

# Architektury i metodyki wdrożeń systemów Sztucznej Inteligencji

Andrzej Wodecki

2022-06-04



# Contents

<b>1</b>	<b>Wprowadzenie</b>	<b>5</b>
1.1	Motywacja . . . . .	5
1.2	Cele kursu . . . . .	6
1.3	Warsztaty . . . . .	8
<b>2</b>	<b>Architektury IT</b>	<b>11</b>
2.1	Potoki danych . . . . .	11
2.2	Jeziora danych . . . . .	15
2.3	Konsumenci danych . . . . .	16
2.4	Producenci danych . . . . .	16
2.5	Transformacja danych . . . . .	17
2.6	Strategia wdrożenia . . . . .	19
<b>3</b>	<b>Modelowanie architektur</b>	<b>23</b>
3.1	Cykl życia modelu uczenia maszynowego . . . . .	23
3.2	Demo: trening i ewaluacja modelu . . . . .	25
3.3	Ćwiczenie: trening i ewaluacja modelu . . . . .	27
3.4	Modelowanie architektury - wprowadzenie . . . . .	28
3.5	Modelowanie architektury - organizacja . . . . .	28
3.6	Modelowanie architektury - specyfikacja wymagań . . . . .	29
3.7	Modelowanie architektury - modelowanie . . . . .	30
3.8	Demo: modelowanie . . . . .	31
3.9	Podsumowanie: modelowanie architektur . . . . .	33
<b>4</b>	<b>Monitoring modeli</b>	<b>35</b>
4.1	Monitoring modeli - wprowadzenie . . . . .	36
4.2	Demo: monitoring modelu . . . . .	37
4.3	Ćwiczenie: monitoring modelu . . . . .	39
4.4	Detekcja dryfu modelu . . . . .	39
4.5	Demo: detekcja dryfu . . . . .	42
4.6	Ćwiczenia: detekcja dryfu i dotrenowanie . . . . .	43
4.7	Monitoring modeli - podsumowanie . . . . .	44
<b>5</b>	<b>Konteneryzacja</b>	<b>45</b>

5.1	Wprowadzenie . . . . .	45
5.2	Organizacja, pierwszy obraz i prosta aplikacja . . . . .	50
5.3	Wykorzystanie gotowych kontenerów . . . . .	55
5.4	Wymiana plików kontener<>host . . . . .	58
5.5	Aplikacje wielokontenerowe . . . . .	70
5.6	Podsumowanie . . . . .	72

# Chapter 1

## Wprowadzenie

### 1.1 Motywacja

Trening modelu uczenia maszynowego to dopiero początek: wcześniej trzeba pozyskać i przygotować dane, następnie go udostępnić, a po udostępnieniu monitorować jego jakość.

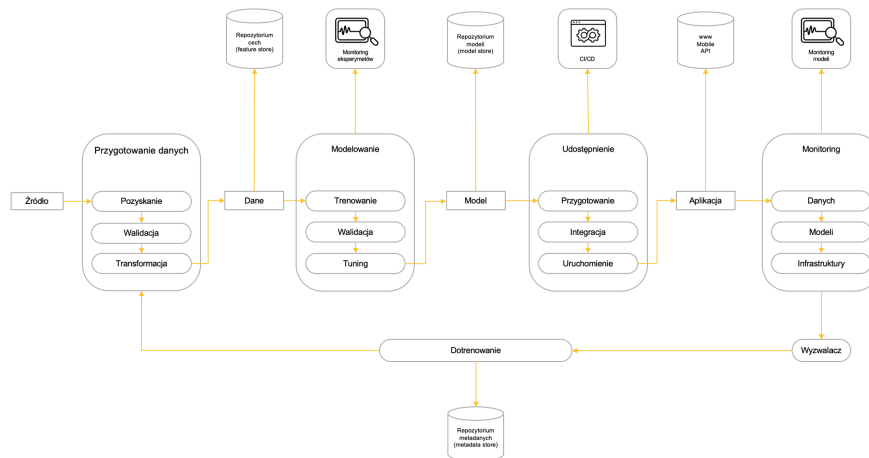


Figure 1.1: image-20220329115350377

W tym „cyklu życia modelu” pracujemy z wieloma komponentami i artefaktami:

1. Komponenty „przetwarzają” dane, są najczęściej skryptami komputerowymi (lub modułami dostępnymi w formie kontenerów)
2. Artefakty są produktami i/lub materiałem wejściowym dla komponentów

Przykładowe komponenty to: przygotowanie danych, modelowanie, udostępnianie, monitoring, dotrenowanie... Przykładowe artefakty zaś to: dane, modele, aplikacje

Komponenty też nie są proste... Składa się na nie najczęściej wiele różnych, złożonych działań.

Artefakty zaś wymagają składowania w dedykowanych strukturach, takich jak:

1. Repozytoria cech (features stores)
2. Repozytoria modeli (model stores)
3. Repozytoria metadanych (metadata stores)

Całość trzeba przy tym:

1. monitorować:
  1. Jakość i charakterystyki danych (np. dryf danych)
  2. Wyniki eksperymentów
  3. Jakość modeli (np. dryf modeli)
2. ... kontrolować
  1. Kontrola przebiegu procesu
  2. Zarządzanie parametrami przebiegów...
3. i wersjonować
  1. Dane
  2. Modele.

Dane zadanie/proces ML może być realizowane:

1. Na różne sposoby
2. Z wykorzystaniem różnych bibliotek i różnych narzędzi
3. W różnych środowiskach (lokalnie, serwery w chmurze, chmura bezserwerowa, urządzenia mobilne.

Źródło: <https://ai-infrastructure.org/maximizing-ml-infrastructure-tools-for-production-workloads-arize-ai/>

Na szczęście, pomaga nam w tym wiele różnych bibliotek i narzędzi. Krajobraz ten szybko się zmienia.

## 1.2 Cele kursu

Co zatem warto wiedzieć? Potrafić?

O wiele ważniejsza od znajomości konkretnych narzędzi i technik jest wiedza o tym, co robić (w danej sytuacji), a nie jak to robić. Świadomość na-

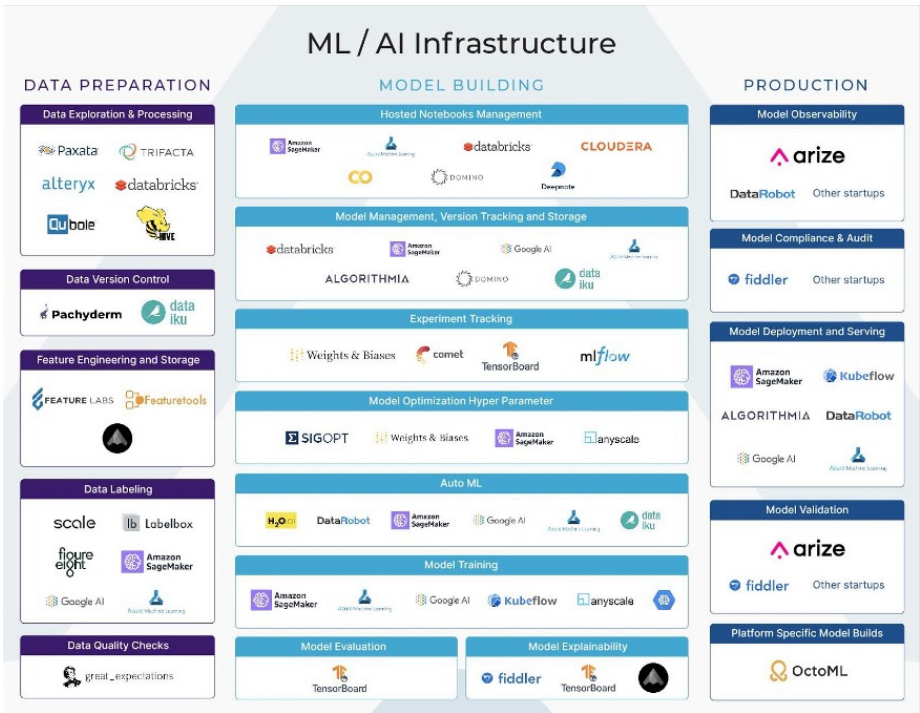


Figure 1.2: image-20220329115417586

jważniejszych, w miarę uniwersalnych etapów, czynności, metod i dobrych praktyk (np. MLOps). I zdolność ich zastosowania w praktyce.

Podczas tego kursu:

1. Stworzysz projekt prostej architektury rozwiązania uczenia maszynowego uwzględniającego najważniejsze etapy cyklu życia modelu
2. Zaimplementujesz tę architekturę w wybranym środowisku wykorzystując różne, dedykowane biblioteki Python
3. Stworzysz dokumentację i plan uruchomienia takiego projektu z wykorzystaniem dobrych praktyk metodyki MLOps.

## 1.3 Warsztaty

Pracę na ćwiczeniach podzielimy na 5 etapów:

1. Stworzenie komponentów umożliwiających
  1. trening
  2. uruchomienie
  3. i monitoring modeli
2. Implementację tych komponentów w postaci kontenerów Docker (opcjonalnie: Kubernetes lub Dataiku)
3. Opracowanie dokumentacji całej architektury.



Trening



Uruchomienie



Monitoring



Konteneryzacja



Dokumentacja

W pracach wykorzystamy następujące narzędzia:

1. Pakiet MLFlow
2. Środowisko Docker/Kubeflow
3. Platformę Dataiku.

Na końcową dokumentację złożą się:

1. Opis problemu: Co i dlaczego modelujemy?
2. Analiza potrzeb.
  1. Krótka charakterystyka organizacji, dla której trenujemy model (1 akapit)
  2. Prognozowana częstość dotrenowywania
  3. Złożoność obliczeniowa algorytmu
  4. Ilość i złożoność danych.



3. Rekomendowana architektura (implementacja w diagrams.net z łączyami do GitHub (artefakty i komponenty))
4. Dokumentacja implementacji:
  1. Komponenty
  2. Artefakty
  3. Środowisko uruchomieniowe
5. Dyskusja wyników (ekrany z poszczególnych faz):
  1. Dane
  2. Modelowanie
  3. Monitoring.



## Chapter 2

# Architektury IT

### 2.1 Potoki danych

Na typową architekturę IT składają się producenci danych, ich konsumenci i system je przetwarzające.

Kluczowe pytania, na które warto odpowiedzieć projektując architekturę IT, to:

1. Skąd pozyskamy dane?
2. W jaki sposób je pozyskamy? W jaki sposób będziemy je przetwarzać?  
Jak je będziemy gromadzić?
3. Dokąd te dane później trafią?

Najważniejsze powody, dla których warto projektować architektury IT, to:

1. Rozwiązanie klasycznych problemów z danymi, takich jak:
  1. zmieniające się schematy baz i scenariusze użycia
  2. rosnąca ilość danych
  3. błędy w danych
  4. duplikacja danych
  5. wycieki danych
  6. opóźnienia (latencja)
  7. awarie w procesach
  8. konieczność manualnego zarządzania procesami IT.
2. Integracja silosów informacyjnych często obecnych w firmach (osobne systemy wspomagające komunikację, zarządzanie różnymi obszarami działania, etc.)

#### 2.1.1 Typy danych

Najważniejsze typy danych, które napotkamy w projektach uczenia maszynowego, to:

1. Dane o zdarzeniach, obiektach i ich agregaty
2. Dane ustrukturyzowane, nieustrukturyzowane i częściowo-ustrukturyzowane.

**Dane o obiektach** (ang. entity data) przedstawiają najczęściej stan obiektu, np. użytkownika, produktu, zamówienia.

**Dane o zdarzeniach** (ang. event data) opisują działania wykonywane przez (lub na) obiektach.

Najbardziej typowe cechy takich zdarzeń to:

- identyfikator
- typ zdarzenia
- znacznik czasu
- informacje uzupełniające.

Warto podkreślić, że współczesne systemy generują zdecydowanie więcej danych o zdarzeniach niż danych o podmiotach (na każdego użytkownika korzystającego z aplikacji mogą przypadać tysiące zdarzeń).

Dane o zdarzeniach mogą być **agregowane** np. w celu analiz biznesowych (np. KPIs). Przykładowo, często wykorzystywane w biznesie miary będące wynikiem agregacji danych o obiektach i wydarzeniach to:

1. Liczba aktywnych użytkowników Dzienni aktywni użytkownicy (DAU) Tygodniowi aktywni użytkownicy (WAU) Miesięczna liczba aktywnych użytkowników (MAU)
2. Długość sesji Czas spędzony przez użytkownika korzystającego z Twojej aplikacji podczas jednej sesji.
3. Współczynnik kliknięć (CTR) Stosunek liczby użytkowników, którzy kliknęli na reklamę lub banner do liczby użytkowników, którzy obejrzeli stronę z tą reklamą lub bannerem.
4. Współczynnik odrzuceń (BR) Procent użytkowników opuszczających witrynę po obejrzeniu tylko jednej strony.
5. Współczynnik konwersji (CR) Procent użytkowników, którzy wykonują pożądaną akcję.

Relację pomiędzy danymi o zdarzeniach i obiektach można podsumować następująco:

Kolejną istotną w uczeniu maszynowym charakterystyką danych jest ich podział na dane ustrukturyzowane, częściowo-ustrukturyzowane i nieustrukturyzowane.

**Dane ustrukturyzowane:**

1. Są uporządkowane w tabelach
2. Można określić między nimi relacje
3. Można odpytywać korzystając z języka SQL (Structured Query Language)
4. Wymagają wskazania schematu (Schema): sposobu organizacji danych.

**Dane częściowo-ustrukturyzowane:**

1. Nie są zgodne z relacyjnymi bazami danych, takimi jak Excel czy SQL, ale mają pewien poziom organizacji, np. znaczniki.
2. Nie są ściśle relacyjne
3. Po przetworzeniu mogą być przechowywane w:
  1. relacyjnych bazach danych
  2. bazach NoSQL
  3. plikach CSV, XML i JSON.

**Dane nieustrukturyzowane:**

1. Najczęściej dane jakościowe
2. Nie posiadają schematu/modelu, czy też relacji
3. Można je składować w bazach NoSQL i jeziorach danych
4. Przykłady: pliki audio, video, dokumenty tekstowe, wpisy na forach dyskusyjnych, etc.

**2.1.2 Bazy danych**

Technologie gromadzenia danych powinny być dostosowane do ich typu.

**2.1.2.1 Bazy SQL****Dane ustrukturyzowane**

1. Uporządkowane w tabelach
2. Można określić między nimi relacje
3. Można odpytywać językiem SQL (Structured Query Language)
4. Wymagają wskazania schematu (Schema): sposobu organizacji danych
5. W efekcie, gromadzimy najczęściej w bazach SQL.

Kluczową technologią w tego typu systemach jest OLTP (OnLine Transaction Processing).

Przykładowe bazy SQL: Oracle, MS SQL Server, MySQL, PostgreSQL

Przykładowe zastosowania: systemy finansowe, transakcyjne, ERP, etc.

**2.1.2.2 Bazy NoSQL**

Bazy NoSQL służą do przechowywania **danych nieustrukturyzowanych**.

Wyróżniamy 4 podstawowe typy baz NoSQL.

1. Bazy zorientowane na dokumenty
2. Bazy kolumnowe
3. Bazy oparte o wartości kluczy (key-value)
4. Bazy grafowe.

---

**Bazy zorientowane na dokumenty:**

1. Nie posiadają ustalonego schematu
  2. Dane składowane w dokumentach JSON (JavaScript Object Notation)
  3. Każdy dokument może mieć inny zestaw pól
  4. Przykładowe bazy: MongoDB, CouchDB, DocumentDB
  5. Przykładowe zastosowania: systemy zarządzania dokumentami.
- 

**Bazy kolumnowe**

1. Dane składowane są w nich w kolumnach (nie w wierszach)
  2. W efekcie, operacje (zapytania, dodawanie, kasowanie, etc.) oparte na kolumnach działają w nich bardzo szybko
  3. Przykładowe bazy: Cassandra
  4. Przykładowe zastosowania: zaawansowane analizy danych.
- 

**Bazy oparte o wartości kluczy (key-value)**

1. Każdy wpis ma w nich unikatowy klucz
  2. Efekt: umożliwiają szybszy zapis i odczyt danych
  3. Przykładowe bazy: Redis, Amazon Dynamo DB
  4. Przykładowe zastosowania: opinie klientów.
- 

**Bazy grafowe**

1. Dane składowane w formie sieci
2. Koncentracja na połączeniach (relacjach) pomiędzy punktami (obiektami)
3. Wykorzystywane w analizach relacji
4. Przykładowe bazy: Neo4j, Inifinite Graph
5. Przykładowe zastosowania: analiza sieci społecznych.

