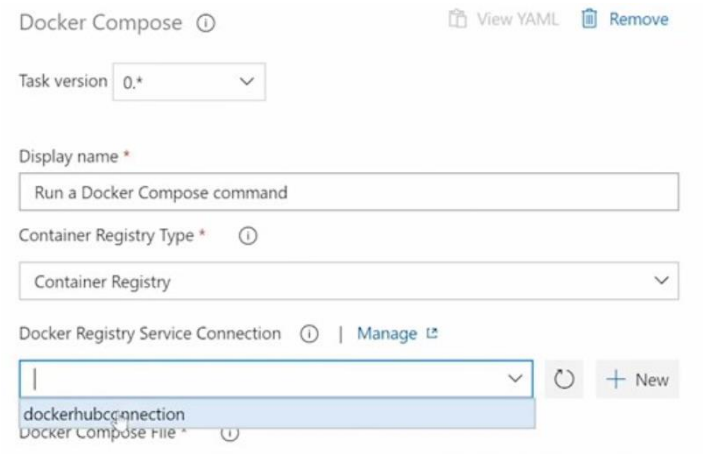

Рис. 56. Вибір з’єднання

Виберіть це з’єднання, прокрутіть список вниз. В пункті Action виберіть Build Service Images.

Рис. 57. Додавання Action

Далі в рядку Agent job натисніть “+” щоб додати ще одне завдання для переміщення цих образів у Docker Hub. Отже, давайте знову шукатимемо Docker. Знайдіть Docker Compose і натисніть Add.

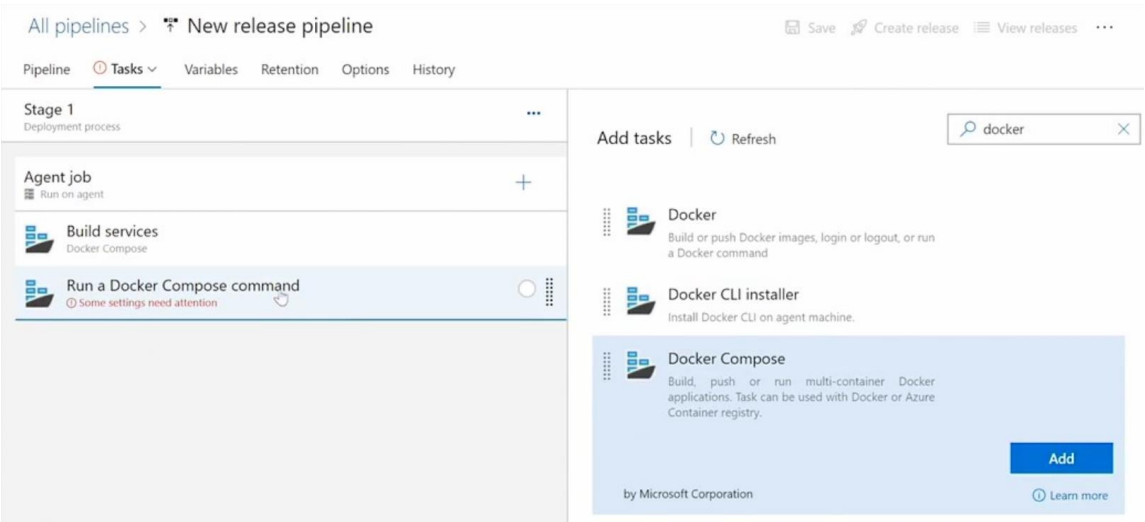

Рис. 58. Додавання завдання Docker Compose

Тепер натисніть Run a Docker Compose command. І зробимо те саме. Змінимо реєстр, який використовує це з'єднання. Задамо те ж саме з'єднання.

