O'REILLY®

Projektowanie systemów rozproszonych

WZORCE I PARADYGMATY DLA SKALOWALNYCH, NIEZAWODNYCH USŁUG

Helion

Brendan Burns

Tytuł oryginału: Designing Distributed Systems:
Patterns and Paradigms for Scalable, Reliable Services

Tłumaczenie: Lech Lachowski

ISBN: 978-83-283-4738-0

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/prsyro
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

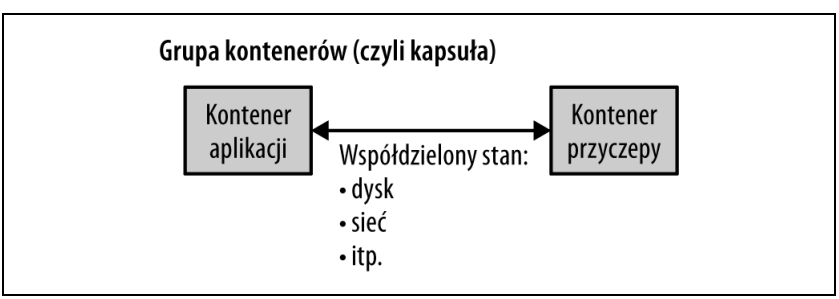
Printed in Poland.

Spis treści

ROZDZIAŁ 2.

Wzorzec Przyczepa

Pierwszym wzorcem jednowęzłowym jest Przyczepa (ang. *sidecar*). Przyczepa jest jednowęzłowym wzorcem składającym się z dwóch kontenerów. Pierwszym z nich jest **kontener aplikacji**. Zawiera podstawową logikę aplikacji. Bez tego kontenera aplikacja by nie istniała. Poza kontenerem aplikacji mamy również **kontener przyczepy**. Rolą przyczepy jest rozszerzanie i usprawnianie kontenera aplikacji, często bez jego wiedzy. W najprostszej formie kontener przyczepy może być używany w celu dodania funkcjonalności do kontenera, który byłoby trudno ulepszyć w inny sposób. Kontenery przyczepy są rozplanowywane na tej samej maszynie poprzez niepodzielną **grupę kontenerów**, taką jak obiekt API pod w Kubernetes. Poza rozplanowaniem na tej samej maszynie kontener aplikacji i kontener przyczepy współdzielą różne zasoby, w tym części systemu plików, nazwę hosta i sieć oraz wiele innych przestrzeni nazw. Ogólny wygląd wzorca Przyczepa jest pokazany na rysunku 2.1.

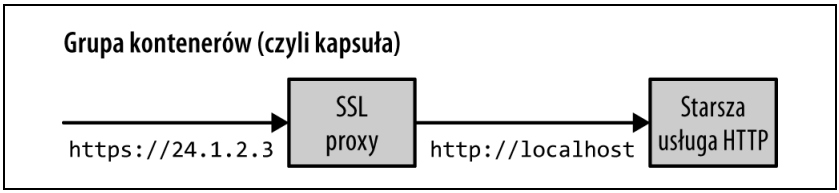


Rysunek 2.1. Ogólny wzorzec Przyczepa

Przykład przyczepy: dodawanie HTTPS do starszej usługi

Rozważmy przykład starszej usługi internetowej. Wiele lat temu, gdy budowano aplikację, bezpieczeństwo wewnętrznej sieci nie było dla firmy aż tak istotne, więc aplikacja obsługuje tylko żądania za pośrednictwem nieszyfrowanego protokołu HTTP, a nie HTTPS. Ze względu na ostatnie incydenty naruszenia bezpieczeństwa zarząd nakazał korzystanie z protokołu HTTPS na wszystkich stronach internetowych firmy. Aby dopełnić niedoli zespołu wyznaczonego do zaktualizowania tej konkretnej usługi sieciowej, kod źródłowy aplikacji został skompilowany za pomocą starej wersji firmowego systemu kompilacji, który nie jest już używany. Konteneryzacja tej aplikacji HTTP jest dość prosta: plik binarny może działać w kontenerze z jakąś starszą wersją dystrybucji Linuksa, bazując na nowocześniejszym jądrze uruchamianym przez orkiestrator kontenerów zastosowany przez zespół. Jednak dodanie do tej aplikacji obsługi protokołu HTTPS jest znacznie trudniejsze. Zespół dywagował nad wskrzeszeniem starego systemu kompilacji albo zaimportowaniem kodu źródłowego aplikacji do nowego systemu kompilacji, gdy jeden z członków zespołu zasugerował użycie wzorca Przyczepa, aby łatwiej rozwiązać ten problem.