**2.1.3 Hurtownie danych**

Bazy danych, oparte na technologii OLTP (OnLine Transaction Processing) są zaprojektowane w celu zapewnienia efektywnego działania systemów transakcyjnych. Ich celem nie jest optymalizacja analityki

Hurtownie danych:

1. oparte są o technologie OLAP (Online Analytical Processing), wspomagające użytkowników w interaktywnej analizie wielowymiarowych danych, w szczególności:

1. Konsolidacji (grupowania)
2. Drążenia (drill-down)
3. Przekrojów danych
2. Hurtownie danych wykorzystują dane zgromadzone w bazach danych (OLTP), tworząc warstwę zoptymalizowaną pod kątem zastosowań analitycznych.

W efekcie, hurtownie danych integrują dane z różnych źródeł, będąc często centralnym repozytorium informacji zoptymalizowane pod kątem analityki biznesowej.

Źródłami danych dla hurtowni danych są różne systemy transakcyjne i inne bazy danych.

Główne **zalety i korzyści** ze stosowania hurtowni danych to:

1. Konsolidacja danych w jednym miejscu
2. Szybsze analizy biznesowe
3. Ułatwione procesy transformacji i wzbogacania danych oraz inżynierii cech
4. Poprawa jakości danych.

**Wady i ograniczenia** hurtowni danych:

1. Mogą być kosztowne
2. Wymagają ciągłej opieki (czyszczenie, transformacja, integracja danych, ...)
3. Bywają zbyt złożone w przypadku doraźnych potrzeb analitycznych.

Hurtownie danych **warto stosować** w danej organizacji:

1. Jest wiele rozproszonych baz danych/systemów dziedzinowych
2. Jest wiele różnych baz danych
3. Jest gromadzona duża ilość danych historycznych.

## 2.2 Jeziora danych

Jeziora danych to repozytoria, które przechowują dane w ich naturalnej postaci. Stanowią zazwyczaj pojedynczy zbiór wszystkich danych przedsiębiorstwa. Są źródłem danych dla systemów umożliwiające raportowanie, wizualizację, zaawansowaną analitykę i uczenie maszynowe.

Główne **zalety** jezior danych:

1. Można w nich przechowywać duże ilości danych...
2. ...które mogą mieć różne formy:
  1. Ustrukturyzowane
  2. Częściowo ustrukturyzowane
  3. Nieustrukturyzowane
3. Przetwarzanie przed załadowaniem nie jest wymagane.

Główne **wady i ograniczenia** jezior danych to:

1. To technologia, która wciąż się rozwija
2. Problemy ze specjalistami
3. Zarządzanie danymi może być uciążliwe
4. Niskie koszty mogą stymulować gromadzenie danych niepotrzebnych
5. Prywatność danych: dane całej organizacji w jednym repozytorium.

Jeziora danych **warto stosować** w następujących sytuacjach:

1. Eksperymenty Data Science: chcemy sprawdzić proof-of-concept architektury przed zainwestowaniem w profesjonalny potok danych
2. Do analizy danych w obszarze cyberbezpieczeństwo: gromadzenie logów z wielu urządzeń w celu analizy pod kątem bezpieczeństwa
3. Do analizy danych o klientach: gromadzenie i analiza danych o zachowaniach klientów z wielu źródeł i kanałów (www, mobile, sklepy tradycyjne, e-commerce, systemy lojalnościowe, CRM, etc.).

## 2.3 Konsumenci danych

Analizę potrzeb, która będzie podstawą dla projektu architektury systemu IT/uczenia maszynowego, warto rozpocząć od zbadania potrzeb użytkowników końcowych (konsumentów danych), podstawowych celów biznesowych i typowych scenariuszy użycia.

Kluczowe **pytania**, które warto zadać na tym etapie, to:

1. W jaki sposób konsument danych chce z nich korzystać? Do raportowania, tworzenia wizualizacji, podejmowania decyzji, a może do budowania modeli Machine Learning (ML)?
2. Jakie narzędzia są aktualnie używane przez użytkowników? Microsoft Excel, Tableau, Microsoft Power BI lub Google Data Studio?
3. Czy istnieją jakieś standardy w ramach danej grupy użytkowników? Dział prawny może potrzebować danych w innej postaci niż księgowość czy finanse.

Możliwe **cele biznesowe** to:

Najbardziej typowe **scenariusze użycia**:

## 2.4 Producenci danych

Typowe źródła **danych o zdarzeniach** to:



1. Strony www generujące dane o zachowaniach użytkownika:
  1. Pobrania
  2. Kliknięcia
  3. Wypełnienie formularza
  4. Komentarze
2. Media społecznościowe:
  1. Publikacja wpisu
  2. Udostępnienie obiektu (wpis, zdjęcie, film, ...)
  3. Polubienie obiektu
  4. Hashtag
  5. Wystawienie opinii
3. Systemy IT, generujące sygnały takie jak:
  1. Replikacja danych
  2. Synchronizacja danych
  3. Uruchomienie zadania
  4. Wykasowanie zadania, etc.
4. Sensory, np.
  1. Detektory ruchu
  2. Detektory głosu
  3. Detektory temperatury
  4. Detektory dymu, etc.

---

Typowe źródła **danych o obiektach** to:

1. Systemy transakcyjne i dziedzinowe (ERP, CRM, etc.)
2. Bazy danych
3. Hurtownie danych
4. Pliki i (rozproszone) systemy plików
5. Źródła zewnętrzne, API (Application Programming Interface).

## 2.5 Transformacja danych

Dane pozyskane ze źródeł są najczęściej przetwarzane na dwa różne sposoby:

1. ETL: Extract > Transform > Load
2. ELT: Extract > Load > Transform.

### 2.5.1 Przetwarzanie ETL (Extract, Transform, Load)

**Ekstrakcja danych** (ang. extract) to czynność lub proces pobierania danych ze źródeł danych w celu ich dalszego przetwarzania lub przechowywania.

**Transformacja danych** (ang. Transform) to zbiór reguł lub funkcji stosowanych do pozyskanych danych w celu przygotowania ich do załadowania

do docelowego systemu.

**Ładowanie danych** (ang. load) polega na przekazaniu danych do docelowego magazynu: płaskiego pliku, bazy czy hurtowni.

Dane mogą być przetwarzane **wsadowo** lub **w sposób ciągły (strumieniowe)**.

---

**Przetwarzanie wsadowe** (ang. batch processing) polega na jednoczesnym przetwarzaniu dużej ilości danych.

---

**Przetwarzanie strumieniowe** (stream processing) odbywa się w czasie zbliżonym do rzeczywistego - dane są przetwarzane w miarę ich napływu. Przykłady: przetwarzanie płatności i wykrywanie oszustw.

Wyzwania związane ze stosowaniem przetwarzania ETL:

Zbyt dużo danych.

1. Ilość danych generowanych rośnie
2. Programy służące do transformacji mogą liczyć miliony linii. Może to bardzo utrudnić skalowanie potoku ETL.
3. Przekształcanie wszystkich danych przed ich załadowaniem może być zbyt kosztowne. Przykładowo może się okazać, że nie ma potrzeby przetwarzania wszystkich danych o zdarzeniach generowanych na stronie internetowej jednocześnie.

Różne typy danych

1. Różne typy danych (ustrukturyzowane, nie ustrukturyzowane, o obiektach czy zdarzeniach) wymagają różnych metod transformacji.
2. ETL najlepiej sprawdza się w przypadku ustrukturyzowanych danych o obiektach (structured, entity).

---

W efekcie, ETL warto stosować, w sytuacji, gdy dysponujemy dużą ilością ustrukturyzowanych danych transakcyjnych.

### 2.5.2 Przetwarzanie ELT (Extract, Load, Transform)

W przetwarzaniu ELT:

1. dane **różnego typu** (ustrukturyzowane, nieustrukturyzowane lub częściowo ustrukturyzowane)
2. pobierane są z różnych źródeł danych
3. a następnie ładowane do magazynu danych, np. jeziora danych.

Transformacja następuje po załadowaniu do jeziora, po czym przetransformowane dane przekazywane są do dalszego wykorzystania przez ich konsumentów.

**Wyzwania przetwarzania ELT:**

1. Są drogie:
  1. Dużo danych różnego typu
  2. Wymagają skalowalności
  3. Wymagają dużych zasobów (składowanie, przetwarzanie)
2. Odpowiednie technologie są stosunkowo nowe:
  1. i w efekcie mogą być mniej niezawodne niż ETL
  2. trudno w związku z tym o specjalistów
  3. i trudniej zapewnić bezpieczeństwo.

**ELT warto stosować:**

1. Gdy gromadzimy duże ilości danych
2. Nie ma możliwości przetwarzać ich przed załadowaniem
3. Dane są nieustrukturyzowane lub mieszane.

Przykładowe zastosowania: dane do analizy sentymentu (opinie, e-mail'e, gwiazdki), dane z logów systemowych, etc.

## 2.6 Strategia wdrożenia

### 2.6.1 Własne czy gotowe?

Jedną z pierwszych decyzji, którą należy podjąć już podczas projektowania architektury IT, jest to, czy wykorzystamy rozwiązanie gotowe, czy też stworzymy własne?

Generalnie, rekomenduję kierować się następującą zasadą:

1. w pierwszej kolejności sprawdź, czy dane rozwiązanie jest dostępne na rynku? Jeśli tak: wykorzystaj je.
2. dalej, sprawdź, czy można dostosować jakieś rozwiązanie do Twoich potrzeb
3. ... a dopiero jeśli nie jest możliwe skorzystanie z rozwiązania gotowego, nie ma też niczego, co można by dostosować: stwórz własne rozwiązanie.

**Kup gotowe, gdy...:**

1. całkowity koszty zakupu jest dużo niższy niż wytworzenie
2. budowa zajmie zbyt dużo czasu
3. dla w miarę uniwersalnych problemów biznesowych (np. HR)
4. jeśli Twoje problemy biznesowe już zostały przez kogoś rozwiązane, i na rynku są już liderzy takich rozwiązań.

Kupno gotowego rozwiązania przyspiesza zarówno wejście na rynek, jak i jego skalowanie w przyszłości

**Stwórz własne, gdy...:**

1. obszar obsługiwany systemem jest kluczowym czynnikiem przewagi konkurencyjnej
2. koszty dostosowania gotowego produktu są bardzo duże, całkowity koszty wytworzenia jest dużo niższy (programowanie, testowanie, konfiguracja, skalowanie, ludzie (specjaliści), infrastruktura)
3. wytworzenie wymaga głębokiego zrozumienia specyfiki biznesu
4. bardzo zależy Ci na czasie, a dostawcy nie mogą zagwarantować realizacji niezbędnego zakresu w terminie.

### 2.6.2 Otwarte czy komercyjne?

Decyzja o tym, czy zakupić rozwiązanie licencjonowane, czy też budować własne w oparciu o technologie otwarte (ang. open-source), to kolejny dylemat, przed którym stoją kierownicy projektów czy projektanci architektur systemów IT.

**Wybierz rozwiązanie otwarte (open-source), gdy:**

1. ktoś już stworzył to, czego potrzebujesz?
2. kod jest dobrze wspierany przez dużą i aktywną społeczność (np. na dużo twórców i commit'ów na GitHub.com)
3. całkowity koszt posiadania takiego rozwiązania (TCO: Total Cost of Ownership) jest niski
4. koresponduje to z wizerunkiem Twojej marki
5. potrzebujesz nie tylko aplikacji, ale też kodu źródłowego.

**Wybierz rozwiązanie komercyjne, gdy:**

1. jest prostsze w dostosowaniu do Twoich potrzeb i wdrożeniu
2. jest standardem branżowym (np. arkusze kalkulacyjne)
3. potrzebujesz wiarygodnego wsparcia
4. całkowity koszt posiadania rozwiązania otwartego (TCO: Total Cost of Ownership) jest wysoki
5. gdy musisz zagwarantować, że nie naruszasz czyichś praw autorskich
6. istotne jest bezpieczeństwo (choć nie zawsze...).

### 2.6.3 U siebie czy “w chmurze”?

Systemy IT/uczenia maszynowego możesz utrzymywać na własnej infrastrukturze, lokalnie w organizacji, lub też “w chmurze” (prywatnej lub publicznej).

**Utrzymuj na swoich serwerach**, gdy:

1. chcesz mieć pełną kontrolę nad infrastrukturą i danymi
2. całkowite koszty posiadania będą niższe niż w chmurze
3. będziesz musiał często przysyłać duże ilości danych z własnych rozwiązań do “chmury” (egress) i z powrotem (ingres)
4. rozwiązanie chmurowe może generować nieakceptowalne problemy z latencją (szybkością odpowiedzi serwera)
5. stabilność działania jest krytyczna, i niemożliwa do zapewnienia przez dostawcę.

**Utrzymuj w chmurze**, gdy:

1. zależy Ci na koncentracji na swoim biznesie (uwolnieniu uwagi z konieczności monitoringu i rozwoju IT)
2. chcesz obniżyć koszty posiadania i utrzymania infrastruktury IT
3. istotna jest elastyczność skalowania rozwiązania (sezonowość, rozwój)
4. potrzebujesz ciągłego, wiarygodnego wsparcia
5. zasady firmy i reguły bezpieczeństwa dopuszczają takie rozwiązanie.



## Chapter 3

# Modelowanie architektur

### 3.1 Cykl życia modelu uczenia maszynowego

Na typowy projekt uczenia maszynowego składają się następujące etapy:

1. Przygotowanie danych
2. Modelowanie (trening i ewaluacja)
3. Udostępnianie modelu
4. Monitoring
5. Dotrenowywanie.

Za realizację każdego z nich odpowiada najczęściej osobny **komponent** (na rysunku powyżej zaprezentowany w formie owalnej), który przyjmuje na wejściu oraz generuje na wyjściu tzw. **artefakt** (oznaczony jako prostokąt).

W realnych projektach sytuacja jest bardziej złożona: na przygotowanie danych, modelowanie, udostępnianie i monitoring składa się wiele etapów cząstkowych (realizowanych przez odpowiednie komponenty):

1. Przygotowanie danych
  1. pozyskanie danych
  2. walidacja danych
  3. transformacja danych
2. Modelowanie:
  1. trening i ewaluacja
  2. walidacja finalnego modelu
  3. tuning hiperparametrów
3. Udostępnianie modelu
  1. przygotowanie do udostępnienia
  2. (ciągła) integracja (CI: Continuous Integration)
  3. (ciągłe) uruchamianie (CD: Continuous Deployment)
4. Monitoring

1. monitoring i identyfikacja dryfu danych
2. monitoring i identyfikacja dryfu modeli
3. monitoring infrastruktury
5. Dotrenowywanie
  1. generowanie sygnału dotrenowania
  2. wybór nowych danych treningowych
  3. dotrenowanie.

Realizację tych procesów wspiera wiele dedykowanych systemów, wyspecjalizowanych w realizacji konkretnych zadań. Szczególnie istotną rolę odgrywają:

1. repozytoria cech (ang. feature stores)
2. systemy monitoringu eksperymentów
3. repozytoria modeli (ang. model stores)
4. systemy ciągłej integracji i udostępniania (ang. continuous integration (CI) and deployment (CD))
5. systemy monitorujące modele
6. repozytoria metadanych generowanych przez cały proces (ang. metadata stores).

Wybór technologii wykorzystanych do implementacji poszczególnych komponentów zależy od stopnia złożoności projektu. W prostych przedsięwzięciach można z powodzeniem wykorzystać pliki CSV, skrypty Python czy aplikacje typu Flask, bardziej złożone wymagają wdrożenia dedykowanych rozwiązań dostępnych za darmo (np. MLFlow, FastApi czy darmowa wersja WandB.ai), zaś zaawansowane dedykowanych systemów takich jak Feast, Kubeflow, Heroku czy Arize.

Pejzaż dostępnych w tym zakresie rozwiązań jest bardzo dynamiczny: pojawia się tu coraz więcej nowych rozwiązań, czemu towarzyszy mniej lub bardziej dynamiczny rozwój już istniejących.

### Przydatne źródła

1. Przykład kompletnej architektury projektu uczenia maszynowego, ze świetnie opisanymi poszczególnymi etapami jej konstrukcji, opisany jest tutaj.
2. Bardzo dobrym wprowadzeniem w zagadnienie cyklu życia projektu maszynowego, i ogólniej w tematykę MLOps jest artykuł dostępny tutaj.
3. Nieco bardziej rozbudowane wprowadzenie do różnych poziomów automatyzacji procesów MLOps można znaleźć tutaj:
  1. poziom 0: manual pipelines
  2. poziom 1: continuous training
  3. poziom 2: CI/CD
4. Interesujący przegląd narzędzi wspomagających zarządzanie cyklem życia oraz poszczególnymi etapami projektu data science jest dostępny tutaj.



