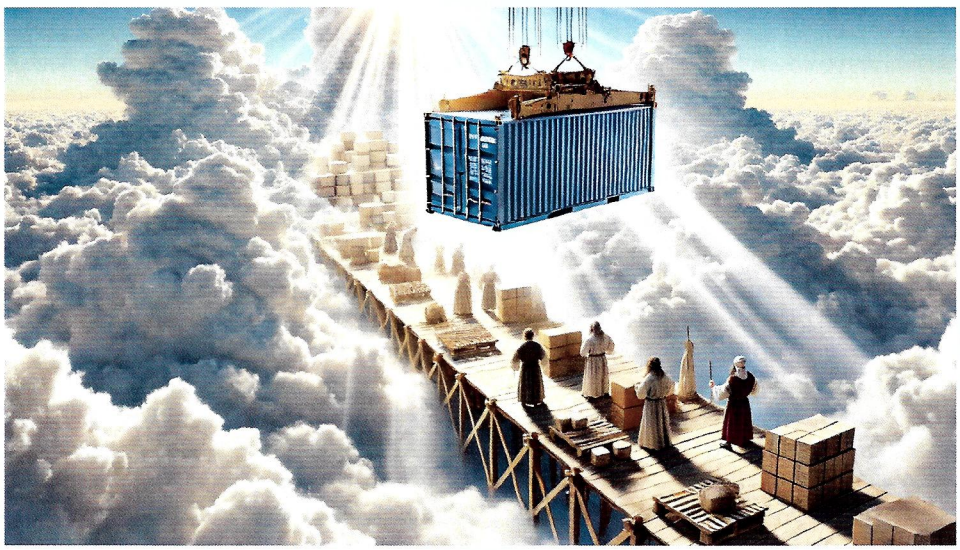
Docker – remedium na problem złożoności środowiska programistycznego

Wojciech Moszczyński

Wraz z rozwojem technologii i oprogramowania, złożoność środowisk, w których tworzone są aplikacje, znacząco wzrosła. Dawniej, gdy aplikacje były proste, a ich wymagania techniczne ograniczone, programiści nie mieli większych problemów z konfiguracją środowiska pracy. Obecnie jednak, w świecie zaawansowanego oprogramowania, problem kompatybilności bibliotek i wersji oprogramowania staje się powszechny. Problem ten stał się swego rodzaju codziennością, która nawet nie irytuje, ale zabiera bardzo dużo czasu i energii, która mogłaby być spożytkowana na tworzenie czegoś pożytecznego i rozwojowego.



Czym jest problem konfigurowania środowiska?

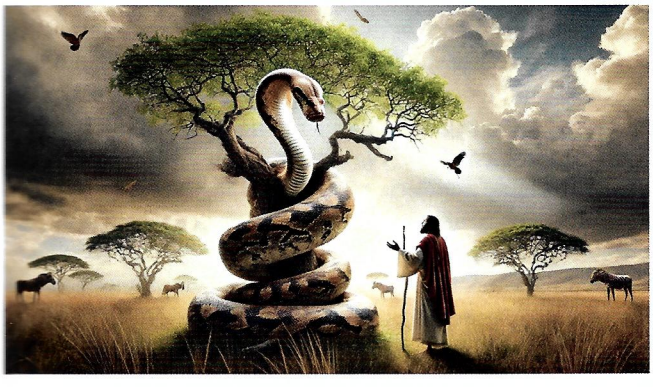
Środowisko programistyczne to zestaw narzędzi, bibliotek, zależności i systemów operacyjnych, w którym działa aplikacja. Rozważmy przykład języka Python. To popularny język programowania, który wymaga do działania wielu bibliotek, takich jak NumPy, Pandas czy TensorFlow.

Python jest językiem programowania, natomiast biblioteki są specjalistycznymi programami – jedne służą do grupowania i organizacji danych, tak jak wspomniana Pandas, inne jak TensorFlow czy Pytorch służą wyłącznie do tworzenia sieci neuronowych. Za pomocą samego Pythona oczywiście można zbudować sieć neuronową lub zorganizować dane. Wyspecjalizowane biblioteki jednak to znacznie ułatwiają i przyśpieszają. Kiedy zaczynałem moją przygodę z językiem Python odkryłem biblioteki i zacząłem je chętnie stosować. Niestety równoczesne stosowanie różnych bibliotek kończyło się konfliktami i zatrzymaniami aplikacji. Byłem zrozpaczony; coś co powinno działać bez problemu powodowało paraliż i anihilację gotowych rozwiązań. W ten sposób dowiedziałem się o konieczności konfigurowania środowiska – czynności wyjątkowo czasochłonnej i czasem bardzo irytującej.

