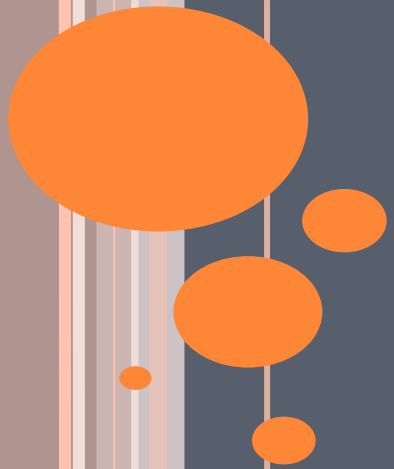
W4

PROGRAMOWANIE W CHMURZE W OPARCIU O MIKROSERWISY - WYTYCZNE “12 FACTORS”



W4 Idea powstania wytycznych „12 Factors”

Obecnie bardzo dużą popularnością cieszy się dostarczanie oprogramowanie jako SaaS. Omawiane w tym wykładzie 12 wytycznych (ang. 12 Factors) jest opisem metodologii tworzenia oprogramowania jako usługi, które opierają się na koncepcji mikrousług a również:

- wykorzystują deklaratywnych formatów do automatyzacji konfiguracji, aby zminimalizować czas i koszty realizacji/rozwoju projektu,
- mają jasno określony związek (zależności) z bazowym systemem operacyjnym, gwarantując maksymalna przenośność pomiędzy różnorodnymi środowiskami wykonawczymi,
- spełniają wymogi wdrożenia na platformach chmurowych, minimalizując lub eliminując potrzebę administrowania serwerami i systemami,
- minimalizują rozbieżności pomiędzy fazą rozwoju a fazą produkcji, umożliwiając tym samym stosowanie procedur ciągłego wdrażania,
- stwarzają możliwość skalowania aplikacji bez konieczności wprowadzania znaczących zmian w narzędziach, architekturze lub metodach programistycznych.

Idee przedstawił Adam Wiggins ze wsparciem serwisu Heroku

https://12factor.net/ https://www.heroku.com/

W4 Definicja i struktura wytycznych „12 Factors”

Definicja zaczerpnięta z Wikipedii: https://en.wikipedia.org/wiki/Twelve-Factor_App_methodology

The Twelve-Factor App methodology is a methodology for building software-as-a-service applications. These best practices are designed to enable applications to be built with portability and resilience when deployed to the web.

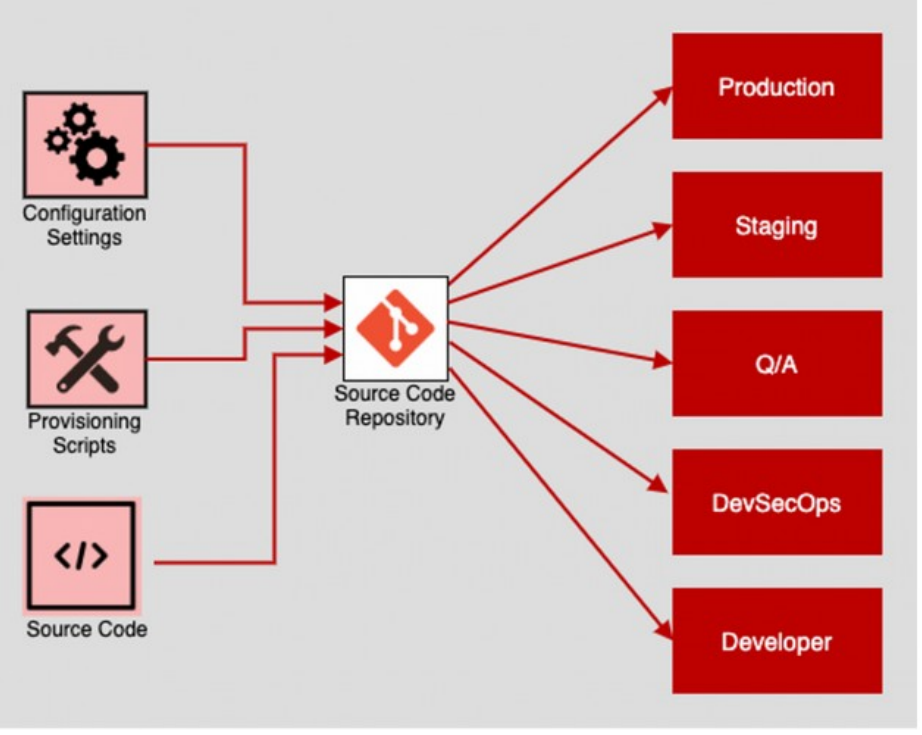
#	Factor	Description
I	Codebase	There should be exactly one codebase for a deployed service with the codebase being used for many deployments.
II	Dependencies	All dependencies should be declared, with no implicit reliance on system tools or libraries.
III	Config	Configuration that varies between deployments should be stored in the environment.
IV	Backing services	All backing services are treated as attached resources and attached and detached by the execution environment.
V	Build, release, run	The delivery pipeline should strictly consist of build, release, run.
VI	Processes	Applications should be deployed as one or more stateless processes with persisted data stored on a backing service.
VII	Port binding	Self-contained services should make themselves available to other services by specified ports.
VIII	Concurrency	Concurrency is advocated by scaling individual processes.
IX	Disposability	Fast startup and shutdown are advocated for a more robust and resilient system.
X	Dev/Prod parity	All environments should be as similar as possible.
XI	Logs	Applications should produce logs as event streams and leave the execution environment to aggregate.
XII	Admin Processes	Any needed admin tasks should be kept in source control and packaged with the application.

W4 1. CODEBASE

Jedna baza kodu skojarzona z systemem kontroli wersji jako podstawa wielu wdrożeń (1)

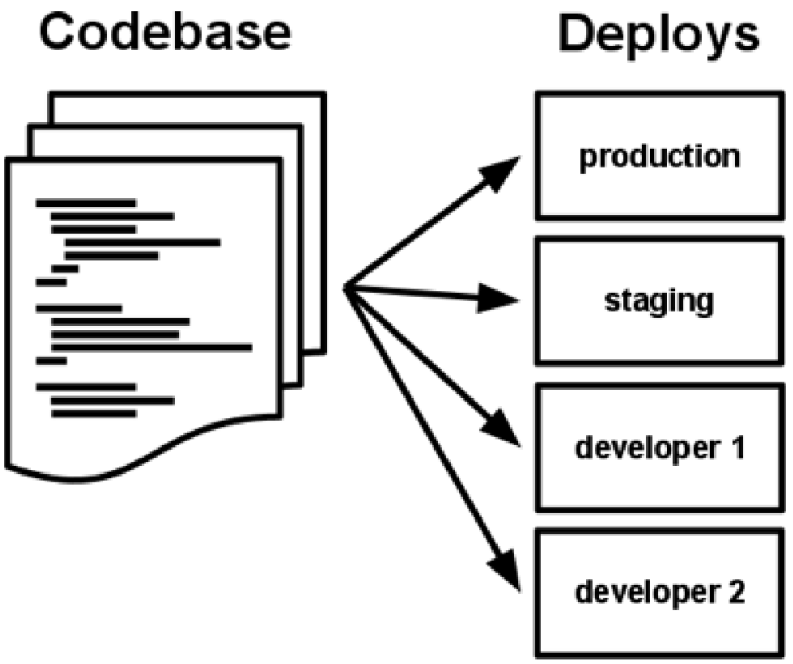
Pierwsza z omawianych zasad, Codebase definiuje centralną rolę jako powinna pełnić repozytorium kodu źródłowego. Repozytorium to przechowuje wszystkie zasoby związane z aplikacją, od kodu źródłowego, skryptu udostępniania po ustawienia konfiguracyjne I udostępnia je programistom realizującym dany projekt.