## 3.2 Demo: trening i ewaluacja modelu

*Uwaga: komplet wersji demonstracyjnych, ćwiczeń i rozwiązań oraz rekomendacje dotyczące środowiska uruchomieniowego znajdziesz tutaj:*

[https://github.com/wodecki/ASI\\_2022](https://github.com/wodecki/ASI_2022)

### Cel

Nasz pierwszy kod będzie stanowił punkt wyjścia dla kolejnych ćwiczeń. Stworzymy sekwencję skryptów Python, których zadaniem będzie wczytanie danych treningowych, stworzenie modelu i jego ewaluacja na danych testowych.

Do modelowania wykorzystamy dane syntetyczne, które będą możliwe do zamodelowania z wykorzystaniem prostego modelu regresji liniowej.

Stworzony dzięki temu szkielet oprogramowania będziemy mogli później rozwijać w następujących wymiarach:

1. Urealnienie problemu:
  1. przejście od danych syntetycznych do danych realnych, np. modelowania cen mieszkań
  2. zmianę typu problemu uczenia maszynowego: przejście od regresji do np. klasyfikacji
2. Zamiany prostych skryptów Python na dedykowane biblioteki, służące np. do monitorowania modeli czy pre-processingu danych
3. Zmiany środowisk uruchomieniowych: z własnego komputera na docker czy kubernetes.

W ćwiczeniu tym przedstawimy też podstawowe zasady projektowania architektury kodu uczenia maszynowego, w szczególności pojęcia artefaktów, komponentów i wizualizacji relacji pomiędzy nimi z wykorzystaniem diagramów.

### Lista kontrolna

Skrypt, który stworzymy, będzie realizował następujące zadania:

1. Wczytanie danych
  - ☐ Wczytuje plik treningowy data\_init.csv
  - ☐ Zapisuje go do pliku data\_train.csv
2. Trenowanie modelu
  - ☐ Wczytuje dane treningowe data\_train.csv
  - ☐ Przygotowuje dane do modelowania
  - ☐ Trenuje model korzystając algorytmu LinearRegression
  - ☐ Drukuje na ekranie parametry modelu
  - ☐ Zapisuje model do pliku model\_1.0.pkl w folderze model
3. Ewaluacja modelu
  - ☐ Wczytuje model z pliku model/model\_1.0.pkl

- ☐ Wczytuje dane treningowe z pliku data/data\_test.csv
- ☐ Generuje predykcje i ocenia model
- ☐ Drukuje wyniki ewaluacji na ekranie komputera

### Architektura

#### Artefakty

1. Wejście
  1. Inicjalny plik treningowy: data\_init.csv
  2. Plik testowy: data\_test.csv
2. Wyjście
  1. model: model\_1.0

#### Komponenty

1. Wczytanie danych (1. read.py)
  1. Wejście: data\_init.csv
  2. Działania:
    1. wczytuje dane
    2. zapisuje na dysku
  3. Wyjście: data\_train.csv
2. Trening (2. train.py):
  1. Wejście: data\_train.csv
  2. Działania:
    1. Wczytuje dane treningowe data\_train.csv
    2. Przygotowuje dane do modelowania
    3. Trenuje model korzystając algorytmu LinearRegression
    4. Drukuje na ekranie parametry modelu
    5. Zapisuje model do pliku model\_1.0.pkl w folderze model
  3. Wyjście: model\_1.0.pkl
3. Ewaluacja (3. test.py):
  1. Wejście:
    1. data\_test.csv
    2. model\_1.0.pkl
  2. Działania:
    1. Wczytuje model z pliku model/model\_1.0.pkl
    2. Wczytuje dane treningowe z pliku data/data\_test.csv
    3. Generuje predykcje i ocenia model
    4. Drukuje wyniki ewaluacji na ekranie komputera
  3. Wyjście: ekran

### Decyzje

Projektując to rozwiązanie, musimy podjąć następujące decyzje:

1. Wybór miary jakości modelu
  1. RMSE
  2. R2
2. Algorytm(y) ML: LinearRegression

## Podsumowanie

### 3.3 Ćwiczenie: trening i ewaluacja modelu

#### Zadanie 1. Zapis danych ewaluacyjnych do pliku csv

Zmodyfikuj moduł ewaluacyjny tak, by wyniki były zapisywane do pliku `model_eval.csv` w folderze `evaluation`.

Zapisz do niego:

- stempel czasu (aktualny czas) w kolumnie `time_stamp`
- wersję modelu (liczbę po słowie `model` w nazwie modelu) w kolumnie `'Model version'`
- nazwę miary w kolumnie `Measure`
- wartości obu miar (RMSE i `r2`) w kolumnie `score`

Wykorzystaj w tym celu skrypt 3. `evaluate.py`

#### Lista kontrolna

Zmodyfikowany skrypt zapisuje w pliku `evaluation/model_eval.csv`:

- ☐ stempel czasu (aktualny czas) w kolumnie `time_stamp`
- ☐ wersję modelu (liczbę po słowie `model` w nazwie modelu) w kolumnie `'Model version'`
- ☐ nazwę miary w kolumnie `Measure`
- ☐ wartości obu miar (RMSE i `r2`) w kolumnie `score`

#### Zadanie 2. Zapis danych ewaluacyjnych do systemu MLFlow lub Weights&Biases

Zmodyfikuj moduł ewaluacyjny tak, by wyniki były zapisywane do wybranego systemu monitorującego jakość modelu, np. MLflow lub Weights&Biases ([www.wandb.com](http://www.wandb.com)).

#### Lista kontrolna

Zmodyfikowany skrypt zapisuje w wybranym systemie monitoringowym następujące artefakty:

- ☐ stempel czasu (aktualny czas) w kolumnie `time_stamp`
- ☐ wersję modelu (liczbę po słowie `model` w nazwie modelu) w kolumnie `'Model version'`
- ☐ nazwę miary w kolumnie `Measure`
- ☐ wartości obu miar (RMSE i `r2`) w kolumnie `score`.

*Uwaga: komplet wersji demonstracyjnych, ćwiczeń i rozwiązań oraz rekomendacje dotyczące środowiska uruchomieniowego znajdziesz tutaj:*

[https://github.com/wodecki/ASI\\_2022.git](https://github.com/wodecki/ASI_2022.git)

### 3.4 Modelowanie architektury - wprowadzenie

Początkowo proste przepływy zadań w projektach uczenia maszynowego mogą z czasem ewoluować w bardzo złożone. Dlatego od samego początku warto stosować metody projektowania zapewniające kontrolę nad architekturą całości.

Rozwiązania takie jak Kubeflow czy Dataiku pozwalają wizualizować zaimplementowane już przepływy, jest to jednak tylko wizualizacja przepływu zaimplementowanego wcześniej np. w formie plików JSON czy skryptów Python.

Do zaprojektowania przepływu uczenia maszynowego można wykorzystać systemy do wspomagające modelowanie procesów. Przykładowo, dostępne na licencji otwartej Diagrams.net, oferowane w szczególności jako aplikacja Google.

Bloki diagrams.net możemy wykorzystać do wizualizacji komponentów i artefaktów przepływu ML. Każdy blok może mieć dedykowany zestaw cech (properties), które można wykorzystać do wprowadzenia własnych meta-opisów.

Szczególną cechą jest link do zasobu zewnętrznego skojarzonego z blokiem. Jeśli wykorzystamy go do połączenia z repozytorium GitHub, rozwiązanie będzie nam służyć nie tylko do wizualizacji architektury, ale też monitoringu aktualnego stanu komponentów i artefaktów. Przykładowo, każdy współpracownik w projekcie będzie miał dostęp do aktualnych miar jakości modelu.

W efekcie, proste, darmowe systemy wspomagające modelowanie procesów czy architektur mogą posłużyć do stworzenia prostych systemów zarządzania procesami MLOps.

### 3.5 Modelowanie architektury - organizacja

Modelowanie architektur systemów uczenia maszynowego można zrealizować w następujących krokach:

1. Organizacja
2. Specyfikacja wymagań
3. Modelowanie.

Cały proces warto rozpocząć od stworzenia środowiska pracy (zarówno własnej, jak i grupowej). W pierwszej kolejności załóż dedykowany folder (opcjonalnie: udostępnij go współpracownikom), skonfiguruj środowisko zarządzania projektem i utwórz dedykowane repozytorium w systemie Git (np. GitHub).

Następnie zdefiniuj problem i określ zakres modelowania. Chcesz modelować architekturę tylko dla procesu trenowania? A może wnioskowania na produkcji? Czy też zaprojektować pełny cykl MLOps. Przydatny może być tu artykuł dostępny tutaj.

W ostatnim kroku stwórz w dedykowanym folderze pliki odpowiadające pierwszej wersji architektury (w oparciu o swoją intuicję i poprzednie doświadczenia, na razie bez tworzenia szczegółowej specyfikacji). Mogą być to szablony (puste

### 3.6. MODELOWANIE ARCHITEKTURY - SPECYFIKACJA WYMAGAŃ<sup>29</sup>

pliki lub zawierające jedynie podstawowe elementy) komponentów, artefaktów oraz ich dokumentacji. Na koniec wprowadź tę strukturę do repozytorium GitHub.

#### Przydatne źródła

Świetny opis procesu projektowania pełnej architektury MLOps znajdziesz tutaj.

## 3.6 Modelowanie architektury - specyfikacja wymagań

Mając już przygotowane środowisko pracy, przystąp do tworzenia specyfikacji wymagań. Sugeruję w tym celu określić:

1. wymagania funkcjonalne w formie listy kontrolnej
2. specyfikację artefaktów
3. specyfikację komponentów
4. kluczowe decyzje, które należy podjąć implementując rozwiązanie.

---

#### Lista kontrolna:

1. Specyfikuje zakres działania komponentu
2. Określa standard wykonania
3. Jest zbiorem kryteriów odbioru

---

**Artefakty** to lista obiektów (najczęściej plików z danymi lub modeli), które mogą być:

1. Wsadem do komponentu („wejscie”)
2. Powstawać w wyniku uruchomienia komponentu („wyjscie”).

W przypadku plików warto określić:

1. Miejsca i nazwy
2. Formatu

W przypadku danych przydatne jest też wskazanie:

1. Nazw i znaczenia poszczególnych pól
2. Typów zmiennych (np. numeryczne, kategoryjne, czas, etc.).

---

Specyfikacja **komponentów** określa:

1. Wejścia (artefakt wejściowy)

2. Procedury działań (w oparciu o listę kontrolną)
3. Wyjścia (artefakt wyjściowy).

Działania realizowane przez komponent warto pisać tak, by później posłużyły jako dokumentacja wewnętrzna kodu (ang. one-line docstrings, por. np. <https://peps.python.org/pep-0257/>).

Czynność tę i tak z pewnością trzeba będzie wykonać w przyszłości. Realizacja tego zadania **PRZED** programowaniem:

1. Ułatwi zrozumienie celu i idei działania kodu
2. Może istotnie uprościć samo programowanie w środowiskach IDE wykorzystując technologie typu GitHub Copilot (<https://copilot.github.com>).

---

W kolejnym kroku wskaż najważniejsze **decyzje**, jakie trzeba będzie podjąć tworzyć dany komponent. Mogą być one później podjęte np. podczas spotkań zespołu projektowego.

---

Całość podsumuj np. w prostym arkuszu i zapisz w folderze ze specyfikacją projektu.

### 3.7 Modelowanie architektury - modelowanie

Mając przygotowaną szczegółową specyfikację działania modułu, stwórz szkic architektury korzystając np. z pakietu diagrams.net (możesz z niego korzystać w ramach aplikacji Google).

Umieść na schemacie:

1. komponenty
2. artefakty
3. oraz relacje pomiędzy nimi.

W kolejnym kroku, podłącz poszczególne komponenty i artefakty do odpowiednich plików w repozytorium GitHub.

Aby podłączyć specyfikację artefaktu/komponentu stwórz dedykowane pole (np. *Description*), i wklej do niego hiperłącze do specyfikacji. Uwaga: ze względu na ograniczoną liczbę znaków, sugeruję wcześniej skrócić link do dokumentacji korzystając z serwisu takiego jak [www.bit.ly](http://www.bit.ly).

---

Z czasem rozwój systemu wymusza modyfikację architektury. Aby utrzymać przejrzystość architektury, wart porządkować komponenty i artefakty w osobne struktury.

Przydatne może być wtedy uporządkowanie poszczególnych etapów modelowania w osobnych folderach.

W pakiecie `diagrams.net` mogą nam w tym pomóc  *pionowe kontenery* (ang. *vertical containers*). Pozwalają one na grupowanie komponentów i artefaktów, i w efekcie uproszczenie całej wizualizacji.

### Podsumowanie

Tworzenie aplikacji warto rozpocząć od jej zaprojektowania.

Projektowanie architektur uczenia maszynowego nie musi być poprzedzone ich tworzeniem. Z powodzeniem można wykorzystać w tym celu ogólnodostępne, darmowe rozwiązania wspomagające projektowanie schematów.

Korzyści z takiego podejścia to:

1. Wizualizacja całości rozwiązania (komponenty, artefakty i relacje pomiędzy nimi)
2. Szybki dostęp do aktualnych wersji komponentów i artefaktów dzięki połączeniu z GitHub
3. Tworzenie własnych meta-opisów dzięki personalizowanym cechom obiektów.

### Przydatne źródła

Świetny opis procesu projektowania pełnej architektury MLOps znajdziesz tutaj.

## 3.8 Demo: modelowanie

Na tym krótkim filmie pokazuję, w jaki sposób można wykorzystać oprogramowanie `Diagrams.net` do stworzenia prostej architektury systemu uczenia maszynowego. Poniżej znajdziesz listę kontrolną, specyfikację artefaktów i komponentów oraz decyzje, jakie należało podjąć w procesie projektowania.

### Lista kontrolna

Skrypt, który stworzymy, będzie realizował następujące zadania:

1. Wczytanie danych
  - ☐ Wczytuje plik treningowy `data_init.csv`
  - ☐ Zapisuje go do pliku `data_train.csv`
2. Trenowanie modelu
  - ☐ Wczytuje dane treningowe `data_train.csv`
  - ☐ Przygotowuje dane do modelowania
  - ☐ Trenuje model korzystając algorytmu `LinearRegression`
  - ☐ Drukuje na ekranie parametry modelu
  - ☐ Zapisuje model do pliku `model_1.0.pkl` w folderze `model`

### 3. Ewaluacja modelu

- ☐ Wczytuje model z pliku model/model\_1.0.pkl
- ☐ Wczytuje dane treningowe z pliku data/data\_test.csv
- ☐ Generuje predykcje i ocenia model
- ☐ Drukuje wyniki ewaluacji na ekranie komputera

## Architektura

### Artefakty

1. Wejście
  1. Inicjalny plik treningowy: data\_init.csv
  2. Plik testowy: data\_test.csv
2. Wyjście
  1. model: model\_1.0

### Komponenty

1. Wczytanie danych (1. read.py)
  1. Wejście: data\_init.csv
  2. Działania:
    1. wczytuje dane
    2. zapisuje na dysku
  3. Wyjście: data\_train.csv
2. Trening (2. train.py):
  1. Wejście: data\_train.csv
  2. Działania:
    1. Wczytuje dane treningowe data\_train.csv
    2. Przygotowuje dane do modelowania
    3. Trenuje model korzystając algorytmu LinearRegression
    4. Drukuje na ekranie parametry modelu
    5. Zapisuje model do pliku model\_1.0.pkl w folderze model
  3. Wyjście: model\_1.0.pkl
3. Ewaluacja (3. test.py):
  1. Wejście:
    1. data\_test.csv
    2. model\_1.0.pkl
  2. Działania:
    1. Wczytuje model z pliku model/model\_1.0.pkl
    2. Wczytuje dane treningowe z pliku data/data\_test.csv
    3. Generuje predykcje i ocenia model
    4. Drukuje wyniki ewaluacji na ekranie komputera
  3. Wyjście: ekran

## Decyzje

Projektując to rozwiązanie, musimy podjąć następujące decyzje:

1. Wybór miary jakości modelu



1. RMSE
2. R2
2. Algorytm(y) ML: LinearRegression

### Podsumowanie

## 3.9 Podsumowanie: modelowanie architektur

Cykl życia modelu uczenia maszynowego jest złożony - dlatego już na samym początku projektu zacząć pracować nad architekturą docelowego rozwiązania.

Większość rozwiązań wizualizuje architekturę w oparciu o już stworzony kod. Do zaprojektowania przepływu uczenia maszynowego można jednak wykorzystać systemy do wspomagające modelowanie procesów. Przykładowo, dostępne na licencji otwartej Diagrams.net, oferowane w szczególności jako aplikacja Google.