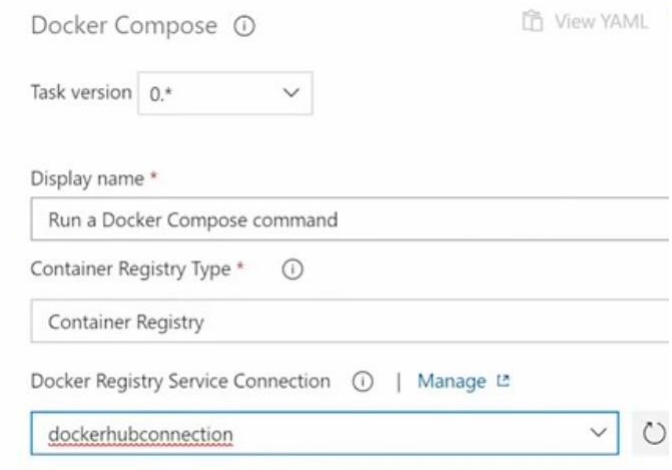

Рис. 59. Зміна Registry Type і зміна Connection

І, на завершення, виберіть Action – Push service images.

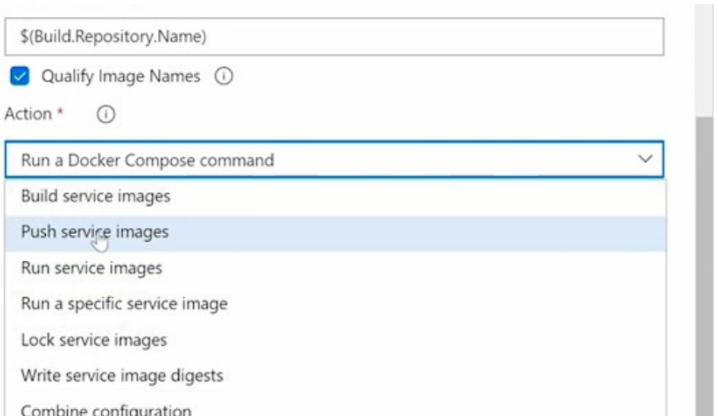

Рис. 60. Вибір Action

Тепер файл Docker Compose вказує змінну середовища для префікса імені зображення. Натисніть Save і ще раз Save в діалоговому вікні.

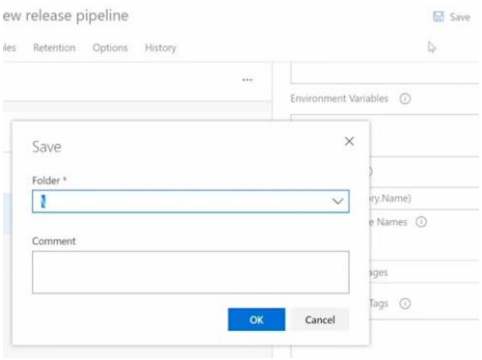

Рис. 61. Збереження pipeline

Тож ми можемо перейти до Repos. І давайте відкриємо файл docker-compose.yml. І як ви бачите, ось змінна середовища. Назва – DOCKER_REGISTRY.