Repozytorium kodu źródłowego jest również dostępne dla wszystkich skryptów automatyzacji, które są częścią procesów ciągłej integracji/ciągłego dostarczania (CI/CD), które są częścią cyklu rozwoju oprogramowania danego przedsiębiorstwa (SDLC).



W4 1. CODEBASE

Jedna baza kodu skojarzona z systemem kontroli wersji jako podstawa wielu wdrożeń (2)



Według omawianej zasady, powinna istnieć tylko jedna baza kodu na aplikację, niezależnie jak wiele będzie wdrożeń tej aplikacji. Każdy programista ma kopię aplikacji uruchomioną w swoim lokalnym środowisku programistycznym, z których każde również kwalifikuje się jako wdrożenie (na rys. developer 1, developer 2). Pozostałe wdrożenia to zazwyczaj wdrożenie w lokalizacji produkcyjnej i co najmniej jedno wdrożenie pomostowe (testowe).

Baza kodu to dowolne pojedyncze repozytorium (w scentralizowanym systemie kontroli wersji, takim jak Subversion), lub dowolny zestaw repozytoriów, które współdzielą commit-y (w zdecentralizowanym systemie kontroli wersji, takim jak Git).

W4 1. CODEBASE

Jedna baza kodu skojarzona z systemem kontroli wersji jako podstawa wielu wdrożeń (3)

Zawsze istnieje korelacja jeden do jednego między bazą kodu a aplikacją:

- Jeśli istnieje wiele baz kodu, nie jest to aplikacja — jest to system rozproszony. Każdy składnik w systemie rozproszonym jest aplikacją, a każdy z nich może indywidualnie spełniać wymogi zaleceń ujętych w “12 Factors”.

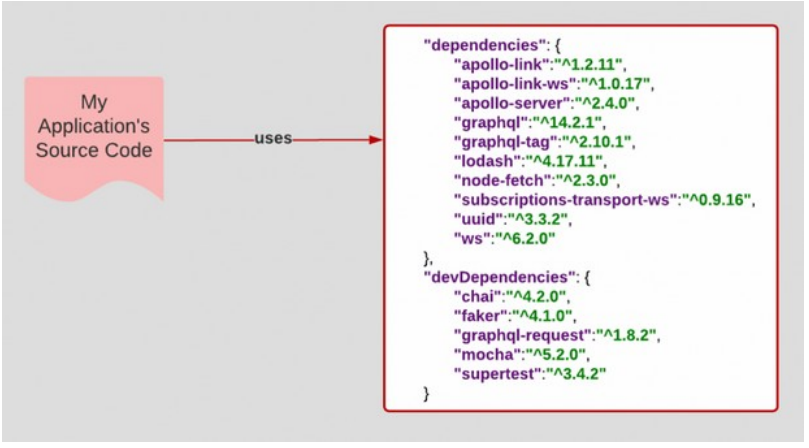
- Wiele aplikacji udostępniających ten sam kod jest naruszeniem zaleceń 12 Factors. Rozwiązaniem jest tutaj rozłożenie kodu współdzielonego na biblioteki, które można dołączyć za pomocą dostępnego menedżera zależności.

W projektach opartych o kontenery Docker, opracowywana jest jedna aplikacja na kontener. Taka aplikacja powinna mieć swój własny plik Dockerfile i zazwyczaj bardziej zoptymalizowaną wersję do produkcji, np. plik prod.Dockerfile.

Obrazy są artefaktem wdrażania. Wobec tego przepływ CI/CD powinien zakończyć się nową wersją obrazu aplikacji przesłaną do wykorzystywanego rejestru.

W4 2. DEPENDENCIES

Jawne deklaracje i izolacja zależności (1)



- Zasada DEPENDENCIES definiuje politykę, która gwarantuje, że tylko kod, który jest bezpośrednio związany i przeznaczony do realizacji celu aplikacji, jest przechowywany w repozytorium kodu źródłowego.

- Składniki zewnętrzne, takie jak np. pakiety Node.js, pliki Java .jar lub biblioteki DLL .NET, powinny być przywoływane w tzw. manifeście zależności ładowanym do pamięci podczas fazy rozwijania oprogramowania, testowania i wdrożenia w wybranym środowisku uruchomieniowym.

WNIOSEK: Należy unikać przechowywania zależności w postaci zewnętrznych elementów wraz z kodem źródłowym w repozytorium kodu źródłowego.

W4 2. DEPENDENCIES

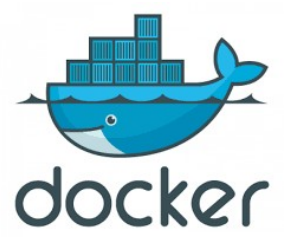
Jawne deklaracje i izolacja zależności (2)

- Aplikacja zgodna z “12 Factors” nigdy nie opiera się na niejawnym istnieniu pakietów systemowych. Wręcz przeciwnie, zawiera deklaracje wszystkich zależności, zazwyczaj za pomocą wspomnianego manifestu deklaracji zależności.

- Zaleca się wykorzystywanie metod i narzędzi, które minimalizują prawdopodobieństwo, że jakieś niejawne zależności „wyciekają” z systemu “otaczającego” daną aplikację.

- Pełna i jawna specyfikacja zależności powinna być stosowana zarówno w fazie produkcyjnej jak i rozwoju I testowania programowania.

Przykład: Python posiada dwa oddzielne narzędzia zgodne z omawianym zaleceniem:
- Pip wykorzystywane do deklaracji zależności,
- Virtualenv do realizacji izolacji.

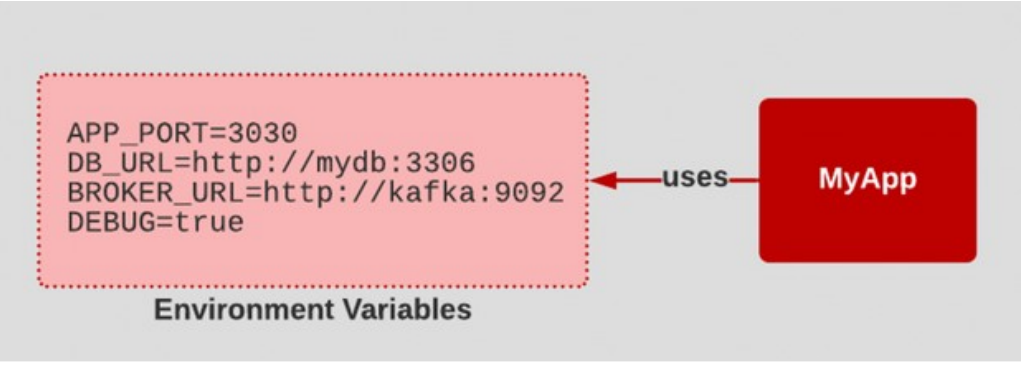


Środowisko Docker bardzo dobrze współgea z zaleceniem DEPENDENCES. W pliku Dockerfile zawiera opis każdej pojedynczej zależność, której jest wymagana przez aplikację, a która realizowana jest na poziomie obranego systemu operacyjnego.

W4 3. CONFIG

Przechowywanie konfiguracji w deklaracjach środowiska (1)

Zasada CONFIGi określa, że informacje konfiguracyjne powinny być przekazywane do wykorzystywanego środowiska wykonawczego jako zmienne środowiskowe lub jako ustawienia zdefiniowane w niezależnym pliku konfiguracyjnym.



WNIOSEK: W wyjątkowych przypadkach (ale jest to formalnie naruszenie zasad “12 Factors”) dozwolone jest przechowywanie ustawień domyślnych, które można zastąpić bezpośrednio w fazach uruchomieniowych (ang. runtime) ale generalna zasada mówi, że ustawienia, takie jak numer portu, adresy URL zależności i ustawienia stanu, takie jak DEBUG, powinny istnieć osobno i być stosowane podczas wdrażania.