Proces modelowania proponujemy zrealizować w trzech krokach: rozpocząć od organizacji, potem stworzyć wstępną specyfikację, a następnie model architektury.

Jeśli wykorzystamy go do połączenia z repozytorium GitHub, rozwiązanie będzie nam służyć nie tylko do wizualizacji architektury, ale też monitoringu aktualnego stanu komponentów i artefaktów. Przykładowo, każdy współpracownik w projekcie będzie miał dostęp do aktualnych miar jakości modelu.



## Chapter 4

# Monitoring modeli

Po udostępnieniu model uczenia maszynowego jest wykorzystywany w środowisku produkcyjnym. Jakość jego predykcji (lub innych zadań, do których został wytrenowany) może być oceniana poprzez porównanie zgodności prognoz z rzeczywistością. Przykładowo, model prognozujący popyt na wybrany produkt w danym horyzoncie czasowym (np. tygodnia) może być oceniony po upływie tego czasu poprzez porównanie predykcji z faktycznym popytem. Etap ten nazywamy **ewaluacją**. Artefakty procesu ewaluacji to najczęściej **miary efektów** (ang. *scoring*) oraz **zbiory doświadczeń** (zawierające najczęściej stempel czasu, prognozę oraz stan faktyczne).

Modele uczenia maszynowego nie są doskonałe. Ich jakość zależy od typu problemu (klasyfikacja? regresja? predykcja szeregu czasowego? segmentacja?) i jest mierzona w procesie trenowania (przed akceptacją do wdrożenia produkcyjnego).

Często okazuje się, że początkowo bardzo dobra jakość modelu, z czasem istotnie się obniża. Innymi słowy, model początkowo świetnie prognozujący np. popyt na dany produkt po pewnym czasie zaczyna popełniać coraz więcej błędów.

Przyczyn takiej sytuacji jest kilka. Po pierwsze, może wystąpić **zmiana w zmiennej prognozowanej**. Przykładowo, w przypadku sprzedaży inflacja może w naturalny sposób podnieść poziom wszystkich cen, czego efektem może być podniesie wolumenu sprzedaży - model wytrenowany na danych historycznych będzie w takiej sytuacji prognozował niższe wartości niż realne.

Po drugie, mogą **zmienić się zmienne objaśniające (cechy)**. Użytkownicy mogą zmieniać urządzenia, z których korzystają, na rynku mogą pojawiać się produkty nowego typu, etc. Przykładowo, model prognozujący popyt na mieszkania wytrenowany na danych, w których były dostępne jedynie budynki 3-4 piętrowe przestaną działać w sytuacji, gdy w mieście pojawią się 10-piętrowe wieżowce.

Kolejną przyczyną deaktualizacji modeli mogą być **zmiany w relacjach pomiędzy cechami a zmiennymi prognozowanymi**. W naturalny sposób mogą zmieniać się gusta klientów, pojawiać nowe trendy zakupowe, nowe kampanie marketingowe, konkurencja, substytuty produktów czy też po prostu zmiany globalne.

Wszystkie te czynniki wpływają na stopniowe pogarszanie jakości modeli stosowanych produkcyjnie, co jest motywacją dla wdrożenia procesów **monitoringu i udoskonalania modeli**.

#### Przydatne źródła

1. Podstawowe wprowadzenie do przyczyn, skutków i metod zapobiegania dryfowi modeli można znaleźć tutaj.
2. Bardzo dobre wprowadzenie w różne strategie postępowania w przypadku detekcji dryfu modelu znajduje się tutaj.
3. Interesujący przegląd narzędzi wspomagających monitoring modeli (uwaga: stworzony przez jednego z dostawców tego typu narzędzi) dostępny jest tutaj.

## 4.1 Monitoring modeli - wprowadzenie

Ocena jakości modelu jest kluczowym elementem procesu trenowania: decyzję o przekazaniu modelu do wdrożenia produkcyjnego podejmujemy kierując się w dużej mierze wartością odpowiedniej miary tej jakości.

#### Metody ewaluacji

Sposób oceny jakości zależy przede wszystkim od typu problemu, jaki ma rozwiązać model. Miary ewaluacji będą różne dla problemów klasyfikacji, regresji, prognozy szeregu czasowego czy segmentacji.

#### Doświadczenia

Podczas korzystania z modelu w środowisku predykcyjnym, możemy z czasem gromadzić **doświadczenia**: porównywać prognozę modelu z tym, co wydarzyło się faktycznie, i gromadzić te porównania w osobnym **repozytorium**.

Doświadczenia gromadzimy w miarę upływu czasu, i dopiero po pewnym czasie możemy ocenić, na ile dobrze nasz model sprawdza się w nowej (innej niż podczas treningu) rzeczywistości. W tym celu wykorzystujemy właśnie repozytoria doświadczeń.

---

Predykcję realizować możemy realizować:

1. w ciągły sposób (przetwarzanie strumieniowe, predykcja on-line)
2. ... lub w pakietach (przetwarzanie pakietowe, predykcja off-line))

W efekcie, dane na potrzeby ewaluacji możemy więc pozyskiwać:

1. rekord po rekordzie < predykcja on-line
2. w pakietach < predykcja off-line.

### Potok wnioskowania

Sekwencję działań, których celem jest monitoring jakości modelu, można uporządkować w tzw. **potok wnioskowania** (ang. inference pipeline). W dużym uproszczeniu, sprowadza się on do cyklicznego gromadzenia doświadczeń w dedykowanym repozytorium (bazie danych czy zwykłym pliku), obliczania miar jakości modelu i gromadzenia tych miar w kolejnym repozytorium.

Na potrzeby kolejnych rozważań uprościmy notację, i pełny proces wnioskowania oznaczać będziemy symbolem jednego bloku **IF**.

Schemat powyżej ilustruje kolejne etapy wnioskowania produkcyjnego, których rezultatem są nie tylko predykcje, ale też kolejne doświadczenia (gromadzone w repozytorium oznaczonym symbolem T) oraz szereg czasowy miar jakości modelu (gromadzony w repozytorium oznaczonym symbolem S).

Jeśli miara jakości przekroczy pewien założony przez nas próg, system może wygenerować **sygnał dotrenowania**.

### Podsumowanie

Jak widać, cały proces uczenia maszynowego jest sekwencją wnioskowań raz na jakiś czas przerywanych sygnałem dotrenowania, który uruchamia proces udoskonalania modelu (treningu). Kluczową decyzją w tym przepływie jest określenie reguł generowania sygnału dotrenowania.

### Przydatne źródła

Interesującą platformą dedykowaną do monitoringu modeli jest <https://arize.com/>. Warto przeanalizować dostępne tam materiały, w szczególności <https://arize.com/ml-observability/>.

## 4.2 Demo: monitoring modelu

*Uwaga: komplet wersji demonstracyjnych, ćwiczeń i rozwiązań oraz rekomendacje dotyczące środowiska uruchomieniowego znajdziesz tutaj:*

[https://github.com/wodecki/ASI\\_2022](https://github.com/wodecki/ASI_2022)

### Cel

Celem naszego programu jest ewaluacja już wytrenowanego modelu na kolejnych partiach danych i zapis metryk jakości do osobnego pliku.

W module wykorzystamy:

- wytrenowany w poprzednim ćwiczeniu model regresji

- syntetyczne zbiory danych, zbliżone do zbioru treningowego, niemniej na tyle różne, by zidentyfikować potencjalny dryf modelu.

Stworzony dzięki temu program będziemy mogli później wykorzystać jako komponent procesu identyfikacji dryfu modelu.

### Lista kontrolna

Skrypt, który stworzymy, będzie realizował następujące zadania:

- ☐ Wczytanie wytrenowanego modelu `model/model_1.0.pkl`
- ☐ Wczytanie danych testowych:
  - ☐ wybór paczki danych testowych: `batch_no` w zakresie od 1 do 6
  - ☐ wczytanie odpowiedniej paczki
- ☐ Wygenerowanie predykcji modelu
- ☐ Obliczenie miar jakości:
  - ☐ RMSE
  - ☐  $r^2$
- ☐ Zapisanie w pliku `evaluation/model_eval.csv`:
  - ☐ stempla czasowego
  - ☐ nr paczki danych
  - ☐ wartości miary RMSE
  - ☐ wartości miary  $r^2$
  - ☐ UWAGA:
    - ☐ jeśli pliku nie istnieje: utworzenie go
    - ☐ w przeciwnym przypadku: uzupełnienie pliku (dodanie aktualnych rekordów).

### Architektura

#### Artefakty

1. Wejście
  1. Model: `model/model_1.0.pkl`
  2. Pliki testowe: `data/batch_n.csv`, z `n` w zakresie od 1 do 6
2. Wyjście
  1. Plik ewaluacyjny: `evaluation/model_eval.csv`

#### Komponenty

Jeden program 1. `Evaluate.py` realizujący zadania z listy kontrolnej.

#### Decyzje

Projektując to rozwiązanie, musimy podjąć następujące decyzje:

1. Wybór miary jakości modelu
  1. RMSE
  2.  $R^2$

## 4.3 Ćwiczenie: monitoring modelu

*Uwaga: komplet wersji demonstracyjnych, ćwiczeń i rozwiązań oraz rekomendacje dotyczące środowiska uruchomieniowego znajdziesz tutaj:*

[https://github.com/wodecki/ASI\\_2022](https://github.com/wodecki/ASI_2022)

---

Sterowanie parametrami uruchomieniowymi z linii komendą jest powszechną, dobrą praktyką we wdrożeniach produkcyjnych. Jest nie tylko wygodne i bezpieczne (brak konieczności zmiany skryptu przy zmianie parametrów), ale zapewnia też lepszą kontrolę procesu oraz umożliwia wykorzystanie pakietów dedykowanych do skanowania hiperparametrów (takich jak <https://hydra.cc/> czy <https://optuna.org/>)

Zmodyfikuj moduł demonstracyjny tak, by nazwy artefaktów wczytywane były do pliku z linii komend, a nie wprowadzane bezpośrednio w skrypcie Python.

Wykorzystaj w tym celu bibliotekę **argparse**.

Tak zmodyfikowany plik będziesz mogła/mógł wykorzystać potem w implementacji pełnego potoku monitoringu modelu i detekcji dryfu.

### Lista kontrolna

Zmodyfikowany skrypt umożliwia przekazanie z linii komend następujących parametrów:

- ☐ **batch\_no**: nr paczki danych testowych z folderu **data\**
- ☐ **model\_path**: ścieżka do pliku z wytrenowanym modelem.

### Przydatne źródła

1. Przystępne, ale kompleksowe wprowadzenie do biblioteki **argparse** znajdziesz tutaj.
2. Hydra.cc: biblioteka umożliwiająca zaawansowane sterowanie parametrami uruchomieniowymi.
3. Optuna.org: biblioteka oferująca inteligentne metody skanowania optymalizacji hiperparametrów w uczeniu maszynowym.

## 4.4 Detekcja dryfu modelu

Jak już wspominaliśmy wcześniej, dryf modelu może być spowodowany w szczególności:

1. Zmianą zmiennej prognozowanej (np. w przypadku sprzedaży: inflacją)
2. Zmianą zmiennych objaśniających (np. cech klientów, charakterystyk produktów, etc.)

3. Zmianą relacji pomiędzy zmiennych prognozowanymi a cechami (np. zmiana koniunktury, wprowadzenie substytutów, nowe kampanie marketingowe, etc.).

Poniżej przedstawimy najczęściej spotykane metody identyfikacji dryfu modelu i generowania sygnału dotrenowania.

### Metoda 1. Twardy próg

Najprostszą metodą identyfikacji dryfu modelu jest:

1. ustalenie pewnego “twardego” progu: minimalnego akceptowalnego poziom jakości modelu (np. maksymalny dopuszczalny poziom błędu RMSE)
2. uruchamianie sygnału dotrenowania w momencie, gdy miara jakości modelu przekroczy ten próg.

### Metoda 2. Test porównawczy

Kolejna metoda, tzw. **test porównawczy**, uruchamia sygnał w momencie, gdy **nowa miara jakości modelu jest gorsza niż wszystkie poprzednie wartości**. Na rysunku miara oznaczona czerwoną kropką jest większa niż wszystkie pozostałe, co jest źródłem “alarmu”.

Wadą tego rozwiązania jest często jego “nadwrażliwość”: sygnał dotrenowania uruchamiany jest zbyt często i niepotrzebnie.

### Metoda 3. Test istotności parametrycznej

Rozwiązaniem problemu “nadwrażliwości” testu porównawczego jest test istotności parametrycznej.

W tym podejściu sygnał dotrenowania uruchamiamy w sytuacji, gdy **aktualna wartość miary jakości jest gorsza (mniejsza lub większa) o dwa odchylenia standardowe od średniej poprzednich wartości**. Wybór *mniejsza/większa* zależy od tego, czy wyższa wartość miary świadczy o poprawie, czy też zmniejszeniu jakości modelu (por. rysunek powyżej).

Procedura postępowania jest w tym przypadku następująca:

1. Oblicz średnią poprzednich wartości miary
2. Oblicz odchylenie standardowe poprzednich wartości miary
3. Sprawdź, czy aktualna wartość miary jest gorsza (mniejsza lub większa) niż średnia  $\pm 2 \cdot \text{odchylenie standardowe}$ .

Problematiczne w tym podejściu mogą być sytuacje, gdy nasze odczyty pomiarów jakości:

1. nie układają się krzywą dzwonową
2. mają wartości odstające.



**Metoda 4. Testy nieparametryczne (detekcja wartości odstających)**

W przypadku testów nieparametrycznych sygnał dotrenowania uruchamiamy w sytuacji, gdy **aktualna wartość miary jakości jest zinterpretowana jako wartość odstająca i jest gorsza od poprzednich wartości**.

Procedura postępowania jest w tym podejściu następująca:

1. Dla zbioru poprzednich miar oblicz
  1. Q1 (pierwszy kwartył)
  2. Q3 (trzeci kwartył)
  3. Odległość międzykwartylową:  $IQR = Q3 - Q1$
2. Ustal granice wartości "normalnych":
  1. Dolna granica:  $Q1 - 1.5 IQR$
  2. Górna granica:  $Q3 + 1.5 IQR$
3. Dla nowej miary jakości modelu sprawdź, czy mieści się ona w dopuszczalnych granicach:  $Q1 - 1.5 IQR < miara < Q3 + 1.5 IQR$
4. Wygeneruj sygnał dotrenowania w sytuacji, gdy:
  1. nowa wartość miary jest poza tymi granicami i
  2. jest gorsza niż pozostałe (mniejsza lub większa, w zależności od typu miary).

Dla przypomnienia, pierwszy kwartył to wartość miary, od której 25% wszystkich miar w zbiorze jest mniejsza.

**Metoda 5. Testy hipotez statystycznych**

Często stosowaną metodą identyfikacji dryfu modelu jest sformułowanie i weryfikacja hipotezy, że rozkład jednego zbioru danych różni się od rozkładu innego zbioru. Podejście to może służyć zarówno do identyfikacji zmiany jakości modelu, jak tzw. dryfu danych prognozowanych.

Do weryfikacji takich hipotez można wykorzystać **testy hipotez statystycznych**. W uczeniu maszynowym szczególnie popularne są **test Kolmogorowa-Smirnowa** oraz **test Chi-squared**. Osoby zainteresowane zachęcam do lektury artykułów z sekcji *Przydatne źródła*.

**Przydatne źródła**

1. Bardzo dobrą prezentację różnych metod identyfikacji dryfu modelu można znaleźć tutaj.
2. Wprowadzenie do testowania hipotez dostępne jest w tej lekcji na Khan Academy.
3. Wykorzystanie testów hipotez statystycznych w identyfikacji dryfu modelu w przystępny sposób opisane jest tutaj.

## 4.5 Demo: detekcja dryfu

*Uwaga: komplet wersji demonstracyjnych, ćwiczeń i rozwiązań oraz rekomendacje dotyczące środowiska uruchomieniowego znajdziesz tutaj:*

[https://github.com/wodecki/ASI\\_2022](https://github.com/wodecki/ASI_2022)

### Cel

Celem naszego programu jest detekcja dryfu w oparciu o wyniki ewaluacji z poprzedniego ćwiczenia, zarejestrowane w pliku `evaluation/model_eval.csv`.

Stworzony dzięki temu program będziemy mogli później wykorzystać jako komponent pełnego potoku MLOps.