Każda z tych bibliotek ma swoje wersje, które powstają w różnym czasie. Z czasem mogą pojawiać się nowe, które nie są w pełni kompatybilne z wcześniejszymi. Np., aplikacja napisana w Pythonie 3.6 może działać poprawnie z wersją NumPy 1.18, ale przestanie działać z nowszą wersją NumPy 1.21. Tego rodzaju problemy, związane z niedopasowaniem wersji, są częstym bólem głowy programistów. Python 3.6 działa z biblioteką Pandas w wersji 0.25.3, ale nowa wersja 1.0.0 wymaga już nowszej wersji Pythona, np. 3.7. W takim wypadku programiści muszą albo zaktualizować cały swój system, co może spowodować inne konflikty, albo znaleźć sposób na uruchomienie tych bibliotek w kompatybilny sposób. W takich sytuacjach pomocna jest platforma Anaconda, która umożliwia łatwe zarządzanie różnymi wersjami Pythona i bibliotek.

Anaconda jako rozwiązanie

Anaconda to środowisko, które rozwiązuje problem z kompatybilnością wersji bibliotek. Pozwala tworzyć izolowane środowiska, w których każda aplikacja ma dostęp do swoich specyficznych wersji bibliotek, co eliminuje konflikty między różnymi ich wersjami. Jednak, mimo że Anaconda doskonale radzi sobie z problemem bibliotek w ekosystemie Python, to problem ten występuje także w innych językach programowania i platformach. Anakonda to po prostu pakiet najczęściej używanych bibliotek, który jest spójny z programem wgrywanym do komputera. Rzadko jednak jeden pakiet bibliotek wystarczy do wykonania projektu. Trzeba ciągle coś dogrywać, teoretycznie dograne kolejnej biblioteki mogłoby zaburzyć spójność oprogramowania wewnątrz ekosystemu Anaconda. Twórcy rozwiązania przewidzieli ten scenariusz i stworzyli rozwiązanie, tzw. pip.

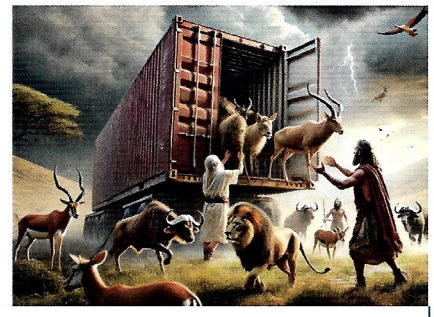


Czym jest pip?

Skrót pip oznacza *pip installs packages*. Jest to rekurencyjny akronim, co oznacza, że rozwinięcie zawiera samą nazwę. Pip jest narzędziem do instalowania i zarządzania pakietami Python, pobierając je głównie z Python Package Index (PyPI). Pip automatycznie zajmuje się instalacją, aktualizacją oraz zarządzaniem zależnościami potrzebnymi do działania danej aplikacji lub projektu w Pythonie. Jest to standardowy menedżer pakietów, który pozwala na instalowanie, aktualizowanie i usuwanie bibliotek oraz zależności niezbędnych do pracy z Pythonem. Pip pobiera pakiety z oficjalnego repozytorium Python Package Index (PyPI), gdzie znajduje się ogromna liczba bibliotek, które mogą być wykorzystywane do różnych zadań, od obliczeń naukowych po rozwój aplikacji webowych. Pozwala na instalację pakietów, aktualizację już zainstalowanych bibliotek oraz ich usuwanie. Jest to narzędzie niezwykle przydatne, ponieważ programista nie musi dbając o kompatybilność wersji i zależności.

Pomysł zamknięcia całego ekosystemu w „pudełku"

Oprogramowanie składa się nie tylko z kodu źródłowego, ale także z zestawu bibliotek, narzędzi i zależności, które muszą być odpowiednio skonfigurowane, aby działały razem poprawnie. Aby rozwiązać ten problem, ktoś wpadł na genialny pomysł – zamknięcia całego ekosystemu aplikacji w jednym, przenośnym środowisku. Takie podejście pozwala na zapakowanie wszystkich zależności i bibliotek w jeden kontener, który będzie działał wszędzie tak samo, niezależnie od środowiska, w którym go uruchomimy. Podobne rozwiązanie oferuje Anaconda, ale w szerszym zakresie taki sposób rozwiązywania problemów umożliwia Docker.