W4 3. CONFIG

Przechowywanie konfiguracji w deklaracjach środowiska (2)

Plik konfiguracji aplikacji to wszystko, co może się różnić w zależności od etapu wdrożenia. Zaleca się by zawierał on następujące informacje:

- Dowiązania do zasobów pełniących rolę usług wsparcia (ang. backing services), np. do bazy danych,
- Poświadczenia (hasła, klucze itp) do usług zewnętrznych, np. Amazon S3.
- Wartości dla każdego wdrożenia, takie jak kanoniczna nazwa hosta dla wdrożenia

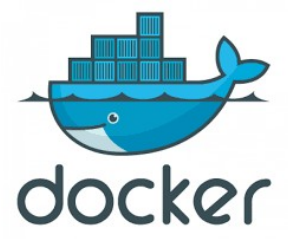
Aplikacja zgodna z “12 Factors” przechowuje konfigurację w zmiennych środowiskowych (często nazywanych w skrócie: env vars lub env). Zmienne środowiskowe można łatwo zmieniać między wdrożeniami bez zmiany kodu. W przeciwieństwie do plików konfiguracyjnych, istnieje niewielka szansa, że zostaną one przypadkowo zaewidencjonowane w repozytorium kodu oraz są one, w ogólnym przypadku, elementem niezależnym od wykorzystywanego języka programowania i systemu operacyjnego.

W4 3. CONFIG

Przechowywanie konfiguracji w deklaracjach środowiska (3)

UWAGA: W aplikacji zgodnej z “12 Factors” zmienne środowiskowe są szczegółowymi narzędziami kontroli, z których każda jest w pełni ortogonalna względem innych zmiennych środowiskowych. Nigdy nie są one grupowane jako „środowiska”, ale są zarządzane niezależnie dla każdego wdrożenia. Jest to model, który płynnie się skaluje, ponieważ aplikacja w naturalny sposób rozwija się do większej liczby wdrożeń w ciągu swojego życia.

Przykład: Zewnętrzne pliki konfiguracyjne, takie jak Java properties, Kubernetes manifest, lub plik docker-compose.yml.

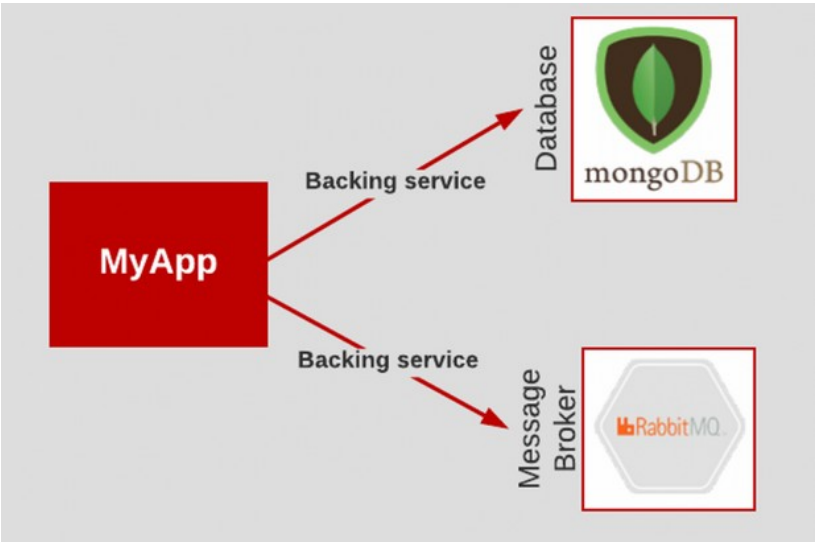


W środowisku Docker spełnienie omawianego zalecenia polega na przekazywaniu konfiguracji za pomocą zmiennych środowiskowych, tzw. secrets, ConfigMaps lub w jakikolwiek inny sposób, na jaki pozwala danu orkiestrator kontenerów. Realizacja odbywa się w oparciu o argumenty wiersza poleceń podczas uruchamiania kontenera lub pliki docker-compose.

W4 4. BACKING SERVICES

Traktowanie usługi wspierających (ang. backing services) jako dołączonych zasobów (1)

Zasada BACKING SERVICES nakłada na architektów usług wymóg do traktowania komponentów zewnętrznych, takich jak bazy danych, serwery poczty e-mail, brokerów wiadomości i inne niezależne usługi, które mogą być udostępniane lub obsługiwane przez dany system, jako usługi wspierające.



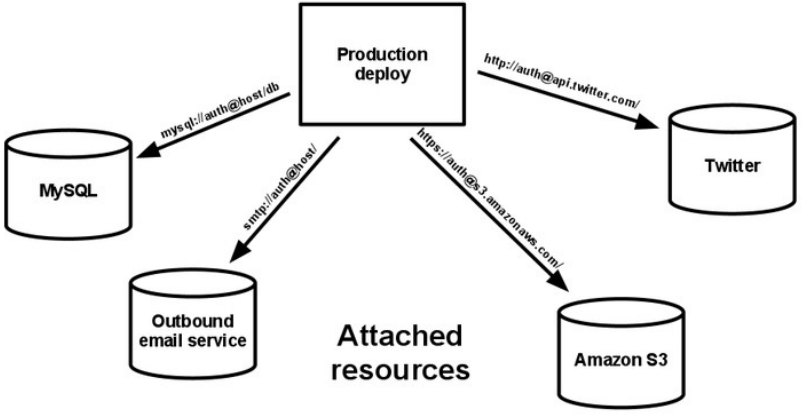
Uwaga: Każda odrębna usługa wspierająca jest zasobem. Na przykład baza danych MySQL jest zasobem; dwie bazy danych MySQL (używane do shardingu w warstwie aplikacji) kwalifikują się jako dwa różne zasoby.

Traktowanie zasobów jako usług pomocniczych zdecydowanie elastyczność i wydajność w cyklu życia oprogramowania (SDLC).

W4 4. BACKING SERVICES

Traktowanie usługi wspierających (ang. backing services) jako dołączonych zasobów (2)

Kod aplikacji zgodnej z “12 Factors” nie rozróżnia usług lokalnych i usług stron trzecich. Oba rodzaje są dołączonymi do aplikacji zasobami, do których można uzyskać dostęp za pośrednictwem adresu URL lub innego typu lokalizatora/poświadczenia przechowywanych w pliku konfiguracji.

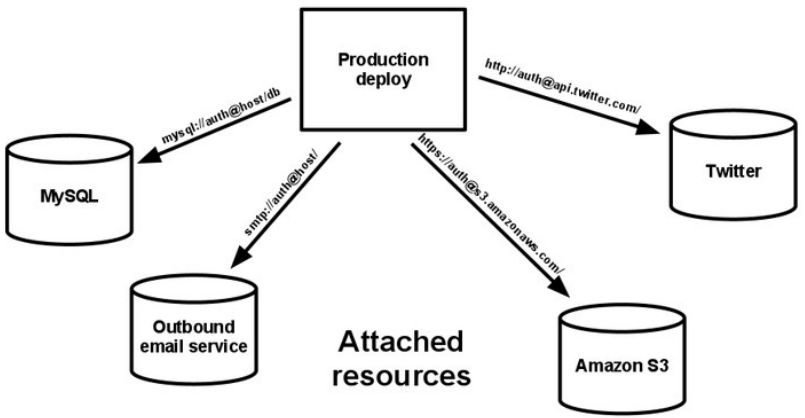


Wdrożenie aplikacji według zasady BACKING SERVICE powinno dawać możliwość elastycznej zmiany źródła usłu wspierających. Przykładowo, powinna być możliwość zamiany lokalnej bazy danych MySQL na bazę zarządzaną przez stronę trzecią (taką jak Amazon RDS) bez żadnych zmian w kodzie aplikacji. Podobnie lokalny serwer SMTP można zamienić na usługę SMTP innej firmy bez zmiany kodu. W obu przypadkach należy zmienić tylko definicję przyłączenia zasobu w konfiguracji.