### Lista kontrolna

Skrypt, który stworzymy, będzie realizował następujące zadania:

- ☐ Wczytanie wyników ewaluacji z pliku `evaluation/model_eval.csv`.
- ☐ Przygotowanie tych danych do obliczenia testów: “twardego” i parametrycznego
  - ☐ Identyfikacja ostatniego odczytu
  - ☐ Lista logów miar jakości: RMSE i r2
- ☐ Przeprowadzenie testów i wydruk ich wyników na ekranie:
  - ☐ test “twardy”:
    - ☐ Dla RMSE rozpoznajemy dryf (przypisujemy wartość TRUE), jeśli nowe RMSE jest większe od średniej wszystkich poprzednich RMSE
    - ☐ Dla r2 identyfikujemy dryf (przypisujemy wartość TRUE), jeśli nowe r2 jest mniejsze od średniej wszystkich poprzednich r2
  - ☐ test parametryczny:
    - ☐ Dla RMSE rozpoznajemy dryf (przypisujemy wartość TRUE), jeśli nowe RMSE jest większe od średniej wszystkich poprzednich  $RMSE + 2 \cdot \text{odchylenie standardowe}$  (wszystkich poprzednich RMSE)
    - ☐ Dla r2 identyfikujemy dryf (przypisujemy wartość TRUE), jeśli nowe r2 jest mniejsze od średniej wszystkich poprzednich  $r2 - 2 \cdot \text{odchylenie standardowe}$  (wszystkich poprzednich r2).

### Architektura

#### Artefakty

1. Wejście
  1. Pliku z ewaluacjami: `evaluation/model_eval.csv`.
2. Wyjście
  1. wydruk wyników testów na ekranie.

#### Komponenty

Jeden program `1.detect_model_drift.py` realizujący zadania z listy kontrolnej.

## 4.6 Ćwiczenia: detekcja dryfu i dotrenowanie

### 4.6.1 Ćwiczenie: detekcja dryfu

*Uwaga: komplet wersji demonstracyjnych, ćwiczeń i rozwiązań oraz rekomendacje dotyczące środowiska uruchomieniowego znajdziesz tutaj:*

[https://github.com/wodecki/ASI\\_2022](https://github.com/wodecki/ASI_2022)

---

W pliku `1.detect_model_drift.py` dodaj funkcjonalności umożliwiające:

- ☐ Przeprowadzenie testu nieparametrycznego (IQR) i wydruk jego wyniku na ekranie
  - ☐ Dla RMSE rozpoznajemy dryf (przypisujemy wartość TRUE), jeśli nowe RMSE jest większe od trzeciego kwartyłu RMSE + 1.5\*IQR
  - ☐ Dla r2 identyfikujemy dryf (przypisujemy wartość TRUE), jeśli nowe r2 jest mniejsze od pierwszego kwartyłu r2 - 1.5\*IQR
- ☐ Wygenerowanie sygnału dotrenowania w sytuacji, gdy co najmniej jeden z testów dał wynik pozytywny
- ☐ Zapis sygnału dryfu w przypadku jego zastąpienia do pliku `evaluation/model_drift.csv`
  - ☐ Jeśli ten plik jeszcze nie istnieje: skrypt powinien go stworzyć
  - ☐ Jeśli ten plik już istnieje: powinien dodać do niego nowe rekordy
  - ☐ Zapisywany rekord powinien zawierać następujące pola:
    - ☐ Stempel czasu
    - ☐ Wersja modelu
    - ☐ Uzasadnienie wygenerowania sygnału dryfu: 6 kolumn (3 testy po dwa parametry) z wartościami TRUE (test pozytywny: sygnał dryfu) lub FALSE (test negatywny: brak sygnału dryfu)
- ☐ Przykładowy kształt pliku:

### 4.6.2 Ćwiczenie: uruchomienie dotrenowania

*Uwaga: komplet wersji demonstracyjnych, ćwiczeń i rozwiązań oraz rekomendacje dotyczące środowiska uruchomieniowego znajdziesz tutaj:*

[https://github.com/wodecki/ASI\\_2022](https://github.com/wodecki/ASI_2022)

---

To proste ćwiczenie zostawiliśmy na koniec: pozwoli one zamknąć pętlę MLOps.

Na końcu pliku `1.detect_model_drift.py` wykorzystaj bibliotekę Python `subprocess` do uruchomienia pliku `2. train_model.py` w przypadku wystąpienia dryfu.

## 4.7 Monitoring modeli - podsumowanie

Kluczowe etapy w cyklu życia modelu to trenowanie i wykorzystanie produkcyjne (inferencja).

Modele nie są doskonałe nie tylko na końcu procesu trenowania: z czasem mogą się one też deaktualizować.

Najważniejsze przyczyny dryfu modelu to:

1. Zmiana w zmiennej prognozowanej
2. Zmiana w zmiennych objaśniających (cechach)
3. Zmiany w relacjach cech i zmiennych prognozowanych.

Rozwiązaniem jest ciągły monitoring modeli.

Najbardziej popularne metody detekcji dryfu to:

1. Test „twardy” (porównawczy)
2. Test istotności parametrycznej
3. Test nieparametryczny (identyfikacja wartości odstających IQR).

Detekcja dryfu jest sygnałem konieczności dotrenowania modelu.

W efekcie, cykl życia modelu jest sekwencją procesów trenowania i wnioskowania produkcyjnego - aż do momentu wycofania go z użytkowania.

## Chapter 5

# Konteneryzacja

### 5.1 Wprowadzenie

Wyobraźmy sobie bardzo elementarny przepływ trenowania modelu uczenia maszynowego, na który składają się etapy przygotowania danych, trenowania i ewaluacji modelu.

Proces ten najczęściej nie jest prostą sekwencją kroków, ale procesem zawierającym różne rozgałęzienia i artefakty pochodzące z różnych źródeł. We wdrożeniach produkcyjnych dodatkową komplikacją jest równoległa praca w często osobnych środowiskach twórców aplikacji, oraz inżynierów odpowiedzialnych za ich uruchamianie i utrzymanie.

Na szczęście istnieje wiele metod i technik umożliwiających efektywne zarządzanie tak złożonymi procesami i systemami. Wśród nich jedną z ważniejszych jest tzw. konteneryzacja, która najczęściej wdrażana jest z wykorzystaniem środowiska Docker.

#### Korzyści

Najczęściej wskazywane przez praktyków korzyści z wykorzystania konteneryzacji w projektach uczenia maszynowego to:

1. **Przenaszalność.** Możliwość uruchomienia rozwiązania w różnych środowiskach, stacjach roboczych i systemach
2. **Reużywalność.** Możliwość wykorzystania komponentów dostarczonych przez innych, oraz własnych, stworzonych wcześniej, np. w innych projektach
3. **Reprodukowalność.** Możliwość odtworzenia wyników innych zespołów
4. **Produktywność.** Istotne usprawnienie pracy własnej oraz zespołowej.

Czego się nauczysz?

Mam nadzieję, że po kolejnych lekcjach będziesz potrafił/a:

**Stworzyć, udostępnić i uruchomić obraz pojedynczej aplikacji**

1. Zaprojektować obrazu z wykorzystaniem Dockerfile
2. Zbudować go
3. Udostępnić ten obraz innym
4. Pobrać gotowy obraz i uruchomić go w postaci kontenera.

**Wykorzystać istniejące kontenery do stworzenia własnej aplikacji**

A konkretnie, zidentyfikować i wykorzystać już stworzone obrazy

- z repozytoriów publicznych
- z własnego repozytorium.

**Zaimplementować mechanizmy wymiany plików w kanałach kontener<>host i kontener<>kontener**

Czyli dobrać i wdrożyć optymalną w danym przypadku strategię wymiany informacji pomiędzy kontenerami i hostem. W szczególności, wymieniać dane, parametry sterujące i pliki, wykorzystując metody docker volume i docker mount.

**Tworzyć aplikacje wielokontenerowe** Projektować, udostępniać i uruchamiać aplikacje wielokontenerowych jako:

1. Równoległe działające, niezależne usługi
2. Usługi wymieniające między sobą dane
3. Usługi uruchamiane sekwencyjnie.

**Zarządzać obrazami, kontenerami i woluminami** korzystając z:

1. Linii komend
2. Aplikacji Docker Desktop
3. Środowisk IDE, np. MS VSCode.

**Powodzenia!**

### 5.1.1 Scenariusze użycia docker w projektach Data Science

Konteneryzacja jest jedną z kluczowych metod wykorzystywanych w projektach Data Science.

Poniżej prezentuję najciekawsze moim zdaniem scenariusze jej użycia, porządkując je wg kluczowych etapów cyklu życia projektu: organizacji, wytwarzania, udostępniania i wdrożenia produkcyjnego.

#### Organizacja

Chcę szybko wprowadzić nowego pracownika do projektu

Do naszego zespołu dołącza nowy pracownik. Chcemy szybko przygotować mu maszynę (PC, serwer, notebook).

Zamiast instalować komplet pakietów: uruchamiamy kontener Docker.

### **Chcę kontynuować pracę nad moim projektem na innym komputerze**

Często zmieniam lokalizację: podróżuję, wracam do domu, etc. Chciałbym móc wykonywać obliczenia, trenować modele lub testować aplikację na komputerach, do których mam dostęp. Różnią się one nie tylko systemami operacyjnymi, ale też zainstalowanymi na nich środowiskami oraz konfiguracją sprzętową. W szczególności, część z nich nie może korzystać z akceleratorów takich jak karty graficzne.

Zamiast na każdym z nich instalować dedykowane środowiska, uruchamiam kontener docker z różnymi parametrami (np. z lub bez akceleracji GPU).

### **Wytwarzanie**

### **Chcę szybko sprawdzić działanie nowej biblioteki uczenia maszynowego**

Nie chcę jednak tworzyć dedykowanego środowiska wirtualnego, instalować odpowiednich pakietów, etc.

Pozyskuję oficjalny obraz aplikacji (zawierający odpowiednio skonfigurowane środowisko) i uruchamiam odpowiedni komponent.

### **Chcę stworzyć nową usługę uczenia maszynowego łącząc ze sobą gotowe komponenty**

Wykorzystuję <https://hub.docker.com/> do pozyskania odpowiednich obrazów (trochę jak "sklep" z aplikacjami). Łączę je ze sobą w potok korzystając z docker compose.

### **Chcę zademonstrować innym członkom zespołu aplikację wykorzystującą uczenie maszynowe**

...ale tak, by w razie potrzeby mogli sami eksperymentować z odpowiednimi komponentami. (np. zmieniać dane, algorytmy trenowania, aplikację końcową, etc.).

Udostępniam im obraz docker z odpowiednim środowiskiem oraz kompletem kodów źródłowych.

### **Udostępnianie**

### **Chcę udostępnić innym aplikację wykorzystującą bazę danych**

Moja aplikacja wykorzystuje bazę danych (np. Redis, PostgreSQL czy MySQL). Chcę ją udostępnić osobom, które nie mają jej zainstalowanej, nie potrafią też jej skonfigurować czy zarządzać.

Wykorzystuję docker compose do stworzenia aplikacji wielokontenerowej, zawierającej zarówno moją aplikację, jak i odpowiednią bazę danych.

**Chcę przeprowadzić szkolenie z zakresu Data Science**

Chcę pokazać komuś rozwiązanie w jupyter notebook, wykorzystujące różne biblioteki (np. PyCaret). Użytkownicy nie mają zainstalowanych żadnych środowisk.

Proszę ich o instalację Docker, a potem uruchomienie mojego kontenera.

**Archiwizując projekt, chcę zapewnić odtwarzalność mojego rozwiązania w przyszłości**

Odpowiednio dokumentuję i publikuję obraz docker mojego rozwiązania w repozytorium obrazów.

**Produkcja****Chcę przekazywać do kontenera dane specyficzne dla danego hosta (np. jako parametry uruchomieniowy)**

Wykorzystuję w tym celu docker bind mount.

**Chcę zapewnić ciągłość działania mojej aplikacji**

... tak, aby kolejne uruchomienia kontenera z moją aplikacją mogły korzystać z wyników poprzednich uruchomień

Wykorzystuję w tym celu docker volume.

**Chcę, aby różne równolegle pracujące kontenery wymieniały się danymi**

Wykorzystuję w tym celu docker volume.

**Chcę przechowywać dane wykorzystywane przez moje kontenery u dostawcy usług chmurowych**

Wykorzystuję w tym celu docker volume.

**Chcę zasymulować środowisko produkcyjne**

Nasz system analizy obrazu uruchamiany jest na urządzeniu końcowym typu NVidia Jetson Nano. Na swojej stacji roboczej chcę zasymulować, jak zachowa się na tym urządzeniu najnowsza wersja mojego modelu.

W tym celu uruchamiam kontener Docker z obrazem urządzenia końcowego.

**Chcę na urządzeniu końcowym uruchomić wiele serwisów lub sekwencję usług**

W tym celu wykorzystuję docker compose do stworzenia uruchomienia kilku usług: równolegle lub w odpowiedniej sekwencji.

**Chcę uruchomić mój projekt ML w skalowalnym środowisku, np. Kubernetes, czy u dostawcy usług (GCP, MS Azure czy Amazon EC2)**



Dla każdego z komponentów mojego procesu uczenia maszynowego tworzę osobny obraz. Wykorzystuję system KubeFlow do zarządzania pełnym procesem w systemie Kubernetes.

### 5.1.2 Niezbędne umiejętności

Poniżej przedstawiam najważniejsze umiejętności, które warto posiadać by w pełni wykorzystywać potencjał rozwiązań docker w projektach uczenia maszynowego.

#### Pojedyncza aplikacja

1. Zaprojektowanie obrazu z wykorzystaniem Dockerfile
2. Zbudowanie obrazu
3. Udostępnienie obrazu innym
4. Pobranie obrazu i uruchomienie kontenera.

#### Wykorzystanie istniejących kontenerów

Identyfikacja i wykorzystanie już stworzonych obrazów - z repozytoriów publicznych - z własnego repozytorium.

#### Wymiana plików kontener<>host i kontener<>kontener

Dobór i implementacja optymalnej w danym przypadku strategii wymiany informacji pomiędzy kontenerami i hostem

Informacje:

1. Dane
2. Parametry sterujące
3. Pliki.

Metody:

1. Docker volume
2. Docker bind-mount
3. Networking.

**Tworzenie aplikacji wielokontenerowych** Projektowanie, udostępnianie i uruchamianie aplikacji wielokontenerowych:

1. Równoległe działające, niezależne usługi
2. Usługi wymieniające między sobą dane
3. Usługi uruchamiane sekwencyjnie

#### Zarządzanie obrazami, kontenerami i woluminami

1. Linia komend
2. Aplikacja Docker Desktop
3. Środowiska IDE, np. MS VSCode.

### Przydatne źródła

Najlepszym znanym mi źródłem wiedzy nt systemu Docker jest oficjalny kurs dostępny na stronie:

<https://docs.docker.com/get-started/>

Dobre wprowadzenie w tematykę wykorzystania docker'a w projektach Data Science dostępne jest tutaj

## 5.2 Organizacja, pierwszy obraz i prosta aplikacja

### 5.2.1 Demo - pierwszy obraz docker

#### Organizacja

- ☐ Załóż konto na Docker
- ☐ Zaloguj się w Docker Hub, i stwórz pierwsze repozytorium
- ☐ Zainstaluj aplikację Docker

#### Obraz

- ☐ Załóż folder, i stwórz w nim prostą aplikację
- ☐ Stwórz plik Dockerfile, na wzór:
- ☐ Stwórz obraz docker nadając mu nazwę (flaga `-tag`) *pierwsza\_aplikacja*:  

```
$ docker build --tag pierwsza_aplikacja .
```
- ☐ Uruchom obraz o nazwie *pierwsza\_aplikacja* lokalnie:  

```
$ docker run pierwsza_aplikacja
```

#### Udostępnienie

- ☐ Zmień nazwę obrazu tak, by móc go opublikować swoim repozytorium Docker Hub:  

```
$ docker build -t andrzejwodecki/ml_tests:pierwsza_aplikacja .
```
- ☐ Opublikuj obraz w swoim repozytorium Docker Hub:  

```
$ docker push -t andrzejwodecki/ml_tests:pierwsza_aplikacja
```
- ☐ Uruchom obraz na innym systemie:
  - ☐ zainstaluj na nim Docker'a
  - ☐ uruchom kontener:  

```
$ docker run -t andrzejwodecki/ml_tests:pierwsza_aplikacja
```

Przydatne źródła

Najlepszym znanym mi źródłem wiedzy nt systemu Docker jest oficjalny kurs dostępny na stronie:

<https://docs.docker.com/get-started/>

### 5.2.2 Demo: trenowanie i ewaluacja

W tym prostym ćwiczeniu stworzysz i uruchomisz lokalnie prosty proces trenowania i ewaluacji modelu.