Zastosowanie do tej sytuacji wzorca Przyczepa jest proste. Starsza usługa internetowa jest skonfigurowana do działania wyłącznie na adresie *localhost* (127.0.0.1), co oznacza, że dostęp do niej będą miały tylko te usługi, które współdzielą sieć lokalną z serwerem. Zwykle nie byłby to praktyczny wybór, ponieważ oznaczałby, że nikt nie może uzyskać dostępu do usługi internetowej. Jednak poprzez zastosowanie do starszego kontenera wzorca Przyczepa możemy dodać kontener przyczepy w postaci serwera nginx. Ten kontener nginx rezyduje w tej samej sieciowej przestrzeni nazw co starsza aplikacja internetowa, więc może uzyskać dostęp do usługi działającej na adresie *localhost*. Jednocześnie usługa nginx może na zewnętrznym adresie IP kapsuły i serwera proxy zatrzymać ruch HTTPS skierowany do starszej aplikacji internetowej (zobacz rysunek 2.2). Ponieważ ten nieszyfrowany ruch jest wysyłany tylko za pośrednictwem lokalnego adaptera pętli zwrotnej wewnątrz grupy kontenerów, zespół do spraw zabezpieczeń sieci jest przekonany, że dane są bezpieczne. W ten sposób, używając wzorca Przyczepa, nasz zespół zmodernizował starszą aplikację bez konieczności wymyślania sposobu na odbudowę nowej aplikacji do obsługi protokołu HTTPS.