W4 4. BACKING SERVICES

Traktowanie usługi wspierających (ang. backing services) jako dołączonych zasobów (3)

Ponieważ kontenery Docker zawierają generalnie aplikacje bezstanowe, potrzebny jest mechanizm “zapamiętania” jakiegoś stanu, aby wykonać określoną operację w ramach tej aplikacji. Niezależnie od tego, czy jest to baza danych, pamięć obiektowa, czy system kolejkowy, zalecane jest uruchamianie tych mechanizmów w dedykowanych usługach poza np. klastrem i przekazywać za pomocą pliku konfiguracyjnego (zalecenie: CONFIG)



Możliwa jest sytuacja, gdy usługi wspierające są zależne od innych usług, które również działają np. na danym klastrze. W takim przypadku, np. Kubernetes zapewnia sposób na znalezienie usług działających w klastrze (usługa: service discovery). Dzięki temu możliwe jest używanie nazwy usługi Kubernetes jako nazwy hosta przy czym opracowywana aplikacja musi również otrzymać te dane jako konfigurację, nawet jeśli nie zmienia się one zbyt często.

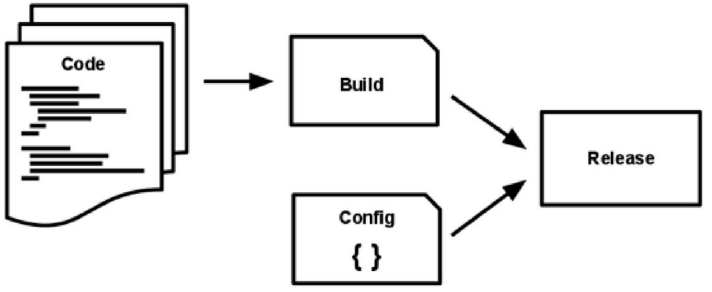
W4 5. BUILD, RELEASE, RUN

Ścisłe oddzielanie etapów budowania i uruchamiania (1)

Baza kodu jest przekształcana we wdrożenie (niedeweloperskie) w trzech etapach:

- Etap kompilacji (ang. build) to transformacja, która konwertuje repozytorium kodu na wykonywalny pakiet znany jako build. Korzystając z wersji kodu w danym commit-cie określonym przez proces wdrażania, etap kompilacji pobiera zależności z zewnętrznych źródeł oraz kompiluje do postaci plików binarnych i ewentualnie pozostałych zasobów.

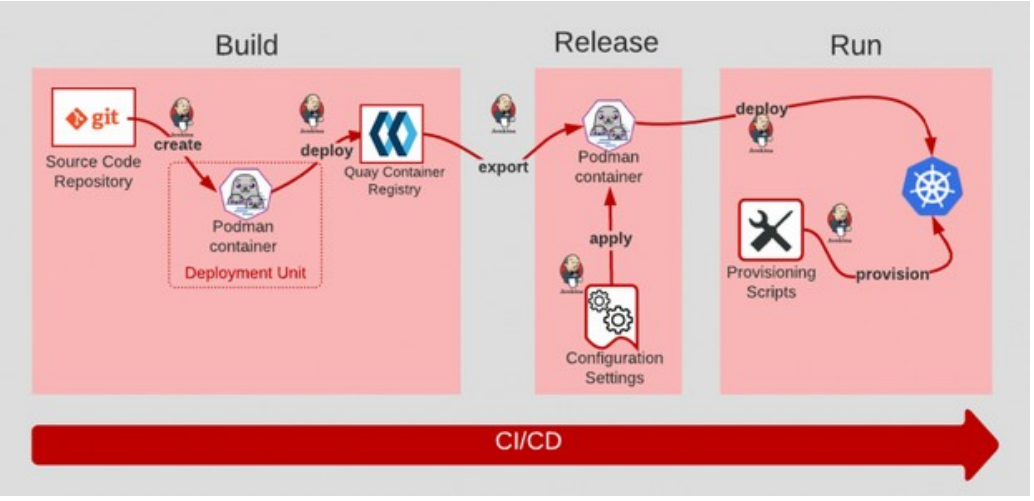
- Etap wdrożenia (ang. release) przejmuje kompilację wytworzoną na etapie kompilacji i łączy ją z bieżącą konfiguracją wdrożenia. Otrzymane wydanie zawiera zarówno kompilację, jak i konfigurację i jest gotowe do natychmiastowego wykonania w wybranym środowisku wykonawczym.



- Etap uruchamiania (ang. run) uruchamia aplikację w środowisku wykonawczym, uruchamiając pewien zestaw procesów aplikacji w wybranym wydaniu.

W4 5. BUILD, RELEASE, RUN

Ścisłe oddzielanie etapów budowania i uruchamiania (2)



Kluczem do kompilacji, wdrożenia i uruchomienia jest to, że procesy te są całkowicie efemeryczne. Jeśli cokolwiek w potoku etapów zostanie zniszczone, wszystkie configuracje i zmienne środowiska można odtworzyć od podstaw przy użyciu zasobów przechowywanych w repozytorium kodu źródłowego.

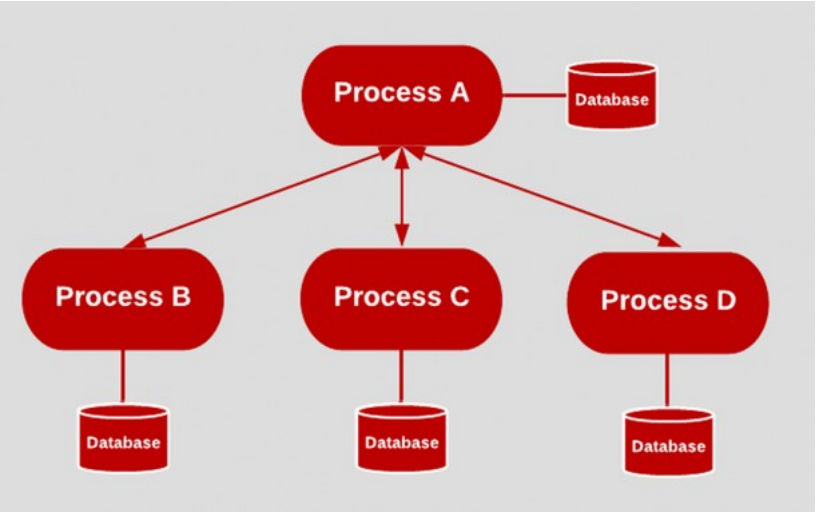
Zgodność z tym zalecenie jest łatwa do dostrzeżenia w środowisku Docker:

Build: docker build ...
Release: docker push ...
Run: kubectl apply -f ...

W4 6. PROCESSES

Uruchamianie aplikacji jako jeden lub więcej procesów bezstanowych (1)

Zasada procesów, które można dokładniej nazwać procesami bezstanowymi, zakłada, że aplikacja opracowana w zgodzie z zasadami “12 Factors” będzie działać jako zbiór procesów bezstanowych.



- Żaden pojedynczy proces nie śledzi stanu innego procesu i żaden proces nie śledzi informacji, takich jak stan sesji lub przepływ damych.

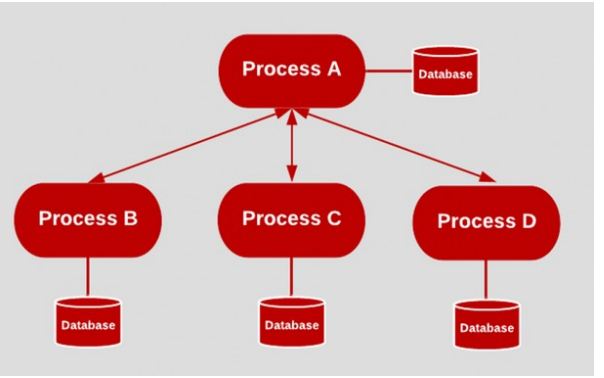
- Proces bezstanowy ułatwia skalowanie. Gdy proces jest bezstanowy, instancje mogą być dodawane i usuwane w celu obsłużenia określonego obciążenia w danym momencie.