Aby stworzyć obraz, w linii komend wpisz:

```
docker build --tag train_eval .
```

Aby uruchomić kontener, w linii komend wpisz:

```
docker run -ti train_eval
```

Zauważ, że w tym podejściu:

1. kontener automatycznie uruchamia skrypt `run.py` (zgodnie z ostatnią instrukcją w pliku `Dockerfile`)
2. jest zarzynywany bezpośrednio po ukończeniu jego realizacji (możesz to sprawdzić uruchamiając w linii komend polecenie `docker ps`)
3. wszystkie pliki niezbędne do uruchomienia (komponenty i artefakty) są przechowywane w kontenerze
4. użytkownik nie ma do nich dostępu, w szczególności nie może ich modyfikować.

### 5.2.3 Ćwiczenie: prosta transformacja danych z pandas

*Uwaga: komplet wersji demonstracyjnych, ćwiczeń i rozwiązań oraz rekomendacje dotyczące środowiska uruchomieniowego znajdziesz tutaj:*

[https://github.com/wodecki/ASI\\_2022](https://github.com/wodecki/ASI_2022)

---

W tym ćwiczeniu stworzysz i uruchomisz prosty kontener docker umożliwiający wczytanie danych z istniejącego pliku csv, przemnożenie ich przez 2 i zapis tak zmienionej ramki danych do nowego pliku.

#### Lista kontrolna

- ☐ Skrypt Python `app1.py`:
  - ☐ Wczytuje pakiet pandas w poprawnej wersji i drukuje tę wersję na ekranie
  - ☐ Wczytuje zawartość pliku `/input/input.csv` do ramki danych, i drukuje ją na ekranie

- ☐ Mnoży tę treść x2, przypisuje do nowej ramki danych i drukuje wynik na ekranie
- ☐ Zapisuje nową ramkę do pliku tekstowego `/output/output.csv`
- ☐ Plik **Dockerfile** specyfikujący obraz, który kopiuje komplet niezbędnych danych, ale nie uruchamia skryptu `app1.py` (dzięki czemu kontener nie zatrzymuje się po jego uruchomieniu)
- ☐ Obraz docker o nazwie `app1`
- ☐ Plik **README.md** z instrukcją dla użytkownika pokazującą,
  - ☐ w jaki sposób zbudować obraz
  - ☐ w jaki sposób uruchomić kontener tak, by użytkownik mógł obejrzeć i modyfikować pliki wewnątrz kontenera (komponent `app1.py` i artefakt `input\input.csv`).
  - ☐ Wskazówka: aby umożliwić edycję pliku, po uruchomieniu kontenera trzeba w nim zainstalować wybrany edytor. Przykładowo, dla edytora `nano`, uruchomić:
    - ☐ `$ apt-get update`
    - ☐ `$ apt-get install nano`
- ☐ w jaki sposób uruchomić skrypt `app1.py` dostępny wewnątrz uruchomionego kontenera?

#### 5.2.4 Zarządzanie obrazami i kontenerami

Zrozumienie podstaw działania obrazów i kontenerów jest krytycznie ważne do ich efektywnego wykorzystania w projektach Data Science. Poniżej przedstawimy kilka kluczowych koncepcji, ilustrując je prostymi przykładami.

##### Zarządzanie obrazami

Najczęściej pierwszym krokiem w procesie tworzenia nowego obrazu jest zgromadzenie w jednym miejscu odpowiednich plików z danymi oraz skryptów oraz przetestowanie ich poprawnego działania.

W kolejnym kroku tworzymy plik **Dockerfile**, specyfikując zakres i konfigurację uruchamiania obrazów.

W naszym przykładzie zaczniemy od następującego pliku **Dockerfile**:

Obraz tworzymy komendą:

```
docker build -t app1:v1 .
```

Możemy teraz sprawdzić, czy nowy obraz pojawił się w systemie, korzystając z komendy (wykorzystujemy to przekierowanie potoku do funkcji `grep` po to, by pokazać wyłącznie obrazy o nazwie `app1`) :

```
docker images | grep app1
```

Jak widać, nowy obraz pojawił się w naszym systemie.

Obrazy możemy też usuwać (`docker images rm <ID kontenera>`), zmieniać ich nazwę, etc.

### Zarządzanie kontenerami

Komenda:

```
docker ps
```

wyświetla wszystkie **aktualnie uruchomione** kontenery.

Aby zobaczyć wszystkie kontenery w systemie, również te zatrzymane, trzeba uruchomić tę komendę z flagą `-a`:

```
docker ps -a
```

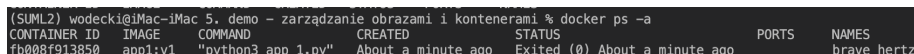
Uruchomienie tych dwóch komend w “czystym” systemie powinno zwrócić pusty wynik.

Zobaczmy, co się stanie, gdy uruchomimy nasz obraz wykorzystując komendę:

```
docker run -ti app1:v1
```

Program uruchamia się poprawnie:

Komenda `docker ps` zwraca pusty wynik, ale już `docker ps -a` wyświetla na ekranie:



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
fb008f913850	app1:v1	"python3 app_1.py"	About a minute ago	Exited (0) About a minute ago		brave_hertz

Figure 5.1: image-20220530185553968

Jak widać, **kontener po uruchomieniu się zatrzymał**: nie jest już aktywny (pusty komunikat `docker ps`), ale istnieje w repozytorium kontenerów.

Dzieje się tak dlatego, że w ostatniej linii naszego pliku Dockerfile wywołał-śmy komendę `CMD ["python3", "app_1.py"]` uruchamiającą skrypt Python: kontener po poprawnym wykonaniu skryptu otrzymuje sygnał do zatrzymania. Brak tej komendy:

1. Z jednej strony nie uruchamiałby skryptu `app_1.py`. Jego uruchomienie byłoby możliwe jedynie “z wnętrza” kontenera
2. ... ale z drugiej nie zatrzymuje działania kontenera.

Spróbujmy teraz uruchomić kontener ponownie, wykorzystując w tym celu jeszcze raz komendę:

```
docker run -ti app1:v1
```

Tak, jak poprzednio komenda `docker ps` zwraca pusty wynik, ale `docker ps -a` wyświetla na ekranie:

Wynika z tego, że **efektem ponownego uruchomienia kontenera komendą `docker run ...` było utworzenie nowego kontenera**.

Wynika z tego istotna obserwacja: **wielokrotne korzystanie z funkcji `docker run ...` generuje wiele kontenerów, osobny dla każdego z uruchomień**.

```
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS      PORTS      NAMES
902771bee841   appl1:v1  "python3 app_1.py"       37 seconds ago    Exited (0) 35 seconds ago
fb008f913850   appl1:v1  "python3 app_1.py"       4 minutes ago     Exited (0) 4 minutes ago
```

Figure 5.2: image-20220530185851387

Po pewnym czasie kontenerów może być tak dużo, że konieczne jest ich usunięcie. Można to zrobić “ręcznie” korzystając z komendy `docker container rm <ID kontenera>`:

```
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker container rm 902771bee841
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS      PORTS      NAMES
fb008f913850   appl1:v1  "python3 app_1.py"       9 minutes ago    Exited (0) 8 minutes ago
```

Figure 5.3: image-20220530190309976

W ekstremalnej wersji można też skorzystać z komendy `docker container prune` usuwającej wszystkie kontenery.

### Uruchamianie zatrzymanych kontenerów

Aby uruchomić ponownie zatrzymany kontener, uruchom:

```
docker container start <ID kontenera>:
```

Wykorzystaliśmy tu flagę `-a` (attach), aby móc wyświetlić komunikaty z kontenera.

Tak, jak poprzednio, po uruchomieniu jest on zatrzymywany, **a wszystkie generowane przez niego dane tracone**. O tym, jak je zachować, opowiemy w części poświęconej wymianie plików pomiędzy kontenerami a hostem.

### Uruchamianie komendy wewnątrz uruchomionego kontenera

Do uruchomienia komendy wewnątrz kontenera można wykorzystać komendę `EXEC`:

```
docker exec -ti <ID kontenera> COMMAND
```

Uwaga: przykład poniżej pokazuje, że **jest to możliwe wyłącznie w sytuacji, gdy kontener jest uruchomiony**:

```
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS      PORTS      NAMES
8b732fc483b6   appl1:v1  "python3 app_1.py"       2 minutes ago    Exited (0) 2 minutes ago
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker exec -ti 8b732fc483b6 sh ls
Error response from daemon: Container 8b732fc483b655640a0b51e31c60e1b5edf0f368acc74f849b2a2be832ffdc8b is not running
```

Figure 5.4: image-20220530192639210

Jak spowodować, aby nasz kontener nie był zamykany bezpośrednio po uruchomieniu?

Pierwszą z opcji przedstawiliśmy powyżej: wystarczy, aby nasz **Dockerfile** nie zawierał na końcu komendy uruchamiającej skrypt, jak na przykładzie poniżej:

Inna możliwość to **uruchomienie kontenera z komendą bash na końcu linii komendy**:

```
docker run -ti app1:v1 bash
```

Dzięki niej uruchamiamy kontener i otrzymujemy dostęp do jej powłoki bash. Będąc w niej, możemy uruchomić nasz skrypt, przeglądać i modyfikować pliki, etc.:

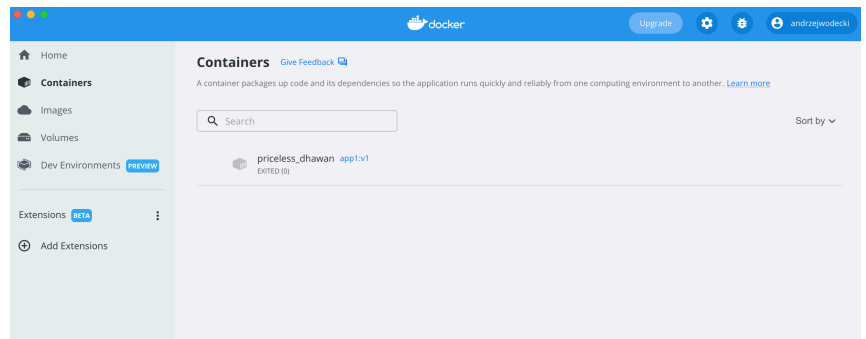
Tym razem już komenda `docker ps` wskazuje, że nasz kontener jest aktywny:

W efekcie, można już w nim uruchamiać różne programy wykorzystując komendę `exec`:

Jest to o tyle wygodne, że **po każdym takim uruchomieniu wracam do naszej lokalnej powłoki**. Może mieć to bardzo ciekawe zastosowania w uruchomieniu produkcyjnej (zarządzanie uruchamianiem kontenerów poprzez skrypty powłoki).

### Zarządzanie obrazami i kontenerami z wykorzystaniem aplikacji Docker Desktop oraz IDE

Linia komend to nie jedyny sposób na inspekcję i zarządzanie obrazami i kontenerami. Można do tego z powodzeniem wykorzystać:



1. Aplikację Docker Desktop;
2. Środowisko IDE, np. MS Visual Studio Code;

## 5.3 Wykorzystanie gotowych kontenerów

### 5.3.1 Uruchomienie gotowego kontenera z notatnikiem Jupyter i środowiskiem DataScience

#### Instrukcje

1. W linii komend uruchom:

```
$ docker run -p 10000:8888 jupyter/scipy-notebook:6b49f3337709
```

2. W przeglądarce wejdź na stronę:

```
http://127.0.0.1:10000/
```

3. W interfejsie Jupyter Lab, kliknij + (lewy górny róg):

aby dodać z własnego dysku pliki `diamonds.ipynb` i `diamonds.csv`

4. Uruchamiaj poszczególne komendy notatnika. Jak widać, komplet niezbędnych bibliotek jest już dostępny.

### Przydatne źródła

Opis publikowania portów z kontenera do hosta lokalnego (flaga `-p` (lub `--publish`) w poleceniu `docker run` powyżej) znajdziesz tutaj.

### 5.3.2 Demo: kontener ze środowiskiem PyCaret

*Uwaga: komplet wersji demonstracyjnych, ćwiczeń i rozwiązań oraz rekomendacje dotyczące środowiska uruchomieniowego znajdziesz tutaj:*

```
https://github.com/wodecki/ASI_2022
```

#### Cel

Naszym celem jest uruchomienie kontenera docker umożliwiającego realizację prostego projektu uczenia maszynowego z wykorzystaniem pakietu PyCaret. Instalacja pakietu na lokalnym systemie wymaga dużej uważności - oficjalny obraz Docker zdecydowanie ułatwia realizację tego zadania.

W tym celu wykorzystamy:

- Oficjalny, “lekki” obraz docker biblioteki Pycaret dostępny tutaj
- Notatnik z przykładowym projektem `Klasyfikacja binarna - CHURN.ipynb`
- Dane treningowe `customers_churn.csv`

#### Lista kontrolna

- ☐ Uruchomiony notatnik Jupyter umożliwiający wczytanie własnego notatnika i aktywację jądra `pycaret`
- ☐ Wczytane pliki:
  - ☐ `Klasyfikacja binarna - CHURN.ipynb`
  - ☐ `customers_churn.csv`
- ☐ Zrealizowane z sukcesem komendy z notatnika `Klasyfikacja binarna - CHURN.ipynb`

#### Rozwiązanie

1. Uruchom oficjalny, “lekki” obraz docker biblioteki Pycaret:

```
docker run -p 8888:8888 pycaret/slim
```



2. Wejdź na wskazany na ekranie adres serwera jupyter (127.0.0.1:8888...)
3. Zimportuj pliki *Klasyfikacja binarna* - `CHURN.ipynb` i `customers_churn.csv` (przycisk *Upload* w prawym górnym rogu)
4. Uruchom notatnik *Klasyfikacja binarna* - `CHURN.ipynb` i uruchom poszczególne komórki.

### Wskazówki

W sytuacji, gdy uruchomiony Jupyter Notebook wymagać będzie od Ciebie podania hasła (lub tokena), zaś ten wskazany przez Docker nie zadziała, problemem jest najpewniej uruchomiony wcześniej inny serwer Jupyter. Rozwiązanie:

1. sprawdź aktualnie uruchomione serwery Jupyter:  

```
jupyter notebook list
```
2. Zatrzymaj poprzednio uruchomiony serwer: 

```
jupyter notebook stop 8888
```
3. Uruchom ponownie obraz.

Niekiedy źródłem problemów mogą być też uruchomione wcześniej, i nadal aktywne kontenery. Aby sprawdzić, które z nich działają, uruchom: `docker run ps`

Aby usunąć wszystkie uruchomione kontenery (uwaga: to ekstremalne rozwiązanie), uruchom:

```
docker container prune
```

### 5.3.3 Ćwiczenie: prosta transformacja danych z pandas

*Uwaga: komplet wersji demonstracyjnych, ćwiczeń i rozwiązań oraz rekomendacje dotyczące środowiska uruchomieniowego znajdziesz tutaj:*

[https://github.com/wodecki/ASI\\_2022](https://github.com/wodecki/ASI_2022)

---

W tym ćwiczeniu stworzysz obraz i oraz instrukcje uruchomienia prostego kontenera docker umożliwiającego uruchomienie notatnika jupyter i wytrenowania prostego modelu regresji

#### Lista kontrolna

- ☐ Plik `Dockerfile` specyfikujący obraz, który:
  - ☐ Instaluje pakiety niezbędne do uruchomienia notatnika 1. **tworzenie modelu regresji.ipynb** Wskazówka: wcześniej zidentyfikuj listę niezbędnych pakietów. Nie zapomnij o pakiecie `jupyter`
  - ☐ Kopiuje zawartość plików z dysku lokalnego (np. 1. **tworzenie modelu regresji.ipynb**, `Boston.csv`, etc.) do katalogu roboczego `\app`

- ☐ Uruchamia jupyter notebook.
- ☐ Plik README.md z instrukcją dla użytkownika pokazującą,
  - ☐ w jaki sposób zbudować obraz
  - ☐ w jaki sposób uruchomić kontener.

## 5.4 Wymiana plików kontener<>host

Jak już wspomnieliśmy we wprowadzeniu, typowy proces uczenia maszynowego przebiega w wielu etapach.

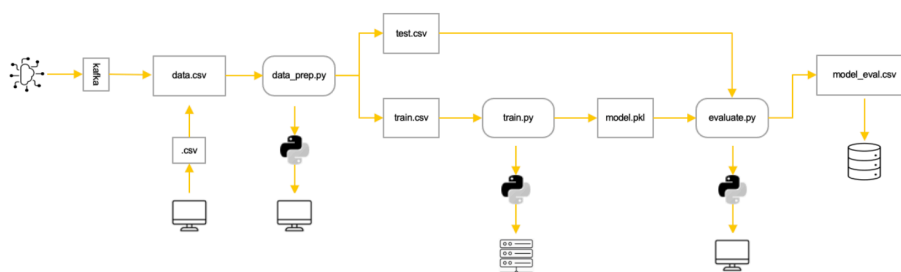


Figure 5.5: image-20220604102745700

Realizowane są one często w różnych środowiskach uruchomieniowych, na różnych maszynach wykorzystujących różne systemy.