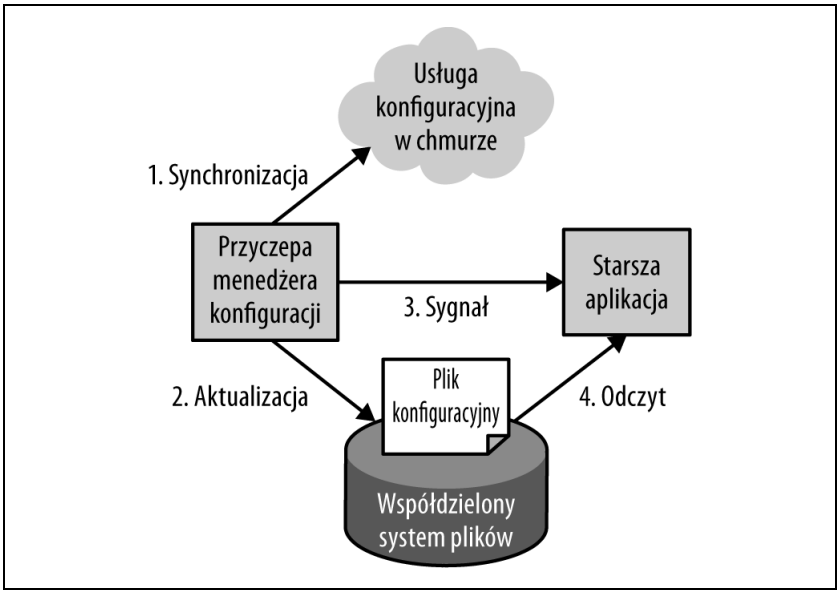


Rysunek 2.2. Przyczepa HTTPS

Dynamiczna konfiguracja za pomocą przyczepy

Zwykłe pośredniczenie w ruchu skierowanym do jakiejś istniejącej aplikacji nie jest jedynym zastosowaniem przyczepy. Innym typowym przykładem jest synchronizacja konfiguracji. Wiele aplikacji wykorzystuje do parametryzacji plik konfiguracyjny. Może to być nieprzetworzony plik tekstowy lub coś bardziej strukturalnego, jak XML, JSON lub YAML. Wiele wcześniejszych aplikacji napisano z założeniem, że ten plik jest obecny w systemie plików, a konfiguracja będzie odczytywana z danej lokalizacji. Jednak w środowisku natywnym dla chmury często dość przydatne jest użycie do aktualizacji konfiguracji interfejsu API. Pozwala to na dynamiczne wysyłanie informacji o konfiguracji za pośrednictwem API zamiast ręcznego logowania się do każdego serwera i aktualizowania pliku konfiguracyjnego za pomocą poleceń imperatywnych. Zapotrzebowanie na takie API wynika zarówno z łatwości użycia, jak i możliwości dodawania automatyzacji, takiej jak przywracanie poprzednich wersji, co sprawia, że konfigurowanie (oraz rekonfigurowanie) jest bezpieczniejsze i łatwiejsze.

Podobnie jak w przypadku protokołu HTTPS, nowe aplikacje można pisać z założeniem, że konfiguracja jest właściwością dynamiczną, która powinna być pozyskiwana za pomocą interfejsu API chmury, ale dostosowanie i aktualizacja istniejącej aplikacji mogą być znacznie większym wyzwaniem. Na szczęście ponownie można użyć wzorca Przyczepa w celu zapewnienia nowej funkcjonalności, która rozszerza starszą aplikację bez jej zmieniania. Wzorzec Przyczepa pokazany na rysunku 2.3 i tym razem ma dwa kontenery: kontener serwujący aplikację oraz kontener, który jest menedżerem konfiguracji. Te dwa kontenery są zgrupowane w kapsule, gdzie współdzielą jeden katalog. W tym wspólnym katalogu przechowywany jest plik konfiguracyjny.



Rysunek 2.3. Przykład przyczepy zarządzającej dynamiczną konfiguracją

Starsza aplikacja po uruchomieniu ładuje swoją konfigurację z systemu plików zgodnie z oczekiwaniami. Menedżer konfiguracji po uruchomieniu sprawdza interfejs API konfiguracji i szuka różnic między lokalnym systemem plików a konfiguracją przechowywaną w API. Jeśli występują różnice, menedżer konfiguracji pobiera nową konfigurację do lokalnego systemu plików i sygnalizuje starszej aplikacji, że powinna ją pobrać. Faktyczny mechanizm tego powiadomienia różni się w zależności od aplikacji. Niektóre aplikacje obserwują plik konfiguracyjny pod kątem zmian, podczas gdy inne reagują na sygnał `SIGHUP`. W skrajnych przypadkach menedżer konfiguracji może wysłać sygnał `SIGKILL`, aby przerwać działanie starszej aplikacji. Gdy aplikacja zostanie zatrzymana, system orkiestracji kontenerów uruchomi ją ponownie, a wtedy aplikacja załaduje swoją nową konfigurację. Podobnie jak w przypadku dodawania obsługi HTTPS do istniejącej aplikacji, ten wzorzec ilustruje, w jaki sposób przyczepa może pomóc w adaptacji istniejących aplikacji do bardziej natywnych dla chmury scenariuszy.

Modułowe kontenery aplikacji

Wybaczę Ci, jeśli na tym etapie uznałeś, że jedynym powodem istnienia wzorca Przyczepa jest dostosowywanie starszych aplikacji w sytuacjach, gdy nie chcemy już modyfikować oryginalnego kodu źródłowego. Chociaż jest to powszechny przypadek użycia tego wzorca, istnieje wiele innych powodów projektowania różnych rzeczy za pomocą przyczep. Innymi istotnymi zaletami korzystania z wzorca Przyczepa są modułowość oraz możliwość ponownego wykorzystania komponentów użytych jako przyczepy. W przypadku wdrażania dowolnej rzeczywistej, niezawodnej aplikacji wymagana jest pewna funkcjonalność umożliwiająca debugowanie lub wykonywanie innych czynności z zakresu zarządzania aplikacją, takich jak umożliwienie odczytu wszystkich procesów używających zasobów kontenera. Może to być coś na kształt narzędzia wiersza poleceń `top`.