- Ponieważ każdy proces działa niezależnie, bezstanowość zapobiega niezamierzonym skutkom ubocznym podczas realizacji procesów inicjowanych i nadzorowanych przez dany orkiestrator.

W4 6. PROCESSES

Uruchamianie aplikacji jako jeden lub więcej procesów bezstanowych (2)

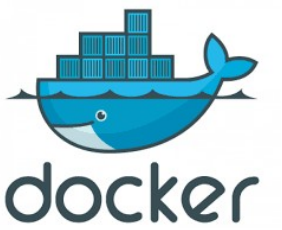
WNIOSEK: Procesy w aplikacjach zgodnych z “12 Factors” są bezstanowe i nie dzielą się zasobami. Wszelkie dane, które muszą być trwałe, muszą być przechowywane w stanowej usłudze tworzenia kopii zapasowych, zwykle w bazie danych.



UWAGA: Przestrzeń pamięci lub system plików procesu może być używany jako, pojedyncza pamięć podręczna transakcji. Na przykład pobranie dużego pliku, operowanie na nim i przechowywanie wyników operacji w bazie danych. Aplikacja zgodna z “12 Factors” nigdy nie zakłada, że cokolwiek z pamięci podręcznej lub na dysku będzie dostępne dla przyszłego żądania lub zadania. Nawet w przypadku działania tylko jednego procesu ponowne uruchomienie (zainicjowane przez wdrożenie kodu, zmianę konfiguracji lub środowisko wykonawcze przenoszące proces do innej fizycznej lokalizacji) zazwyczaj wyczyszcza cały stan lokalny (np. pamięć i system plików).

W4 6. PROCESSES

Uruchamianie aplikacji jako jeden lub więcej procesów bezstanowych (3)



Kontener Docker powinien być samowystarczalny. W idealnym przypadku powinien zawierać jeden proces jako PID 1. Jednak w rzeczywistości zdażają się sytuacje, gdy niezbędna jest większa liczba procesów.

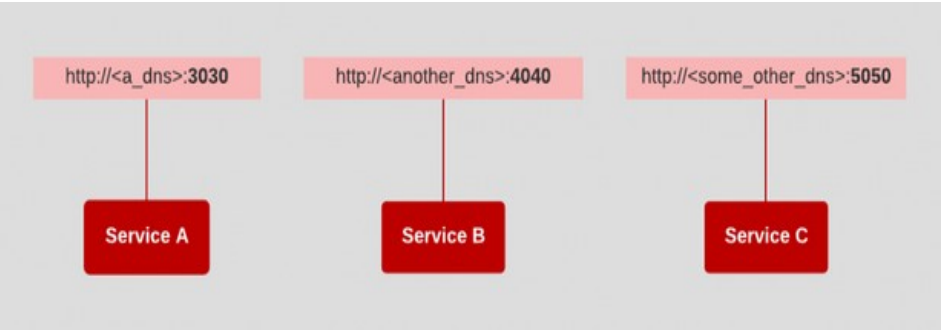
- Dobrym przykładem jest PHP. Do obsługi żądań HTTP potrzebne są zarówno Nginx, jak i PHP-FPM. W takim przypadku proste rozwiązanie oferuje Kubernetes – jego najmniejszą jednostką nie jest kontener a tzw. pod. Pod składa się z minimum 1 kontenera. Teoretycznie może działać kontener PHP-FPM i inny kontener dla Nginx i oba mogą się komunikować za pośrednictwem gniazd TCP (przez localhost, ponieważ działają na tym samym pod). Kontener PHP-FPM ma jeden proces główny i kilka procesów potomnych, ale PHP już nic nie udostępnia, więc nie narusza omawianego zalecenia.

UWAGA: Jeśli niezbędna jest sesja zachowująca kontekst użytkowników pomiędzy żądaniami, zaleca się wykorzystanie usług wspierających (zalecenie 4. Usługi wspierające). Wiele frameworków umożliwia korzystanie z różnych sterowników pamięci masowych dla sesji, takich jak Redis, Memcached, jak i liczne bazy danych.

W4 7. PORT BINDING

Eksportowanie usług przez wiązanie portów (1)

Ideę tego zalecenia dobrze ilustruje dowolna aplikacja webowa, która jest uruchamiana w kontenerze z serwerem http.



Zgodność z “12 Factors” oznacza, że aplikacja ta jest całkowicie samowystarczalna i nie zależy od dołączenia serwera http do środowiska wykonawczego w celu utworzenia usługi dostępnej w Internecie. Aplikacja webowa zamiast tego eksportuje protokół HTTP jako usługę przez powiązanie z portem i nasłuchiwanie żądań przychodzących przez ten port.

UWAGA: Zalecenie PORT BINDING oznacza, że jedna aplikacja może stać się usługą pomocniczą dla innej aplikacji, np. podając adres URL aplikacji tworzącej usługę pomocniczą.

W4 7. PORT BINDING

Eksportowanie usług przez wiązanie portów (2)

07-app.js

```
const app = express()
const port = 3000

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

07-docker-run.sh

```
docker run --rm -it -p "3000:3000" tonysm/myapp:1.0.0
```

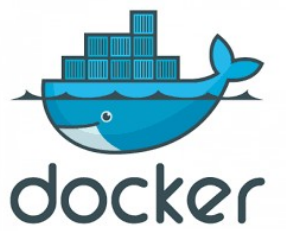
07-file.dockerfile

```
# ...