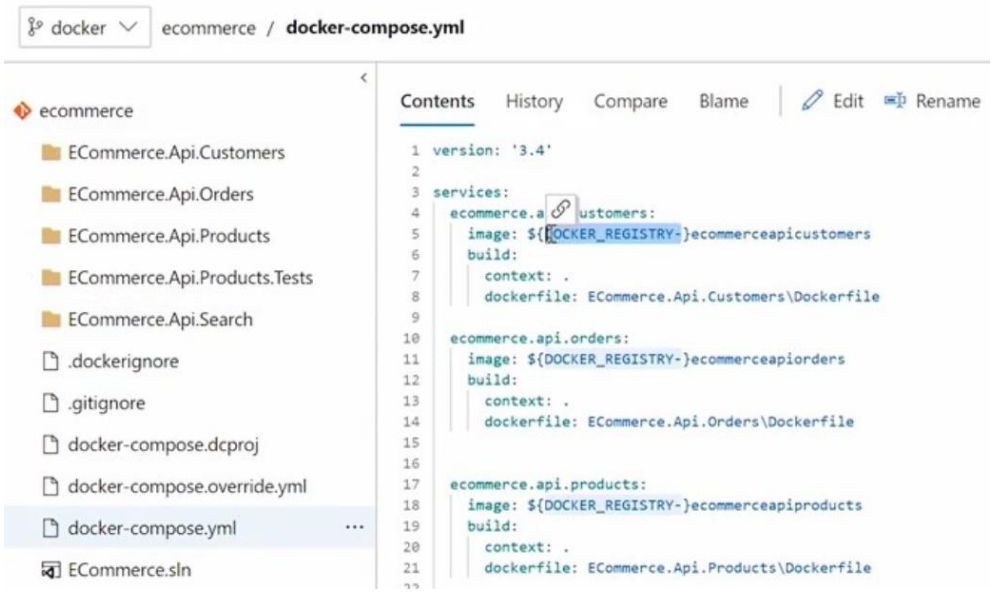

Рис. 62. Перегляд файлу docker-compose.yml

Тож повернемося до конвеєра випуску (release pipeline). Натисніть кнопку Edit. Ми збираємося встановити значення для цієї змінної, тут, у розділі Variables.

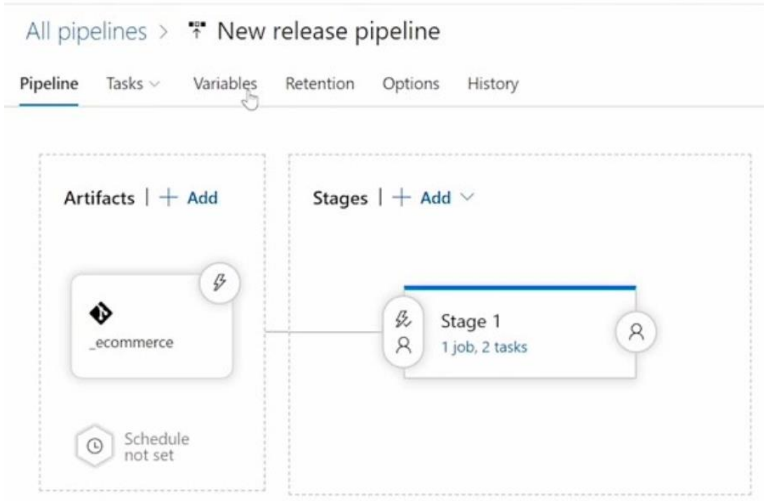

Рис. 63. Вибір пункту Variables

Натисніть Variables, потім Add, введіть Name і Value.

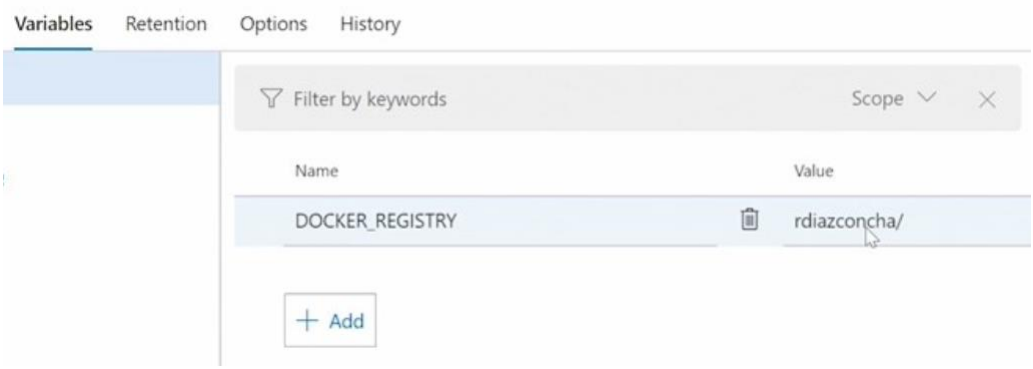

Рис. 64. Встановлення значення змінної DOCKER_REGISTRY

Зауважте, що треба ввести ім’я вашого репозиторію.
Натисніть Save і в діалоговому вікні – OK.
Далі натисніть Tasks, виділіть Build services. І, схоже, зараз все працює.
Натисніть кнопку Create release, а потім внизу кнопку Create.
І ви можете побачити статус цього випуску.