Jednym ze sposobów zapewnienia tej introspekcji jest wymaganie, aby każdy programista implementował interfejs `/topz` HTTP, który zapewnia odczyt zużycia zasobów. Żeby było to łatwiejsze, mógłbyś zaimplementować ten webhook jako charakterystyczną dla danego języka wtyczkę, którą programista może po prostu podłączyć do aplikacji. Jednak nawet w takim przypadku programista byłby zmuszony podłączyć tę wtyczkę, a Twoja organizacja musiałaby zaimplementować interfejs dla każdego języka, który chce obsługiwać. Bez zastosowania ekstremalnej dyscypliny takie podejście niewątpliwie doprowadziłoby do wariacji w przypadku różnych języków oraz do braku wsparcia dla tej funkcjonalności w przypadku używania nowych języków. Zamiast tego funkcjonalność `topz` można wdrożyć jako kontener przyczepy, który współdzieli przestrzeń nazw identyfikatora procesu (ang. *process id*, PID) z kontenerem aplikacji. Taki kontener `topz` może obserwować wszystkie uruchomione procesy i zapewniać spójny interfejs użytkownika. Co więcej, możesz użyć systemu orkiestracji do automatycznego dodania tego kontenera do wszystkich aplikacji wdrożonych za pośrednictwem tego systemu orkiestracji, aby zapewnić spójny zestaw narzędzi dostępny dla wszystkich aplikacji działających w Twojej infrastrukturze.

Oczywiście, tak jak przy każdym wyborze technicznym, istnieją kompromisy między tym modularnym wzorcem opartym na kontenerach a przygotowywaniem własnego kodu dla aplikacji. Podejście oparte na bibliotekach zawsze będzie nieco mniej dostosowane do specyfiki Twojej aplikacji. Oznacza to, że

może być mniej wydajne lub interfejs API może wymagać adaptacji, aby pasował do Twojego środowiska. Porównałbym te kompromisy do różnicy między zakupem odzieży z sieciówki a ubieraniem się u krawca. Ubranie szyte na miarę zawsze będzie Ci pasować, ale poczekasz na nie dłużej i będzie Cię więcej kosztować. Podobnie jest w przypadku kodowania — większość z nas skłoni się raczej ku zakupieniu bardziej uniwersalnego rozwiązania. Oczywiście, jeśli Twoja aplikacja ma ekstremalne wymagania w kwestii wydajności, zawsze możesz wybrać rozwiązanie pisane ręcznie.

Część praktyczna: wdrażanie kontenera topz

Aby zobaczyć przyczepę `topz` w akcji, najpierw musisz wdrożyć inny kontener, który będzie działał jako kontener aplikacji. Wybierz jakąś istniejącą aplikację, z której korzystasz, i wdróż ją za pomocą Dockera:

```
$ docker run -d <obraz_aplikacji>
<wartość_skrótu_kotenera>
```

Po uruchomieniu obrazu otrzymasz identyfikator tego konkretnego kontenera. Będzie on wyglądać mniej więcej tak: `cccf82b85000…` Zawsze możesz też go sprawdzić, używając polecenia `docker ps`, które wyświetli wszystkie aktualnie uruchomione kontenery. Zakładając, że ukryłeś tę wartość w zmiennej środowiskowej o nazwie `APP_ID`, możesz uruchomić kontener `topz` w tej samej przestrzeni nazw PID w następujący sposób:

```
$ docker run --pid=container:${APP_ID} \
    -p 8080:8080 \
    brendanburns/topz:db0fa58 \
    /server --address=0.0.0.0:8080
```

Spowoduje to uruchomienie przyczepy `topz` jako kontenera aplikacji w tej samej przestrzeni nazw PID. Zwróć uwagę, że konieczna może być zmiana numeru portu używanego przez przyczepę do serwowania, jeśli kontener aplikacji również działa na porcie 8080. Po uruchomieniu przyczepy możesz wpisać w przeglądarce adres *http://localhost:8080/topz*, aby uzyskać pełny odczyt procesów uruchomionych w kontenerze aplikacji wraz z ich wykorzystaniem zasobów.

Możesz łączyć tę przyczepę z innym istniejącym kontenerem, aby za pośrednictwem interfejsu internetowego łatwo uzyskać wgląd w to, w jaki sposób ten kontener wykorzystuje swoje zasoby.