EXPOSE 3000

# ...
```

Kiedy istnieje potrzeba udostępnienia określonego zasobu kontenera , na przykład web UI, to wystarczy powiązać ten zasob z określonym portem,



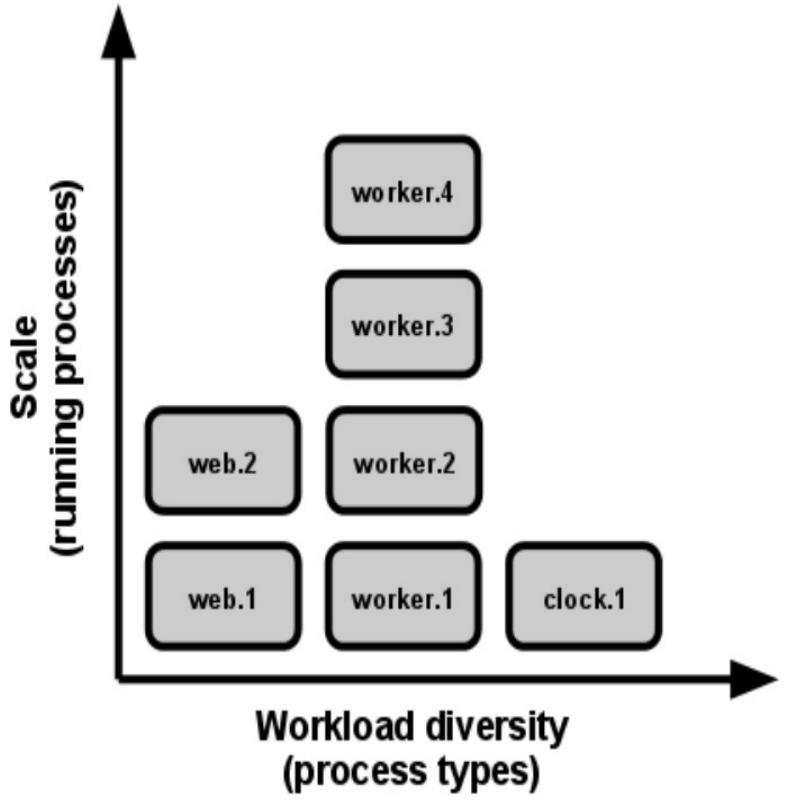
Uwaga: W zależności od orkiestratora użytego do uruchamiania kontenerów, port ma mniej lub bardziej ograniczone znaczenie. Na przykład, w Kubernetes można zdefiniować routing portów, dzięki czemu możliwe jest funkcjonowanie aplikacji odwołujących się do danej usługi za pośrednictwem myapp-svc:80, podczas gdy w rzeczywistości odwołuje się ona do portu 3000 w określonym podzie.

W4 8. CONCURRENCY

Skalowanie bazujące na modelu procesu (1)

W odniesieniu do tego zalecenia, skala jest wyrażona jako ilość uruchomionych (powielonych) procesów, różnorodność obciążenia jest wyrażona poprzez typy procesów.

Procesy w aplikacji zgodnych z “12 Factors” bazują na modelu procesów uniksowych w procesie uruchamiania demonów usług. Korzystając z tego modelu, programista może zaprojektować swoją aplikację pod kątem obsługi różnych obciążeń, przypisując każdy typ pracy do określonego typu procesu.

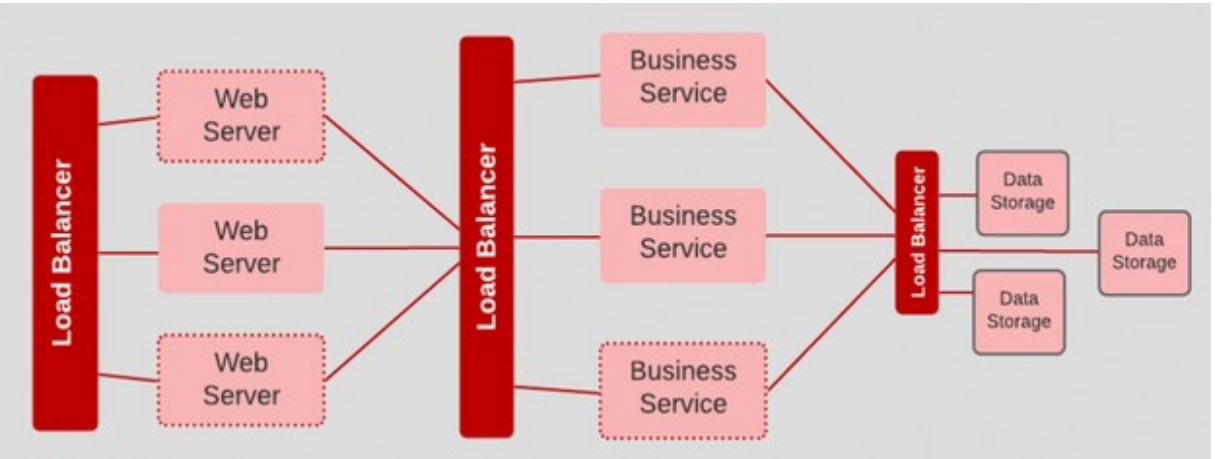


Na przykład żądania HTTP mogą być obsługiwane przez proces sieciowy, a długotrwałe zadania w tle obsługiwane przez proces roboczy.

W4 8. CONCURRENCY

Skalowanie bazujące na modelu procesu (2)

Zasada CONCURENCY zaleca organizowanie procesów zgodnie z ich przeznaczeniem, a następnie rozdzielanie tych procesów tak, aby można je było skalować w górę I w dół w zależności od potrzeb.

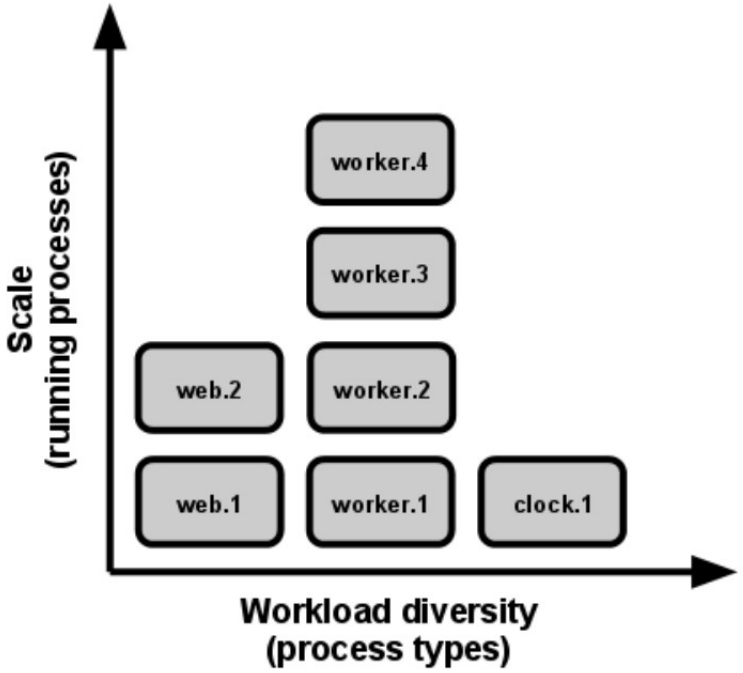


Na rysunku, aplikacja jest udostępniana w sieci za pośrednictwem serwerów internetowych, które działają za systemem równoważenia obciążenia. Z kolei grupa serwerów WWW za load balancerem wykorzystuje logikę biznesową, która znajduje się w procesach Business Service, które działają za własnym load balancerem.

Jeśli obciążenie serwerów webowych wzrośnie, grupa ta może zostać powiększona w izolowany sposób, aby sprostać bieżącym wymaganiom. Jednak w przypadku wystąpienia wąskiego gardła z powodu obciążenia usługi biznesowej, warstwa Business Service może być skalowana niezależnie.

W4 8. CONCURRENCY

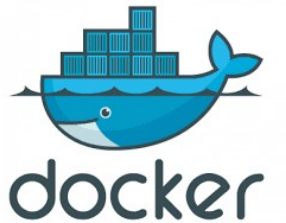
Skalowanie bazujące na modelu procesu (3)



WNIOSEK: Obsługa zalecenia CONCURENCY oznacza, że różne części aplikacji mogą być skalowane w celu zaspokojenia bieżących potrzeb. W przeciwnym razie, gdy zalecenie nie jest obsługiwane, architektury mają niewielki wybór poza skalowaniem wzdłuż osi X.

W4 8. CONCURRENCY

Skalowanie bazujące na modelu procesu (4)



Aplikacja wykorzystująca kontenery Docker musi mieć pojedynczy proces lub proces główny z niektórymi procesami podrzędnymi. W takim przypadku, gdy zachodzi potrzeba skalowania, zaleca się uruchomienie większej ilości kontenerów (skalowanie wzdłuż osi X), a nie przypisywanie jej więcej zasobów (skalowanie wzdłuż osi Y). Nawet jeśli aplikacja uruchamia więcej procesów w kontenerze (tak jak robi to PHP-FPM),należy traktować kontener jako pojedynczą jednostkę, a proces główny powinien być w stanie bez problemów zamknąć procesy potomne po zakończeniu.

UWAGA: W podejściu typu full-stack, dana aplikacja może mieć różne punkty wejścia, np: sieć, pracownik, planista itd. Może to wydawać się zaprzeczeniem omawianej zasady, ale wtedy należy traktować każdą z tych funkcjonalności jako osobne zastosowanie tej samej aplikacji a tym samym stosować do niej zasadę CONCURENCY.

W4 9. DISPOSABILITY

Maksymalizacja niezawodność dzięki szybkiemu uruchamianiu i łagodnemu wyłączaniu (1)

Procesy aplikacji dwunastoczynnikowej są jednorazowe, co oznacza, że można je uruchomić lub zatrzymać w mgnieniu oka. Ułatwia to szybkie elastyczne skalowanie, szybkie wdrażanie zmian kodu lub konfiguracji oraz niezawodność wdrożeń produkcyjnych.

- Zasada DISPOSABILITY zakłada, że aplikacje powinny rozpoczynać się i kończyć z bez skutków ubocznych/zieoczekiwanych problemów. Oznacza to wykonanie wszystkich wymaganych czynności porządkowych, zanim aplikacja zostanie udostępniona konsumentom. Na przykład łagodne uruchomienie zapewni, że wszystkie połączenia z bazą danych i dostęp do innych zasobów sieciowych będą sprawne.