Izolowane środowiska to narzędzia, które pozwalają na uruchamianie aplikacji w odizolowany sposób, aby uniknąć problemów związanych z zależnościami. Do najpopularniejszych narzędzi do tworzenia takich środowisk należą Virtualenv, wspomniana już Conda (Anaconda) oraz Docker.

Co to jest Docker?

Docker to platforma open-source, która umożliwia tworzenie, wdrażanie i uruchamianie aplikacji w izolowanych kontenerach. Pomysł na Docker powstał w 2013 roku, kiedy Solomon Hykes, założyciel firmy dotCloud, zaczął pracować nad narzędziem do uproszczenia wdrażania aplikacji w różnych środowiskach. Historia Dockera zaczyna się od potrzeby stworzenia narzędzia, które umożliwiłoby uruchamianie aplikacji w spójnym środowisku na różnych maszynach – lokalnych, testowych czy produkcyjnych – bez potrzeby ręcznego konfigurowania każdej z nich.

Jedną z anegdot związanych z powstaniem Dockera jest fakt, że Hykes początkowo nie zamierzał udostępniać swojego projektu publicznie, ale po kilku rozmowach i eksperymentach na konferencjach zdecydował się w 2013 roku wypuścić pierwszą wersję Docker na GitHub. Od tego momentu Docker stał się jednym z najważniejszych narzędzi w świecie IT.

Z czego składa się Docker?

Docker składa się z kilku kluczowych elementów.

1. **Kontenery** – lekkie, przenośne środowiska, w których działa aplikacja. Każdy kontener zawiera wszystkie potrzebne do działania aplikacji zależności.
2. **Docker Engine** – silnik Docker, który zarządza kontenerami i odpowiada za ich uruchamianie oraz izolację.
3. **Obrazy Docker** – zapisane środowiska aplikacji, które mogą być uruchamiane jako kontenery. Obraz Docker to „snapshot" aplikacji i jej zależności.
4. **Dockerfile** – plik konfiguracyjny, który zawiera instrukcje, jak zbudować obraz Docker.

Tworzenie Dockera w praktyce

Tworzenie Dockera jest proste. Aby stworzyć kontener dla aplikacji, programista pisze plik konfiguracyjny o nazwie Dockerfile. Przykład prostego Dockerfile w formie kilku linijek tekstowych. Dockerfile to plik tekstowy taki jak txt; może składać się np. z 7 linijek:

```
dockerfile
Copy code
FROM python:3.8
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
CMD [„python", „app.py"]
```

Ten plik konfiguracyjny definiuje bazowy obraz Pythona w wersji 3.8, ustawia katalog roboczy, kopiuje pliki do kontenera, instaluje zależności z pliku requirements.txt i uruchamia aplikację app.py. Najważniejszą rolą Dockera jest implementacja prototypowego oprogramowania do chmury w celu jego wykorzystania produkcyjnego.

Chmury obliczeniowe i ich rola

Chmury obliczeniowe to zewnętrzne infrastruktury IT, które umożliwiają przechowywanie danych i uruchamianie aplikacji bez konieczności posiadania własnych serwerów. Chmury, takie jak AWS, Google Cloud czy Azure, są używane do uruchamiania aplikacji w sposób skalowalny i dostępny z różnych miejsc na świecie. Aplikacje chmurowe pozwalają na dynamiczne przydzielanie zasobów, co zwiększa ich elastyczność.

Docker w chmurze

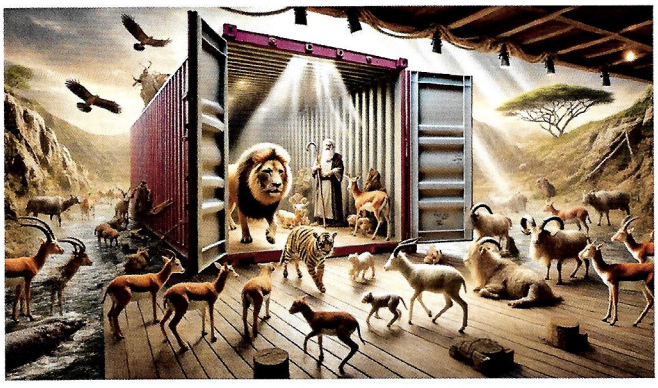
Programy napisane w Pythonie (i nie tylko) często są umieszczane w chmurach, jednak mogą pojawić się problemy z ich konfiguracją. Chmura to taki sam ekosystem informatyczny jaki mamy w laptopach i telefonach. Chmura to po prostu serwer działający gdzieś na świecie i połączony z nami szybkim łączem. Podobnie jak w komputerach, również na różnych serwerach chmurowych może wystąpić brak kompatybilności wersji bibliotek, systemów operacyjnych czy zależności. W takich przypadkach Docker jest idealnym rozwiązaniem, ponieważ umożliwia zapakowanie całego środowiska aplikacji w jeden obraz, który można uruchomić na dowolnym serwerze, niezależnie od jego konfiguracji.