Budowanie prostej usługi PaaS za pomocą przyczepy

Wzorzec Przyczepa ma więcej zastosowań niż tylko dostosowywanie i monitorowanie. Może być również używany jako środek do zaimplementowania pełnej logiki aplikacji w uproszczony, modularny sposób. Wyobraź sobie na przykład budowanie prostej platformy jako usługi (ang. *platform as a service*, **PaaS**) wokół przepływu pracy `git`. Po wdrożeniu tej usługi PaaS wysyłanie nowego kodu do repozytorium Git powoduje jego wdrażanie na działających serwerach. Zobaczymy, w jaki sposób wzorzec Przyczepa upraszcza budowanie takiej usługi PaaS.

Jak już wspomniałem, we wzorcu Przyczepa znajdują się dwa kontenery: główny kontener aplikacji i przyczepa. W naszej prostej aplikacji PaaS głównym kontenerem jest serwer Node.js, który implementuje serwer WWW. Serwer Node.js jest instrumentowany w taki sposób, aby automatycznie przeładowywał serwer po aktualizacji nowych plików. Osiąga się to za pomocą narzędzia `nodemon` (*https://nodemon.io/*).

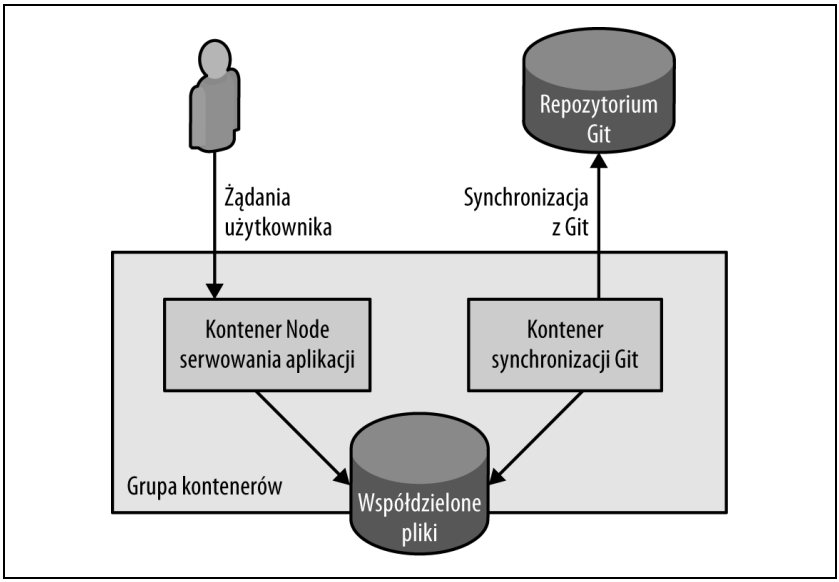
Kontener przyczepy współdzieli system plików z głównym kontenerem aplikacji i uruchamia prostą pętlę, która synchronizuje ten system plików z istniejącym repozytorium Git:

```
#!/bin/bash

while true; do
  git pull
  sleep 10
done
```

Oczywiście, ten skrypt może być bardziej złożony i pobierać z określonej gałęzi zamiast po prostu z `HEAD`. To uproszczenie jest celowe, aby zwiększyć czytelność przykładu.

Aplikacja Node.js i przyczepa synchronizacji Git są rozplanowane i wdrożone razem, aby zaimplementować naszą prostą usługę PaaS (zobacz rysunek 2.4). Po wdrożeniu przy każdorazowym przesłaniu nowego kodu do repozytorium Git kod ten jest automatycznie aktualizowany przez przyczepę i ponownie ładowany przez serwer.



Rysunek 2.4. Prosta usługa PaaS oparta na przyczepie

Projektowanie przyczep pod kątem modułowości i ponownego użycia

We wszystkich przykładach przyczep opisanych szczegółowo w tym rozdziale najważniejsze było to, żeby każdy z nich był modularnym artefaktem wielokrotnego użytku. Aby można było mówić o sukcesie, przyczepa powinna zapewniać możliwość wielokrotnego użytku w szerokim zakresie aplikacji i wdrożeń. Dzięki modułowej konstrukcji wielokrotnego użytku przyczepy mogą znacznie przyspieszyć budowanie aplikacji.