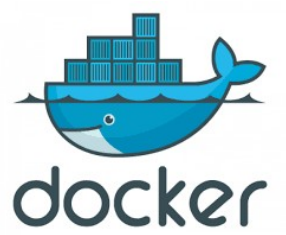
- Jeśli chodzi o wyłączanie, omawiane zalecenie zapewnia, że np. wszystkie połączenia z bazą danych i inne zasoby sieciowe są prawidłowo zakończone oraz że wszystkie działania związane z zamykaniem są rejestrowane,

W4 9. DISPOSABILITY

Maksymalizacja niezawodność dzięki szybkiemu uruchamianiu i łagodnemu wyłączaniu (2)

Procesy wewnątrz kontenera powinny być odporne na nieoczekiwane zakończenie, np. w przypadku awarii sprzętu. Chociaż jest to znacznie rzadsze zjawisko niż zamknięcie za pomocą SIGTERM, nadal może się zdarzyć. Zalecanym podejściem jest użycie niezawodnego mechanizmu kolejkowania, który zwraca zadania do kolejki, gdy klient się rozłączy lub przekroczy limit czasu. Tak czy inaczej, aplikacja zgodna z “12 Factors” musi być zaprojektowana do obsługi nieoczekiwanych zakończeń.

WNIOSEK: Proces PID 1 kontenera musi odpowiadać na sygnały POSIX. Oznacza to, że jeśli kontener otrzyma sygnał SIGTERM, powinien wdzięcznie się zamknąć. .



UWAGA: Należy pamiętać, aby zwolnić blokady (lub ustawić limit czasu), na wypadek gdyby coś poszło źle, a proces uruchomiony w kontenarze nieoczekiwanie przerwał działanie.

W4 10. DEV/PROD PARITY

Utrzymywanie jak najbardziej zbliżony posttaci etapu rozwoju, testowanie i produkcji (1)

Historycznie istniały znaczne luki między programowaniem (programista dokonujący edycji na żywo w lokalnym wdrożeniu aplikacji) a produkcją (działającym wdrożeniem aplikacji, do której uzyskują dostęp użytkownicy końcowi). Luki te przejawiają się w trzech obszarach:

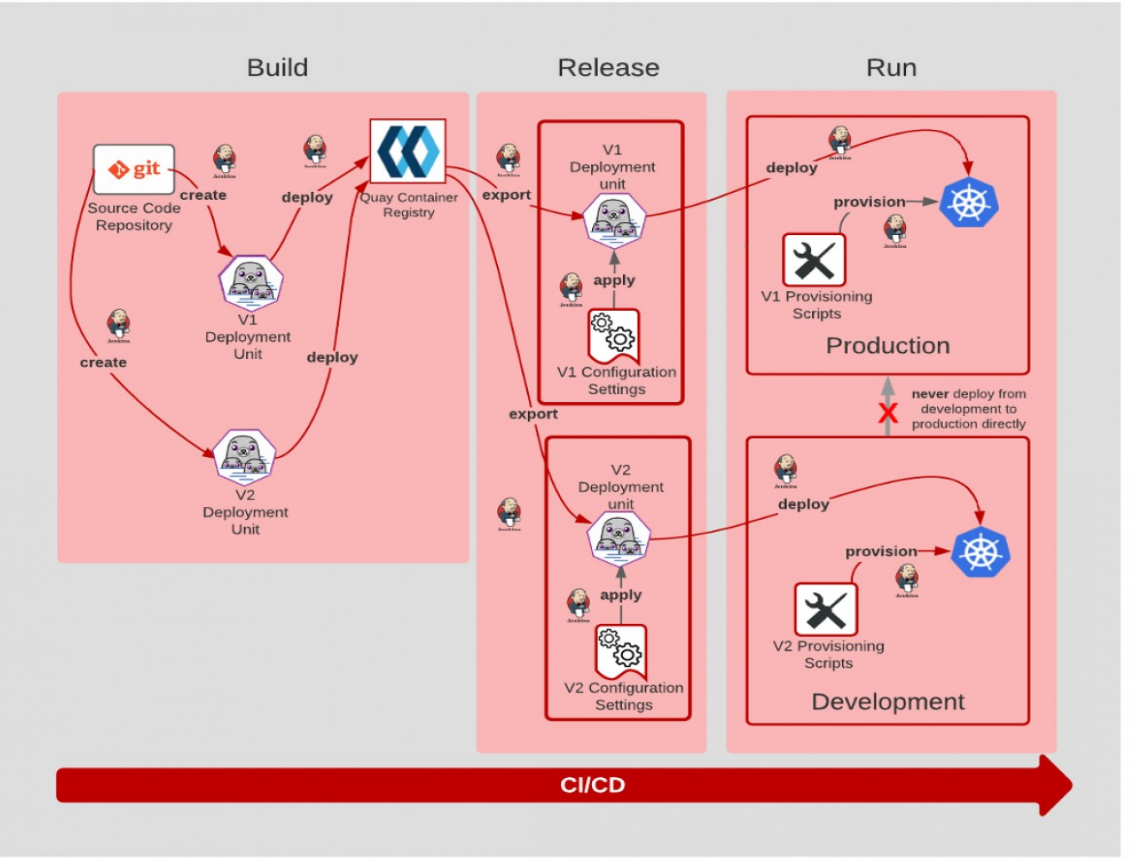
- Luka czasowa: programista może pracować nad kodem, którego wdrożenie do produkcji zajmuje dni, tygodnie, a nawet miesiące.
- Luka organizacyjna: Programista pisze kod aplikacji, inżynierowie z działów wdrażania przygotowują ją do etapu produkcji.
- Luka narzędziowa: Programista może wykorzystywać stos w postaci np. Nginx, SQLite oraz OS X, podczas gdy w produkcji użyty jest stos Apache, MySQL oraz Linux.

WNIOSEK: Zasada DEV/PROD PARITY oznacza, że wszystkie ścieżki wdrażania są podobne, ale niezależne i że żadne wdrożenie nie „przeskakuje” do innego miejsca docelowego wdrożenia.

W4 10. DEV/PROD PARITY

Utrzymywanie jak najbardziej zbliżony posttaci etapu rozwoju, testowanie i produkcji (2)

Rysunek przedstawia dwie wersje kodu aplikacji. Wersja V1 jest przeznaczona do wydania w środowisku produkcyjnym. Nowa wersja, V2, jest przeznaczona dla środowiska programistycznego. Zarówno wersja 1, jak i wersja 2 podążają podobną ścieżką wdrażania, od kompilacji do wydania, a następnie do uruchomienia. Jeśli wersja V2 kodu zostanie uznana za gotową do produkcji, dowiązania zewnętrzne i ustawienia dotyczące wersji 2 NIE MUSZĄ być skopiowane do środowiska produkcyjnego.

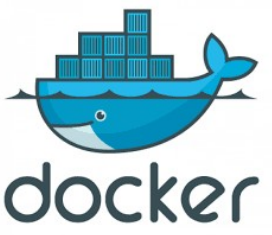


W4 10. DEV/PROD PARITY

Utrzymywanie jak najbardziej zbliżony posttaci etapu rozwoju, testowanie i produkcji (3)

WNIOSEK: Zalecenie Dev/Prod Parity jest bardzo podobne do zalecenia Build, Release, and Run. Ważnym rozróżnieniem jest to, że omawiane zalecenie zapewnia ten sam proces wdrażania dla produkcji, co w przypadku etapu rozwoju.

UWAGA: Programista przestrzegający zasad “12 Factors” powinien ograniczyć do niezbędnego minimum (najlepiej wykluczyć) korzystanie z różnych usług pomocniczych między procesem rozwoju a etapem produkcji. Ewentualne różnice między usługami tworzenia usług pomocniczych oznaczają, że pojawiają się drobne niezgodności, powodując problemy z kodem, który działał i przeszedł testy w fazie projektowania.



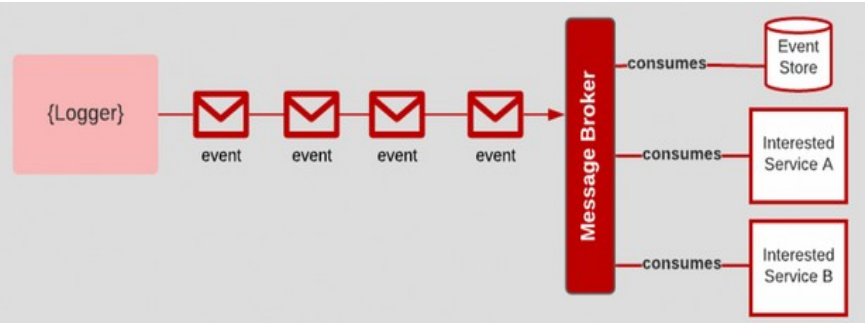
Ponieważ uruchamianie kontenerów jest niezależne od tego, gdzie opracowano obrazy, to zwykle gwarantuje to, że omawiane zalecenie jest spełnione.

W4 11. LOGS

Traktowanie log-ów jako strumienie zdarzeń (1)