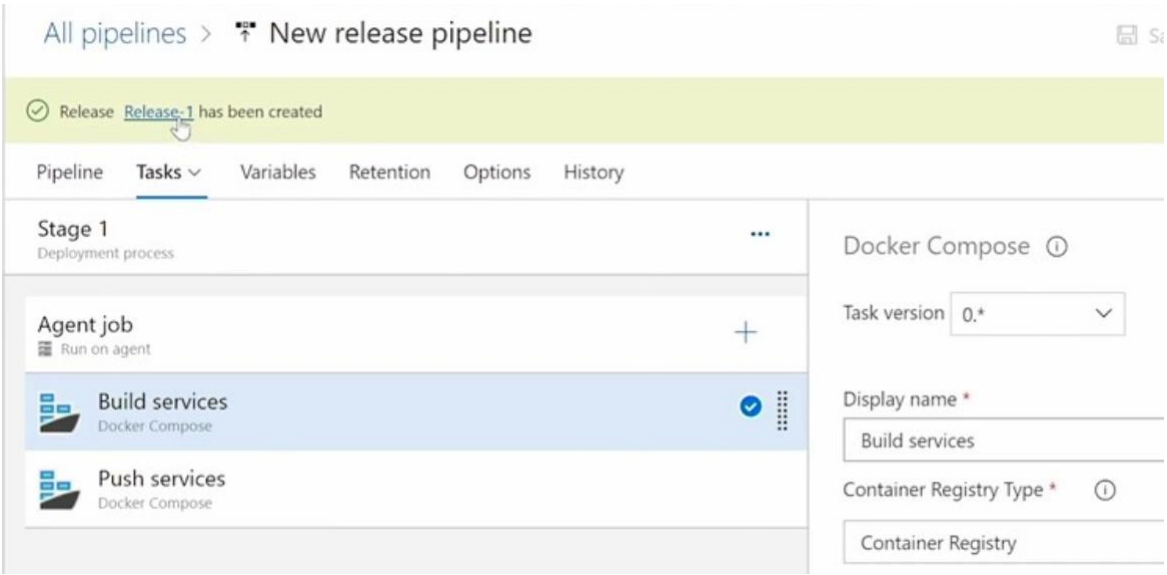

Рис. 65. Зміна статусу

Натисніть посилання Release-1. І як ви бачите, це стартує прямо зараз.

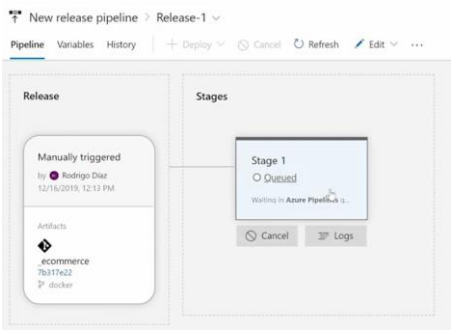

Рис. 66. Старт релізу

Тож давайте натиснемо на Logs, щоб ми могли бачити термінал всередині браузера.