Ta modułowość i możliwość ponownego wykorzystania wymagają jednak skupienia i dyscypliny, podobnie jak uzyskiwanie modułowości podczas tworzenia wysokiej jakości oprogramowania. Musisz się skoncentrować w szczególności na rozwijaniu trzech obszarów:

- parametryzacji kontenerów,
- tworzeniu powierzchni API kontenera,
- dokumentowaniu działania kontenera.

Parametryzacja kontenerów

Parametryzowanie kontenerów jest najważniejszą rzeczą w kwestii zapewniania modułowości i możliwości wielokrotnego ich używania, niezależnie od tego, czy są one przyczepami, chociaż parametryzowanie przyczep i innych dodatkowych kontenerów jest szczególnie istotne.

Co rozumiem przez „parametryzowanie"? Potraktujmy kontener jako funkcję w programie. Ile ma ona parametrów? Każdy parametr reprezentuje dane wejściowe, które mogą dostosowywać ogólny kontener do konkretnej sytuacji. Przyjrzyj się na przykład wdrożonej poprzednio przyczepie dodatku SSL. Aby była ona użyteczna, prawdopodobnie będzie potrzebować co najmniej dwóch parametrów: nazwy certyfikatu używanego do zapewnienia SSL oraz portu serwera „starszej" aplikacji, działającego na adresie *localhost*. Bez tych parametrów trudno sobie wyobrazić, żeby ten kontener przyczepy był użyteczny w wielu aplikacjach. Podobne parametry istnieją dla wszystkich pozostałych przyczep opisanych w tym rozdziale.

Gdy już wiemy, jakich parametrów potrzebujemy, możemy się zastanowić, w jaki sposób będziemy udostępniać je użytkownikom i konsumować wewnątrz kontenera. Istnieją dwa sposoby przekazywania takich parametrów do kontenera: poprzez zmienne środowiskowe lub wiersz poleceń. Oba sposoby są prawidłowe, ale ja z reguły przekazuję parametry za pomocą zmiennych środowiskowych. Poniżej przykład przekazywania takich parametrów do kontenera przyczepy:

```
docker run -e=PORT=<port> -d <obraz>
```

Oczywiście, dostarczanie wartości do kontenera to tylko część zadania. Kolejną kwestią jest używanie tych zmiennych wewnątrz kontenera. Zazwyczaj do tego celu wykorzystywany jest zwykły skrypt powłoki, który ładuje zmienne środowiskowe dostarczone z kontenerem przyczepy i dostosowuje pliki konfiguracyjne lub parametryzuje bazową aplikację.

Jako zmienne środowiskowe można na przykład przekazać ścieżkę do certyfikatu i port:

```
docker run -e=PROXY_PORT=8080 -e=CERTIFICATE_PATH=/ścieżka/do/cert.crt ...
```

W kontenerze użylibyśmy tych zmiennych do skonfigurowania pliku *nginx.conf*, który wskazuje serwerowi WWW prawidłową lokalizację pliku i serwera proxy.

Definiowanie API każdego kontenera

Biorąc pod uwagę, że parametryzujemy kontenery, oczywiste jest, że definiują one „funkcję" wywoływaną za każdym razem, gdy dany kontener jest wykonywany. Ta funkcja jest częścią API zdefiniowanego przez kontener, ale istnieją również inne części API, w tym wywołania wykonywane przez kontener do innych usług, jak również tradycyjne HTTP lub inne interfejsy API dostarczane przez ten kontener.

Przy definiowaniu modułowych kontenerów wielokrotnego użytku trzeba zdawać sobie sprawę, że wszystkie aspekty interakcji kontenera ze światem są częścią interfejsu API zdefiniowanego przez ten kontener wielokrotnego użytku. Podobnie jak w świecie mikrousług, te **mikrokontenery** opierają się na interfejsach API, aby zapewnić czystą separację kontenera aplikacji i przyczepy. Dodatkowo celem API jest zagwarantowanie prawidłowego działania wszystkich konsumentów przyczepy wraz z pojawianiem się kolejnych wersji. Posiadanie czystego API dla przyczepy zapewnia także szybszą pracę programistów, ponieważ mają jasną definicję (i, miejmy nadzieję, testy jednostkowe) dla usług dostarczanych w ramach przyczepy.