- W tym zaleceniu log-i są traktowane jako to strumień zagregowanych, uporządkowanych w czasie zdarzeń zebranych ze strumieni wyjściowych wszystkich uruchomionych procesów i usług pomocniczych. Dzienniki w swojej surowej formie są zwykle w formacie tekstowym z jednym zdarzeniem na wiersz (chociaż wyjątkowo mogą obejmować wiele wierszy).

- Log-i nie mają ustalonego początku ani końca, ale przepływają w sposób ciągły, dopóki aplikacja działa.



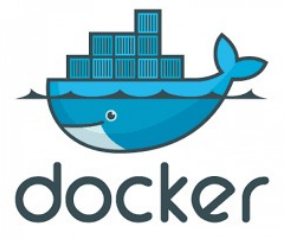
UWAGA: Proces routingu danych dziennika musi być oddzielony od przetwarzania danych dziennika. Na przykład jeden konsument może być zainteresowany tylko danymi błędu, podczas gdy inny konsument może być zainteresowany danymi żądania/odpowiedzi.

W4 11. LOGS

Traktowanie log-ów jako strumienie zdarzeń (2)

Strumień zdarzeń aplikacji może być kierowany do pliku lub oglądany w terminalu w czasie rzeczywistym. Co najważniejsze, strumień można wysłać do systemu indeksowania i analizy dzienników, np. systemu hurtowni danych, takiego jak Hadoop/Hive. Systemy te zapewniają dużą moc i elastyczność w zakresie introspekcji zachowania aplikacji w czasie, w tym:

- Znajdowanie konkretnych wydarzeń z przeszłości.
- Generowanie wizualizacji trendów na dużą skalę (takie jak żądania na minutę).
- Aktywne alertowanie zgodnie z heurystyką zdefiniowaną przez użytkownika (np. alert, gdy liczba błędów na minutę przekroczy określony próg).



W kontenerach nie wolno używać zapisu log-ów do pliku. Zamiast tego zalecane jest przesyłanie strumieniowe danych do stdout i stderr, które są dostępne dla wszystkich programów działających w systemach Unix .

W4 12. ADMIN PROCESSES

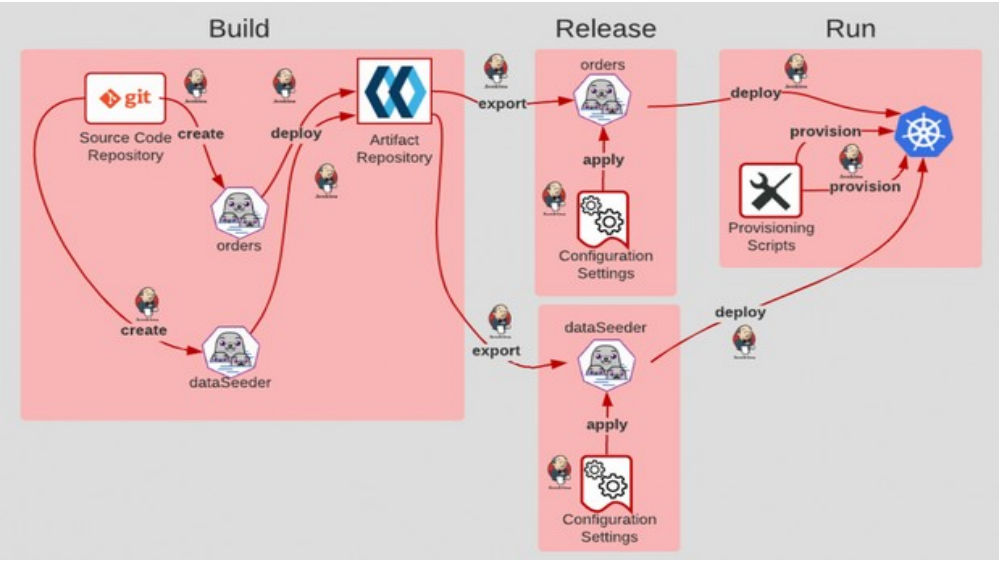
Uruchamianie zadań administracyjnych/zarządczych jako jednorazowe procesy (1)

The process formation is the array of processes that are used to do the app's regular business (such as handling web requests) as it runs. Separately, developers will often wish to do one-off administrative or maintenance tasks for the app, such as:
Running database migrations (e.g. manage.py syncdb in Django, rake db:migrate in Rails).
Running a console (also known as a REPL shell) to run arbitrary code or inspect the app's models against the live database. Most languages provide a REPL by running the interpreter without any arguments (e.g. python or erl) or in some cases have a separate command (e.g. irb for Ruby, rails console for Rails).
Running one-time scripts committed into the app's repo (e.g. php scripts fix_bad_records.php).
One-off admin processes should be run in an identical environment as the regular long-running processes of the app. They run against a release, using the same code and config as any process run against that release. Admin code must ship with application code to avoid synchronization issues.

W4 12. ADMIN PROCESSES

Uruchamianie zadań administracyjnych/zarządczych jako jednorazowe procesy (2)

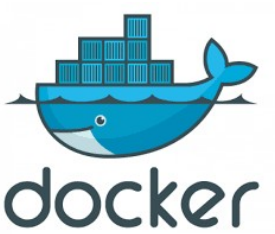
Rysunek po prawej przedstawia usługę o nazwie Orders, która jest wdrażana jako kontener platformy Docker. Ponadto istnieje nazwa usługi administracyjnej dataSeeder, która może umieszczać dane w usłudze Zamówienia. Usługa dataSeeder jest procesem administracyjnym, który ma być używany z Zamówieniami, jak pokazano na poniższym schemacie.



Jednak mimo że dataSeeder jest procesem administracyjnym, otrzymuje ścieżkę wdrażania BUILD-RELEASE-RUN podobną do usługi Zamówienia.

W4 12. ADMIN PROCESSES

Uruchamianie zadań administracyjnych/zarządczych jako jednorazowe procesy (3)



Omawiane zalecenie jest często I błędnie pomijane jeśli chodzi o kontenery Dockera. Kiedy uruchamiany jest kontener, zwykle następuje przypisanie logującego do użytkownika root wewnątrz kontenera.

Na przykład PHP-FPM podczas działania używa www-data użytkownika, nawet jeśli można uruchomić kontener jako root. Tak zabezpiecza się większość oficjalnych obrazów. Generalnie, należy upewnić się, że aplikacje nie są uruchamiane jako root, chyba że naprawdę jest to niebędne. Zaleca się zawsze korzystać z instrukcji USER w Dockerfile.

W4

PYTANIA / UWAGI ?