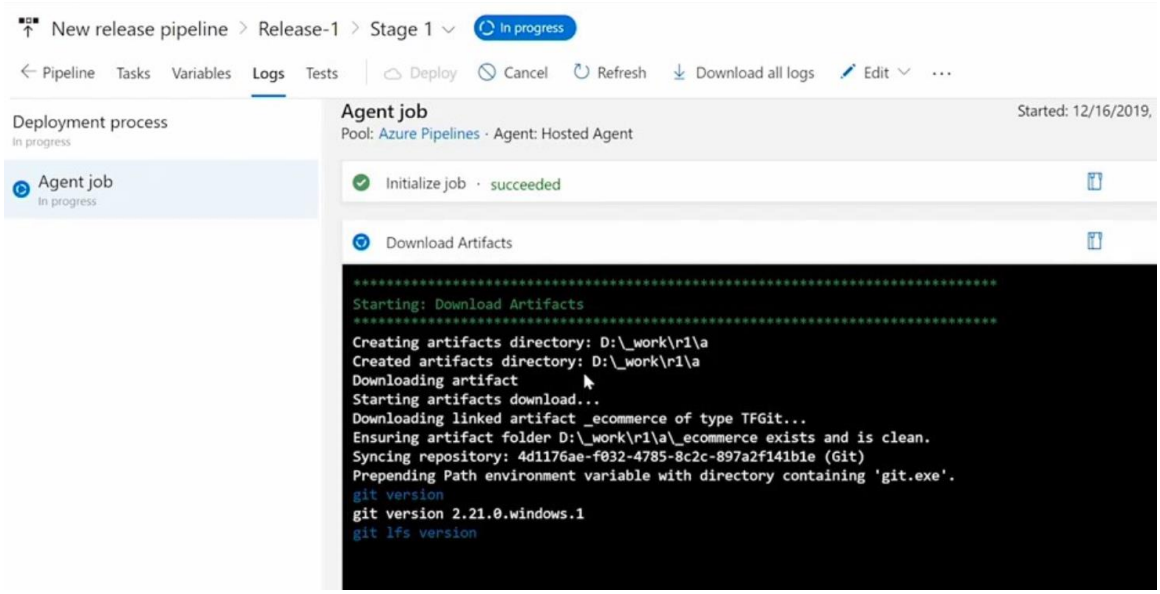

Рис. 67. Відображення журналу

Якщо виконання конвеєра було успішним, ви зможете побачити образи у вашому обліковому записі Docker Hub.

Давайте відкриємо браузер і переходимо до Docker Hub (https://hub.docker.com/u/rdiazconcha). І ось вони.

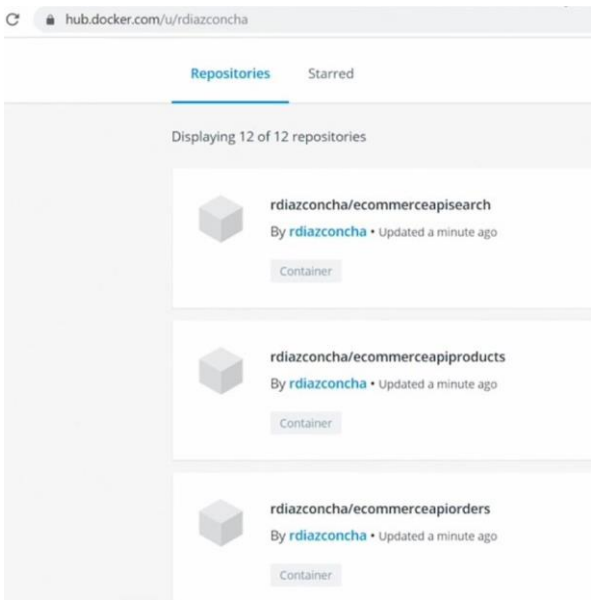

Рис. 68. Образи у Docker Hub

Відмінно. Тепер, коли образи контейнерів опубліковані, ми можемо розгорнути їх на будь-який Docker host або оркестровий контейнер.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Lynda - Azure Microservices with .NET Core for Developers (2020). – URL: https://www.lynda.com/Azure-tutorials/Azure-Microservices-NET-Core-Developers/2825264-2.html