Konkretnym przykładem wagi powierzchni interfejsu API jest omówiona wcześniej przyczepa zarządzania konfiguracją. Przydatną konfiguracją dla tej przyczepy może być parametr `UPDATE_FREQUENCY` (częstotliwość aktualizacji), który wskazuje, jak często konfiguracja powinna być synchronizowana z systemem plików. Oczywiste jest, że jeśli w jakimś późniejszym czasie nazwa parametru zostanie zmieniona na `UPDATE_PERIOD` (interwał aktualizacji), ta zmiana będzie niezgodna z API przyczepy i z pewnością zakłóci jej działanie dla niektórych użytkowników.

Chociaż ten przykład jest oczywisty, nawet subtelniejsze zmiany mogą być niezgodne z interfejsem API przyczepy. Wyobraźmy sobie, że parametr `UPDATE_FREQUENCY` początkowo pobierał wartość w sekundach. W miarę upływu czasu i na podstawie informacji zwrotnych od użytkowników programista przyczepy doszedł do wniosku, że podawanie wartości w sekundach dla dłuższych przedziałów czasu (na przykład minut) było denerwujące. Zmienił więc ten parametr, aby przyjmował łańcuch znaków (10 min, 5 s itp.). Ponieważ stare wartości parametrów (na przykład wartość 10 dla 10 sekund) nie będą parsowane w tym nowym schemacie, jest to zmiana naruszająca interfejs API. Załóżmy jednak, że programista to przewidział, ale ustalił, że wartości bez

jednostek będą parsowane na milisekundy, podczas gdy wcześniej były parsowane na sekundy. Nawet ta zmiana, chociaż nie prowadzi do błędu, stanowi naruszenie interfejsu API przyczepy, gdyż prowadzi do znacznie częstszych kontroli konfiguracji i odpowiednio częstszego ładowania danych na serwer konfiguracji w chmurze.

Mam nadzieję, że te rozważania przekonały Cię, iż w celu zapewnienia prawdziwej modułowości trzeba być bardzo świadomym API dostarczanego przez przyczepę, a zmiany naruszające to API nie zawsze będą tak oczywiste jak zmiana nazwy parametru.

Dokumentowanie kontenerów

Nauczyłeś się już parametryzować kontenery przyczepy w taki sposób, aby były modułowe i wielokrotnego użytku. Poznałeś znaczenie utrzymywania stabilnego API w celu zapewnienia użytkownikom nieprzerwanego prawidłowego działania przyczep. Jest jeszcze jeden krok na drodze do budowania modułowych kontenerów wielokrotnego użytku: zagwarantowanie użytkownikom możliwości korzystania z nich.

Podobnie jak w przypadku bibliotek oprogramowania, kluczem do zbudowania czegoś naprawdę użytecznego jest wyjaśnienie, jak tego używać. Budowanie elastycznego, niezawodnego kontenera modułowego jest mało przydatne, jeśli nikt nie wie, jak z niego korzystać. Niestety nie ma zbyt wielu formalnych narzędzi do dokumentowania obrazów kontenerów, ale istnieje kilka dobrych praktyk, które można zastosować.

W przypadku każdego obrazu kontenera najbardziej oczywistym miejscem szukania dokumentacji jest plik *Dockerfile*, na podstawie którego kontener został zbudowany. Niektóre części pliku *Dockerfile* już dokumentują sposób działania kontenera. Jednym z przykładów jest dyrektywa `EXPOSE` wskazująca porty, na których obraz nasłuchuje. Chociaż dyrektywa `EXPOSE` nie jest konieczna, dobrą praktyką jest umieszczenie jej w pliku *Dockerfile* oraz dodanie komentarza wyjaśniającego, co dokładnie nasłuchuje na tym porcie, na przykład:

```
...
# Główny serwer WWW działa na porcie 8080
EXPOSE 8080
...
```