Koordinacja takiego rozwiązania wymaga nie tylko zarządzania procesami czy infrastrukturą, ale też odpowiedniej wymiany danych, np. w formie plików.

W kolejnych lekcjach poznasz oferowane przez docker mechanizmy wymiany plików umożliwiające implementację następujących scenariuszy:

1. **Kontener udostępnia wyłącznie środowisko uruchomieniowe** (np. biblioteki, pakiety, etc.) - komplet danych jest na systemie lokalnym
2. **Kontener zawiera zarówno środowisko, jak i komponenty oraz artefakty niezbędne do uruchomienia projektu.** System lokalny ma zapewnić jedynie moc obliczeniową.
3. **Docker zapewnia synchronizację pomiędzy systemem lokalnym a systemem plików kontenera.** W efekcie, wszelkie zmiany w systemie lokalnym (a dokładniej: w konkretnej lokalizacji w tym systemie) są uwzględniane w kontenerze, co w szczególności umożliwia nanoszenie zmian w artefaktach/komponentach bez konieczności przebudowania obrazu.
4. **Docker zapewnia wymianę danych pomiędzy kontenerami a wirtualnym woluminem.** Nie jest konieczna synchronizacja z konkretną lokalizacją w systemie lokalnym, można to zastosować

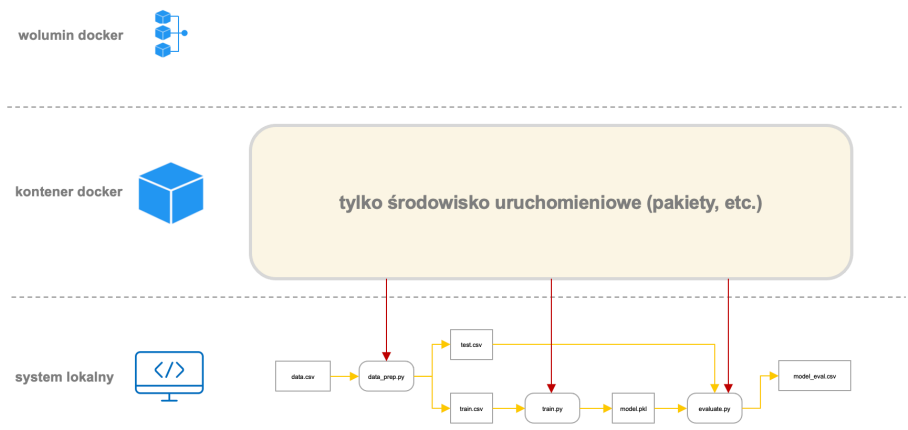


Figure 5.6: image-20220604103009850

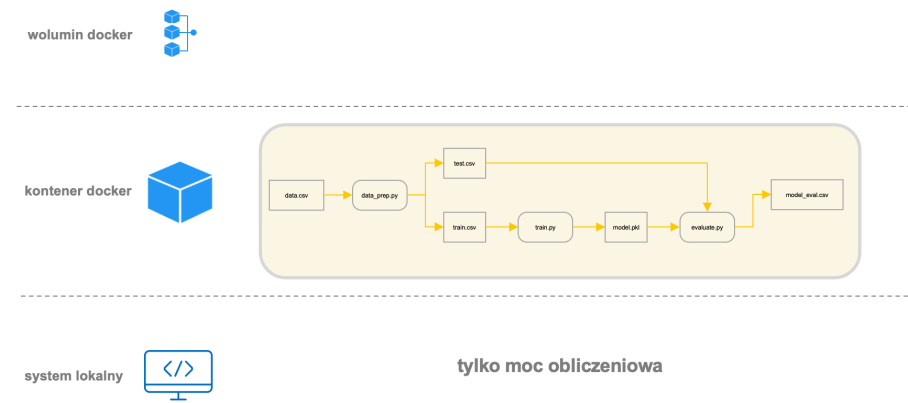


Figure 5.7: image-20220604103030339

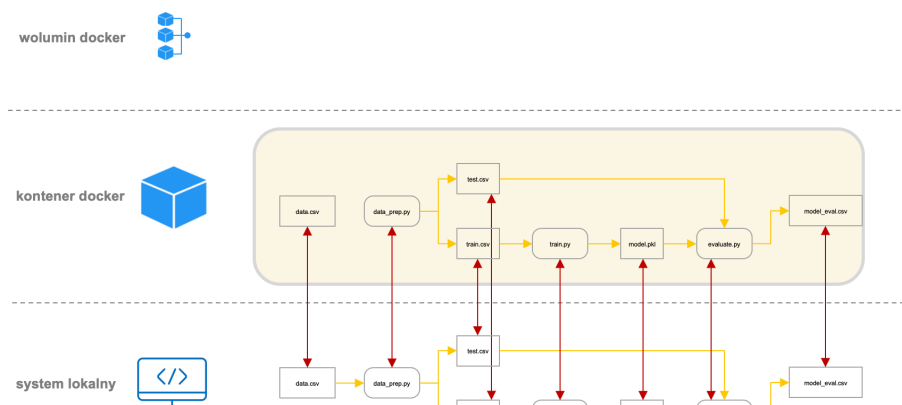


Figure 5.8: image-20220604103043609

zarówno do utrzymania ciągłości pracy jednego kontenera, jak i wymiany danych pomiędzy wieloma kontenerami (pracującymi równolegle lub uruchamianymi sekwencyjnie).

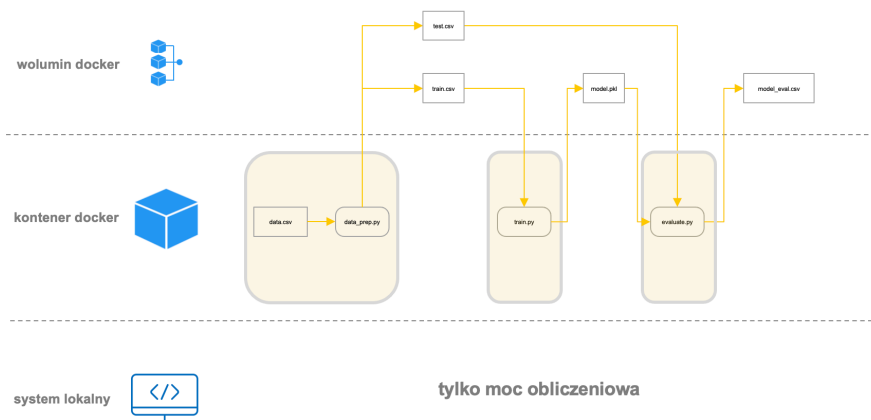


Figure 5.9: image-20220604103054834

W kolejnych przykładach zobaczysz implementację wszystkich tych scenariuszy. Zapraszam!

#### 5.4.1 Stworzenie kontenera udostępniającego wyłącznie środowisko uruchomieniowe

Najprostszym scenariuszem użycia kontenera Docker jest udostępnienie w nim wyłącznie środowiska uruchomieniowego. Komplet plików niezbędnych do uru-

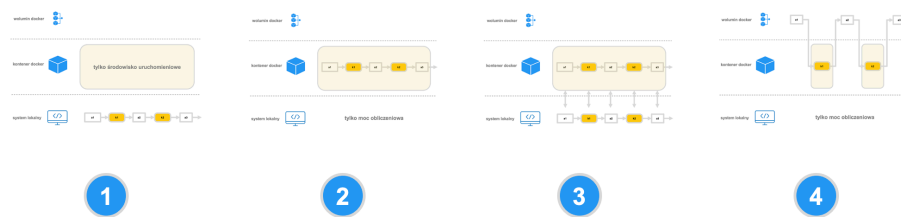


Figure 5.10: image-20220604103109484

chomienia aplikacji powinien znajdować się w systemie lokalnych (host'a).

### Budowa obrazu

Stwórz obraz:

```
docker build -t bind_app1_app2:v1 .
```

Uruchom obraz bez podłączania lokalnego wolumenu:

```
docker run -ti bind_app1_app2:v1
```

Zauważ, że:

1. po uruchomieniu kontenera automatycznie uruchamia się środowisko python
2. w środowisku tym zainstalowany jest pakiet pandas: komenda `import pandas as pd` nie generuje błędu.

### Kopiowanie plików host-kontener z wykorzystaniem linii komend

Aby móc wykorzystać tak skonfigurowane środowisko w praktyce, należy przekazać do niego pliki. W tym celu możesz wykorzystać komendę `docker cp`.

`docker cp ...` umożliwia kopiowanie plików pomiędzy uruchomionym kontenerem a systemem lokalnym. Przyjmuje ona tylko jeden argument po stronie "źródła", dlatego też dobrą praktyką jest zgromadzenie wszystkich plików i folderów w jednym katalogu (w naszym przypadku `my_app`), a potem skopiowanie go w całości do kontenera. Drugim parametrem jest miejsce docelowe: ID kontenera oraz jego folder docelowy.

Kompletna procedura z wykorzystaniem linii komend wygląda następująco:

1. W nowym terminalu uruchom kontener: `docker run -ti bind_app1_app2:v1 bash`
2. Sprawdź jego ID: `docker ps`
3. Przejdź do folderu lokalnego, w którym znajduje się folder, który chcesz skopiować. U mnie: katalog, w którym znajduje się folder `my_app/`

4. Uruchom w tym folderze kolejny terminal i wpisz w nim komendę: `docker cp my_app 111ac631710c:/app` UWAGA: zamień *111ac631710c* na ID własnego kontenera
5. W terminalu z uruchomionym kontenerem:
  1. Sprawdź komendą `ls`, czy Twój folder poprawnie się skopiował
  2. Wejdź do niego i uruchom skrypt: `python3 run.py`

### **Kopiowanie plików host-kontener z wykorzystaniem wtyczki Docker w IDE (np. MS Visual Studio Code)**

W praktyce często wygodniejszym rozwiązaniem jest wykorzystanie środowiska IDE do operacji na plikach w aktywnych kontenerach. W szczególności, MS Visual Studio Code udostępnia możliwości inspekcji i zarządzania obrazami, kontenerami i wolumenami (wtyczka Docker).

### **Scenariusze użycia**

Wyżej wymieniony mechanizm możesz w szczególności zastosować w następujących sytuacjach:

#### **Chcę szybko wprowadzić nowego pracownika do projektu**

Do naszego zespołu dołącza nowy pracownik. Chcemy szybko przygotować mu maszynę (PC, serwer, notebook).

Zamiast instalować komplet pakietów: uruchamiamy kontener Docker.

#### **Chcę kontynuować pracę nad moim projektem na innym komputerze**

Często zmieniam lokalizację: podróżuję, wracam do domu, etc. Chciałbym móc wykonywać obliczenia, trenować modele lub testować aplikację na komputerach, do których mam dostęp. Różnią się one nie tylko systemami operacyjnymi, ale też zainstalowanymi na nich środowiskami oraz konfiguracją sprzętową. W szczególności, część z nich nie może korzystać z akceleratorów takich jak karty graficzne.

Zamiast na każdym z nich instalować dedykowane środowiska, uruchamiam kontener docker z różnymi parametrami (np. z lub bez akceleracji GPU).

#### **Chcę szybko sprawdzić działanie nowej biblioteki uczenia maszynowego**

Nie chcę jednak tworzyć dedykowanego środowiska wirtualnego, instalować odpowiednich pakietów, etc.

Pozyskuję oficjalny obraz aplikacji (zawierający odpowiednio skonfigurowane środowisko) i uruchamiam odpowiedni komponent.

#### **Chcę stworzyć nową usługę uczenia maszynowego łącząc ze sobą gotowe komponenty**

Wykorzystuję <https://hub.docker.com/> do pozyskania odpowiednich obrazów (trochę jak “sklep” z aplikacjami). Łączę je ze sobą w potok korzystając z docker compose.

**Chcę zasymulować środowisko produkcyjne**

Nasz system analizy obrazu uruchamiany jest na urządzeniu końcowym typu NVidia Jetson Nano. Na swojej stacji roboczej chcę zasymulować, jak zachowa się na tym urządzeniu najnowsza wersja mojego modelu.

W tym celu uruchamiam kontener Docker z obrazem urządzenia końcowego.

**Chcę uruchomić mój projekt ML w skalowalnym środowisku**, np. Kubernetes, czy u dostawcy usług (GCP, MS Azure czy Amazon EC2)

Dla każdego z komponentów mojego procesu uczenia maszynowego tworzę osobny obraz. Wykorzystuję system KubeFlow do zarządzania pełnym procesem w systemie Kubernetes.

**5.4.2 Uruchomienie bez podłączania lokalnego systemu plików**

Jeśli chcemy przekazać komuś kontener z kompletem niezbędnych plików (tak, by mógł uruchomić kompletny proces bez konfiguracji zarówno środowiska, jak i posiadania niezbędnych plików), możemy w pliku *Dockerfile* skopiować komplet plików do odpowiedniego katalogu w obrazie. Odpowiada za to komenda `COPY` . . . :

Aby stworzyć obraz:

1. przejdź do folderu, w którym masz komplet niezbędnych danych
2. uruchom w nim terminal, a w nim komendę: `docker build -t bind_app1_app2:v2 .`

Tak stworzony obraz możesz uruchomić w dowolnym katalogu - nie musisz mieć w nim dostępu do niezbędnych plików:

```
docker run -ti bind_app1_app2:v2 python3 run.py
```

Zauważ, że:

1. na końcu komendy dodaliśmy `python3 run.py`. Jest to związane z tym, że zgodnie z plikiem *Dockerfile* nasz obraz nie uruchamia skryptu `run.py`
2. uruchomienie w folderze, w którym nie ma niezbędnych plików, nie generuje błędów
3. kolejne uruchomienia nie dodają żadnych plików do lokalnego systemu.

**Scenariusze użycia**

Wyżej wymieniony mechanizm możesz w szczególności zastosować w następujących sytuacjach:

**Chcę udostępnić innym aplikację wykorzystującą bazę danych**

Moja aplikacja wykorzystuje bazę danych (np. Redis, PostgreSQL czy MySQL). Chcę ją udostępnić osobom, które nie mają jej zainstalowanej, nie potrafią też jej skonfigurować czy zarządzać.

Wykorzystuję docker compose do stworzenia aplikacji wielokontenerowej, zawierającej zarówno moją aplikację, jak i odpowiednią bazę danych.

#### **Chcę przeprowadzić szkolenie z zakresu Data Science**

Chcę pokazać komuś rozwiązanie w jupyter notebook, wykorzystujące różne biblioteki (np. PyCaret). Użytkownicy nie mają zainstalowanych żadnych środowisk.

Proszę ich o instalację Docker, a potem uruchomienie mojego kontenera.

#### **Archiwizując projekt, chcę zapewnić odtwarzalność mojego rozwiązania w przyszłości**

Odpowiednio dokumentuję i publikuję obraz docker mojego rozwiązania w repozytorium obrazów.

#### **Chcę zasymulować środowisko produkcyjne**

Nasz system analizy obrazu uruchamiany jest na urządzeniu końcowym typu NVidia Jetson Nano. Na swojej stacji roboczej chcę zasymulować, jak zachowa się na tym urządzeniu najnowsza wersja mojego modelu.

W tym celu uruchamiam kontener Docker z obrazem urządzenia końcowego.

#### **Chcę na urządzeniu końcowym uruchomić wiele serwisów lub sekwencję usług**

W tym celu wykorzystuję docker compose do stworzenia uruchomienia kilku usług: równoległe lub w odpowiedniej sekwencji.

#### **Chcę uruchomić mój projekt ML w skalowalnym środowisku, np. Kubernetes, czy u dostawcy usług (GCP, MS Azure czy Amazon EC2)**

Dla każdego z komponentów mojego procesu uczenia maszynowego tworzę osobny obraz. Wykorzystuję system KubeFlow do zarządzania pełnym procesem w systemie Kubernetes.

#### **Przydatne źródła**

Bardzo dobrą prezentację metod *bind mount* i *volume mount* znajdziesz w oficjalnej dokumentacji docker dostępnej tutaj.

### **5.4.3 Uruchomienie z podłączeniem lokalnego systemu plików**

Bardzo często istnieje potrzeba synchronizacji lokalnego systemu plików z kontenerem tak, by można było zmieniać dane źródłowe bez konieczności przebudowy obrazu. Przykładowo, chcemy mieć możliwość modyfikacji plików z danymi, zmiany skryptów python, czy też podglądu pod wyniki uruchomień kontenerów w lokalnym systemie.



Powyższe scenariusze mogą być zrealizowane z wykorzystaniem mechanizmu **bind mount**.