Załóżmy, że ktoś napisał aplikację w Pythonie, która działa lokalnie, ale ma problemy podczas uruchamiania na serwerze chmurowym z powodu różnic w wersjach bibliotek. Użycie Dockera umożliwia przeniesienie tej aplikacji razem z całym jej środowiskiem, co sprawia, że działa ona tak samo na każdym serwerze, eliminując problemy związane z konfiguracją.

Pakowanie w kontener

Czasami zdarza się, że inżynierowie danych, skupiając się na tworzeniu modeli i analizach, nie w pełni rozumieją, jak działa cały system, na którym uruchamiane są ich aplikacje. Zajmują się zaawansowanymi algorytmami, stosują skomplikowane techniki uczenia maszynowego i przemyślane rozwiązania, ale gdy przychodzi do wdrożenia ich pracy w rzeczywistym środowisku, napotykają na problemy. W takich sytuacjach, zamiast dokładnie analizować infrastrukturę czy środowisko, w którym aplikacja ma działać, często uciekają się do prostszego podejścia: *zapakujmy to wszystko w Dockera i przenieśmy do chmury*.

Wyobraźmy sobie inżyniera danych, który stworzył zaawansowany model uczenia maszynowego w Pythonie. Model ten zależy od wielu bibliotek, takich jak Pandas, scikit-learn,



TensorFlow, a każda z tych bibliotek ma swoje specyficzne wymagania co do wersji i zależności systemowych. Zamiast szczegółowo analizować, które wersje bibliotek będą działać najlepiej w środowisku produkcyjnym, inżynier po prostu pakuje wszystko do jednego kontenera Docker, łącznie z całym Pythonem i zależnościami. Następnie kontener ten zostaje przeniesiony do chmury, gdzie jest uruchamiany.

Takie podejście często wynika z braku pełnego zrozumienia infrastruktury czy środowiska produkcyjnego. Docker stał się czymś w rodzaju magicznego pudełka, które „rozwiązuje wszystkie problemy". Oczywiście, Docker jest potężnym narzędziem, ale takie podejście może prowadzić do marnotrawienia zasobów – uruchamiania niepotrzebnie dużych obrazów z wieloma zależnościami, które mogłyby być zoptymalizowane lub zredukowane. Docker nie został stworzony z myślą o tym, aby być rozwiązaniem dla niejasnych, niezrozumianych rozwiązań. Jego głównym celem było uproszczenie wdrażania aplikacji przez izolację środowisk, ale nie jako wytrych na każdy problem. Inżynierowie danych często jednak traktują Dockera właśnie w ten sposób – jako narzędzie, które pomoże im ominąć problemy związane z konfiguracją i kompatybilnością. To dodatkowa funkcja, której twórcy Dockera nie przewidzieli: ucieczka od szczegółowego zrozumienia systemu na rzecz „zapakowania wszystkiego" w kontener i wrzucenia do chmury.

Podsumowanie

Docker to potężne narzędzie, które rozwiązuje problem kompatybilności oprogramowania w różnych środowiskach. Zamiast martwić się o wersje bibliotek i zależności, programiści mogą zamknąć cały ekosystem aplikacji w kontenerze, który będzie działał identycznie na każdym serwerze. Docker nie tylko ułatwia pracę programistom, ale także zwiększa efektywność wdrażania aplikacji w chmurach obliczeniowych, ponieważ nie muszą oni wszystkiego rozumieć i często ich rola sprowadza się do funkcji windowego, wysyłającego wszystko „windą" do chmury.

Wojciech Moszczyński

Wojciech Moszczyński – absolwent Katedry Ekonometrii i Statystyki Uniwersytetu Mikołaja Kopernika w Toruniu; specjalista z zakresu ekonometrii, finansów, data science oraz rachunkowości zarządczej. Specjalizuje się w optymalizacji procesów produkcyjnych i logistycznych. Prowadzi badania w obszarze rozwoju i zastosowania sztucznej inteligencji. Od lat zaangażowany w popularyzację machine learning oraz data science w środowiskach biznesowych.