Ponadto jeśli do sparametryzowania kontenera używasz zmiennych środowiskowych, możesz skorzystać z dyrektywy `ENV`, aby ustawić wartości domyślne dla tych parametrów i udokumentować ich użycie:

```
...
# Parametr PROXY_PORT wskazuje port na adresie localhost, do którego ma być
przekierowany ruch.
ENV PROXY_PORT 8000
...
```

Należy także zawsze używać dyrektywy `LABEL` w celu dodawania do obrazu metadanych, takich jak adres e-mail opiekuna, strona internetowa i wersja obrazu:

```
...
LABEL "org.label-schema.vendor"="nazwisko@firma.com"
LABEL "org.label.url"="http://obrazy.firma.com/mój_fajny_obraz"
LABEL "org.label-schema.version"="1.0.3"
...
```

Nazwy tych etykiet pochodzą ze schematu ustalonego przez projekt Label Schema (*http://label-schema.org/rc1/*). W ramach projektu powstaje wspólny zestaw dobrze znanych etykiet. Dzięki zastosowaniu wspólnej systematyki etykiet obrazów wiele różnych narzędzi może polegać na tych samych metainformacjach, aby wizualizować, monitorować i prawidłowo wykorzystywać aplikację. Przyjmując wspólne pojęcia, można używać zestawu narzędzi opracowanych przez społeczność bez modyfikowania obrazu. Oczywiście, możesz również wstawić różne dodatkowe etykiety, które mają sens w kontekście Twojego obrazu.

Podsumowanie

W tym rozdziale wprowadziłem wzorzec Przyczepa, który służy do łączenia kontenerów na pojedynczej maszynie. W tym wzorcu kontener przyczepy rozszerza kontener aplikacji, aby dodać określoną funkcjonalność. Przyczepy można wykorzystywać do aktualizowania istniejących starszych aplikacji, gdy ich zmiana jest zbyt kosztowna. Ponadto można je stosować do tworzenia modułowych kontenerów narzędziowych, które standaryzują implementacje typowych funkcjonalności. Takie kontenery narzędziowe mogą być ponownie wykorzystywane w dużej liczbie aplikacji, gdyż zwiększają spójność i obniżają koszty opracowania każdej z nich. Następne rozdziały wprowadzają kolejne wzorce jednowęzłowe, które demonstrują inne zastosowania dla modułowych kontenerów wielokrotnego użytku.


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

O'REILLY®

Twórz systemy rozproszone!

Nowoczesne oprogramowanie musi sprostać wyśrubowanym wymaganiom: ma cechować się określoną niezawodnością i skalowalnością, a przy tym powinno korzystać z technologii chmury. Dziś standardem jest używanie aplikacji na wielu urządzeniach w różnych lokalizacjach. Niestety, mimo powszechności systemów rozproszonych ich projektowanie nader często przypomina coś w rodzaju czarnej magii, dostępnej dla nielicznych wtajemniczonych.

Ta książka jest praktycznym przewodnikiem dla projektantów systemów rozproszonych. Zaprezentowano tu kolekcję powtarzalnych wzorców oraz zalecanych praktyk programistycznych, dzięki którym rozwijanie niezawodnych systemów rozproszonych stanie się bardziej przystępne i wydajne. Poza podstawowymi wzorcami przedstawiono również techniki tworzenia skonteneryzowanych komponentów wielokrotnego użytku. Znalazło się tu także omówienie zagadnień rozwoju kontenerów i orkiestratorów kontenerów, które zasadniczo zmieniły sposób budowania systemów rozproszonych.

Najważniejsze zagadnienia:

- wprowadzenie do systemów rozproszonych
- znaczenie wzorców i komponentów wielokrotnego użytku
- jednowęzłowe wzorce: Przyczepa, Adapter i Ambasador
- wielowęzłowe wzorce dla replikowania, skalowania i wybierania węzłów głównych

Dr Brendan Burns specjalizuje się w projektowaniu dużych aplikacji i programowaniu obliczeń w chmurze. Jest też współzałożycielem projektu open source Kubernetes. Obecnie pracuje w Microsofcie, gdzie zajmuje się platformą Azure, natomiast wcześniej pracował w Google Cloud Platform. Kiedyś zajmował się również infrastrukturą wyszukiwarek internetowych Google.

ISBN 978-83-283-4738-0
9 788328 347380

Cena: 39,90 zł