#### Synchronizacja folderu host'a z kontenerem

Aby stworzyć obraz, w folderze z plikiem Dockerfile uruchom w terminalu komendę:

```
docker build -t bind_app1_app2:v3 .
```

Po zbudowaniu obrazu, uruchom odpowiedni kontener podłączając lokalny wolumin:

```
docker run -ti -v"${pwd}:/app" bind_app1_app2:v3 python3 run.py
```

Zauważ, że:

1. tym razem pliki wygenerowane w ramach kontenera synchronizują się z Twoim folderem lokalnym. W szczególności, w folderach `app_1/input` i `app_1/output` pojawiły się nowe pliki
2. konieczne jest jednak uruchamianie kontenera w folderze, który zawiera niezbędne pliki. Opcja `-v"${pwd}:/app"` usuwa pliki z katalogu `\app` kontenera
3. jeśli chcesz uruchomić kontener w miejscu, w którym nie ma odpowiednich plików, uruchom go bez opcji `bind mount`: `docker run -ti bind_app1_app2:v3 python3 run.py`

#### Modyfikacja pliku w folderze lokalnym

Sprawdźmy teraz, czy ta synchronizacja działa w obie strony. W tym celu zmodyfikuj skrypt `run.py` dodając do niego wydruk `część stary...`

Jak widać, modyfikacja pliku lokalnego jest uwzględniana przez kontener bez konieczności przebudowania obrazu.

#### Scenariusze użycia

Wyżej wymieniony mechanizm możesz w szczególności zastosować w następujących sytuacjach:

##### Chcę kontynuować pracę nad moim projektem na innym komputerze

Często zmieniam lokalizację: podróżuję, wracam do domu, etc. Chciałbym móc wykonywać obliczenia, trenować modele lub testować aplikację na komputerach, do których mam dostęp. Różnią się one nie tylko systemami operacyjnymi, ale też zainstalowanymi na nich środowiskami oraz konfiguracją sprzętową. W szczególności, część z nich nie może korzystać z akceleratorów takich jak karty graficzne.

Zamiast na każdym z nich instalować dedykowane środowiska, uruchamiam kontener docker z różnymi parametrami (np. z lub bez akceleracji GPU).

##### Chcę zademonstrować innym członkom zespołu aplikację wykorzystującą uczenie maszynowe

...ale tak, by w razie potrzeby mogli sami eksperymentować z odpowiednimi komponentami. (np. zmieniać dane, algorytmy trenowania, aplikację końcową, etc.).

Udostępniam im obraz docker z odpowiednim środowiskiem oraz kompletem kodów źródłowych.

### Chcę przeprowadzić szkolenie z zakresu Data Science

Chcę pokazać komuś rozwiązanie w jupyter notebook, wykorzystujące różne biblioteki (np. PyCaret). Użytkownicy nie mają zainstalowanych żadnych środowisk.

Proszę ich o instalację Docker, a potem uruchomienie mojego kontenera.

### Chcę przekazywać do kontenera dane specyficzne dla danego hosta (np. jako parametry uruchomieniowy)

Wykorzystuję w tym celu docker bind mount.

### Chcę zasymulować środowisko produkcyjne

Nasz system analizy obrazu uruchamiany jest na urządzeniu końcowym typu NVidia Jetson Nano. Na swojej stacji roboczej chcę zasymulować, jak zachowa się na tym urządzeniu najnowsza wersja mojego modelu.

W tym celu uruchamiam kontener Docker z obrazem urządzenia końcowego.

### Przydatne źródła

Bardzo dobrą prezentację metod *bind mount* i *volume mount* znajdziesz w oficjalnej dokumentacji docker dostępnej tutaj.

#### 5.4.4 Utrzymanie ciągłości pracy pojedynczej aplikacji z volume mount

Często istnieje potrzeba współdzielenia danych pomiędzy różnymi kontenerami. Można wyróżnić trzy typowe sytuacje, w których może ona zaistnieć:

1. Wielokrotnie uruchamiamy ten sam kontener, i chcemy zapewnić ciągłość jego działania. Przykładowo, uruchomienie nr 2 korzysta z rezultatów uruchomienia nr 1 (dane z uruchomienia 1 nie znikają, ale są przekazywane na potrzeby kolejnego)
2. Mamy kilka działających równolegle kontenerów i chcemy, by wymieniały one pomiędzy sobą pliki.
3. Mamy kilka kontenerów uruchamianych w sekwencji i chcemy, by przekazywały one sobie wyniki operacji.

W poniższym przykładzie pokażemy, jak można zapewnić ciągłość działania pojedynczej aplikacji korzystając z mechanizmu **volume mount**.

### Nasza aplikacja

W naszym przykładzie wykorzystamy aplikację *app\_1.py* realizującą prostą transformację ramki danych:

Zwróć uwagę na eksporty do pliku csv na końcu aplikacji: wynik obliczeń przekazujemy nie tylko do pliku *data/output\_1.csv*, ale też pliku wejściowego *data/input\_1.csv*. W efekcie, kolejne uruchomienia, przy poprawnie działającym mechanizmie zachowania ciągłości danych, powinny korzystać z wyników uruchomień poprzednich.

### Stworzenie obrazu

Obraz tworzymy w oparciu o plik Dockerfile:

Korzystamy w tym celu z komendy:

```
docker build -t app1_vm:v1 .
```

### Stworzenie współdzielonego woluminu

Funkcjonalność *volume mount* umożliwia stworzenie lokalnego, ale zarządzanego przez docker'a współdzielonego woluminu. Stwórzmy taki wolumen korzystając z komendy:

```
docker volume create vm_app1
```

Możesz teraz sprawdzić, np. korzystając z aplikacji Docker Desktop, że nie zawiera on żadnych danych.

### Sekwencyjne uruchamianie kontenerów z wykorzystaniem mechanizmu volume mount

Aby uruchomić kontener z opcją synchronizacja danych z woluminem Docker, w linii komend wpisz:

```
docker run -ti -v vm_app1_vm:/app/data app1_vm:v1
```

Efekt to nie tylko uruchomiony skrypt:

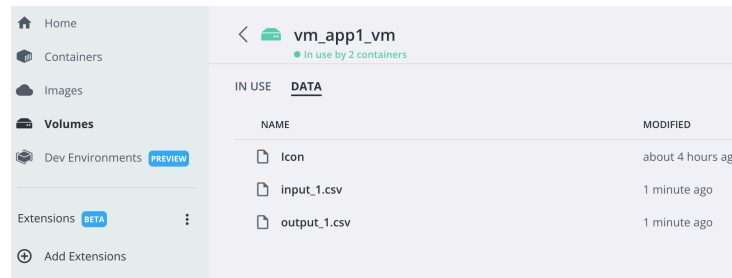
```
((base) wodecki@iMac-iMac 3.4. Pojedyncza aplikacja z volume mount % docker run -ti -v vm_app1_vm:/app/data app1_vm:v1
... App 1 Started ...

Original df:
  0  1  2
0  1 32 10
1  3  4 315

Random number: 2

Transformed df:
  0  1  2
0  2 64 20
1  6  8 630

... App 1 Completed ...
```



... ale też nowe dane w naszym wolumenie:

Kolejne uruchomienia kontenerów wykorzystują wyniki poprzednich:

```
((base) wodecki@iMac-iMac 3.4. Pojedyncza aplikacja z volume mount % docker run -ti -v vm_app1_vm:/app/data app1_vm:v1
... App 1 Started ...

Original df:
  0  1  2
0  1 32 10
1  3  4 315

Random number: 2

Transformed df:
  0  1  2
0  2 64 20
1  6  8 630

... App 1 Completed ...

((base) wodecki@iMac-iMac 3.4. Pojedyncza aplikacja z volume mount % docker run -ti -v vm_app1_vm:/app/data app1_vm:v1
... App 1 Started ...

Original df:
  0  1  2
0  2 64 20
1  6  8 630

Random number: 4

Transformed df:
  0  1  2
0  8 256 80
1 24 32 2520

... App 1 Completed ...

((base) wodecki@iMac-iMac 3.4. Pojedyncza aplikacja z volume mount % docker run -ti -v vm_app1_vm:/app/data app1_vm:v1
... App 1 Started ...

Original df:
  0  1  2
0  8 256 80
1 24 32 2520

Random number: 5

Transformed df:
  0  1  2
0 40 1280 400
1 120 160 12600

... App 1 Completed ...
```

Scenariusze użycia

Wyżej wymieniony mechanizm możesz w szczególności zastosować w następujących sytuacjach:

Chcę zapewnić ciągłość działania mojej aplikacji

... tak, aby kolejne uruchomienia kontenera z moją aplikacją mogły korzystać z wyników poprzednich uruchomień

Wykorzystuję w tym celu docker volume.

**Chcę, aby różne równolegle pracujące kontenery wymieniały wymieniały się danymi**

Wykorzystuję w tym celu docker volume.

**Chcę przechowywać dane wykorzystywane przez moje kontenery u dostawcy usług chmurowych**

Wykorzystuję w tym celu docker volume.

#### 5.4.5 Współdzielenie danych pomiędzy kontenerami z volume mount

W tej części naszego wykładu pokażemy, w jaki sposób można zapewnić wymianę danych pomiędzy **wieloma kontenerami** z wykorzystaniem mechanizmu **volume mount**.

##### Stworzenie obrazów i uruchomienie poszczególnych kontenerów

W naszym przykładzie stworzymy i uruchomimy 2 kontenery: `app1_shared` i `app2_shared`.

Obie dokonują prostej transformacji danych zaczytanych z pliku zewnętrznego, przy czym `app2_shared` jako wejście traktuje wynik działania `app1_shared`:

Aby stworzyć obraz `app1`, w folderze `\app_1` uruchom w terminalu komendę:

```
docker build -t app1_shared:v1 .
```

Aby go uruchomić:

```
docker run -ti app1_shared:v1
```

Analogicznie, aby stworzyć obraz `app2`, w folderze `\app_2` uruchom w terminalu komendę:

```
docker build -t app2_shared:v1 .
```

Aby go uruchomić:

```
docker run -ti app2_shared:v1
```

##### Stworzenie współdzielonego woluminu

Funkcjonalność *volume mount* umożliwia stworzenie lokalnego, ale zarządzanego przez docker'a współdzielonego woluminu. Stwórzmy taki wolumen korzystając z komendy:

```
docker volume create vol_app1_app2
```

**Uruchomienie poszczególnych kontenerów z funkcjonalnością wymiany danych**

Aby uruchomić poszczególne aplikacje w trybie wymiany danych z wykorzystaniem mechanizmu volume mount:

1. W przypadku aplikacji `app1_shared` w linii komend uruchom:

```
docker run -ti -v vol_app1_app2:/app/data app1_shared:v1
```

2. W przypadku aplikacji `app2_shared` w linii komend uruchom:

```
docker run -ti -v vol_app1_app2:/app/data app2_shared:v1
```

Zauważ, że sekwencyjne uruchamianie tych aplikacji powoduje uwzględnianie wyników poprzednich obliczeń jako weściowych do kolejnych:

### Scenariusze użycia

Wyżej wymieniony mechanizm możesz w szczególności zastosować w następujących sytuacjach:

#### Chcę zapewnić ciągłość działania mojej aplikacji

... tak, aby kolejne uruchomienia kontenera z moją aplikacją mogły korzystać z wyników poprzednich uruchomień

Wykorzystuję w tym celu docker volume.

#### Chcę, aby różne równolegle pracujące kontenery wymieniały się danymi

Wykorzystuję w tym celu docker volume.

#### Chcę przechowywać dane wykorzystywane przez moje kontenery u dostawcy usług chmurowych

Wykorzystuję w tym celu docker volume.

### Przydatne źródła

Bardzo dobrą prezentację metod *bind mount* i *volume mount* znajdziesz w oficjalnej dokumentacji docker dostępnej tutaj.

## 5.5 Aplikacje wielokontenerowe

Docker umożliwia tworzenie aplikacji wielokontenerowych współdzielących ze sobą dane.

Mogą być one uruchamiane niezależnie bądź sekwencyjnie.

Do projektowania i uruchamiania takich aplikacji służy moduł **docker compose**, zaś specyfikacja konfiguracji przechowywana jest w pliku `docker-compose.yaml`.

W naszym przypadku plik ten ma następujący kształt:

Zwróć uwagę na:

1. Fakt, że skrypty aplikacji `app1` i `app2` korzystają z inaczej nazwanych katalogów, odpowiednio `/data_1` i `/data_2`
2. Na początku pliku `docker-compose.yaml` zdefiniowaliśmy współdzielony wolumin `shared_data` (o tej i innych mechanizmu wymiany plików pomiędzy kontenerami możesz przeczytać np. tutaj).
3. W specyfikacji usług `app1` i `app2` wskazaliśmy mechanizm mapowania woluminu docker z woluminami poszczególnych kontenerów.

### Budowa i uruchomienie

Aby stworzyć i uruchomić taką aplikację, w linii komend uruchom:

```
docker compose up
```

Aby uruchomić jedną z wybranych aplikacji:

```
docker compose run app1 < uruchamia app1
```

```
docker compose run app2 < uruchamia app2
```

W efekcie, wyniki sekwencyjnego uruchamiania usług pokazują, że dane propagowane są prawidłowo:

Aby uruchomić w trybie zwalnającym terminal (detached):

```
docker compose up -d
```

W takim przypadku, aby zatrzymać aplikacje:

```
docker compose stop
```

```
docker compose down --volumes
```

 < usuń też wszystkie współdzielone woluminy.

Aby sprawdzić listę aktywnych procesów docker compose:

```
docker compose ps
```

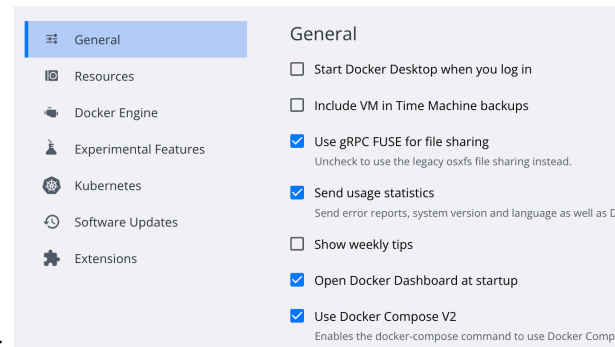
Jeśli w międzyczasie zmodyfikowała/eś pliki źródłowe (np. skrypty Python), przebuduj odpowiednie obrazy:

```
docker compose build
```

**docker compose vs docker-compose**

**UWAGA:** w najnowszych wersjach Docker'a zaimplementowane moduł `compose` bezpośrednio w pakiecie. W efekcie, można go uruchamiać w formie `docker compose ....`

W starszych wersjach korzystanie z tej usługi możliwe jest po zainstalowaniu osobnej biblioteki `docker-compose`, zaś uruchomienie wymaga wpisania `docker-compose ...` (< zwróć uwagę na znak - w środku).



Możesz to też skonfigurować w ustawieniach Docker:

### Przydatne źródła

Dokumentację funkcjonalności docker compose znajdziesz tutaj.

Bardzo dobry opis wymiany plików pomiędzy aplikacjami znajdziesz tutaj.

## 5.6 Podsumowanie

Gratulacje!

Mam nadzieję, że w tym module opanowała/eś podstawowe metody konteneryzacji, co pozwoli Ci jeszcze sprawniej projektować architektury systemów sztucznej inteligencji i zarządzać odpowiednimi środowiskami produkcyjnymi.

W szczególności, mam nadzieję, że potrafisz już:

### Stworzyć, udostępnić i uruchomić obraz pojedynczej aplikacji

1. Zaprojektować obrazu z wykorzystaniem Dockerfile
2. Zbudować go
3. Udostępnić ten obraz innym
4. Pobrać gotowy obraz i uruchomić go w postaci kontenera.

### Wykorzystać istniejące kontenery do stworzenia własnej aplikacji

A konkretnie, zidentyfikować i wykorzystać już stworzone obrazy

- z repozytoriów publicznych
- z własnego repozytorium.

### Zaimplementować mechanizmy wymiany plików w kanałach kontener<>host i kontener<>kontener

Czyli dobrać i wdrożyć optymalną w danym przypadku strategię wymiany informacji pomiędzy kontenerami i hostem. W szczególności, wymieniać dane, parametry sterujące i pliki, wykorzystując metody docker volume i docker mount.



**Tworzyć aplikacje wielokontenerowe** Projektować, udostępniać i uruchamiać aplikacje wielokontenerowych jako:

1. Równolegle działające, niezależne usługi
2. Usługi wymieniające między sobą dane
3. Usługi uruchamiane sekwencyjnie.

**Zarządzać obrazami, kontenerami i woluminami** korzystając z:

1. Linii komend
2. Aplikacji Docker Desktop
3. Środowisk IDE, np. MS VSCode.