Aplikacje w chmurze (Cloud-Native Application)

Wzorce projektowe

Autorzy:
Rafał Franczyk
Sebastian Zając
Damian Szczepański
Maciej Wadowski
Patryk Zych
Dawid Karaś

Koncept Aplikacji Chmurowych

- Skalowalność i dostępność
- Pakowane w kontenery
- Zarządzany dynamicznie
- Zorientowane na mikrousługi

Cele aplikacji chmurowych

Głównymi celami które próbują osiągnąć aplikacje wdrażane w chmurze są:

- Dostępność
- Zarządzanie danymi
- Komunikacja poprzez wiadomości
- Zarządzanie i monitoring
- Wydajność i skalowalność
- Odporność na awarie
- Bezpieczeństwo

Fałszywe założenia podczas projektowania aplikacji chmurowych

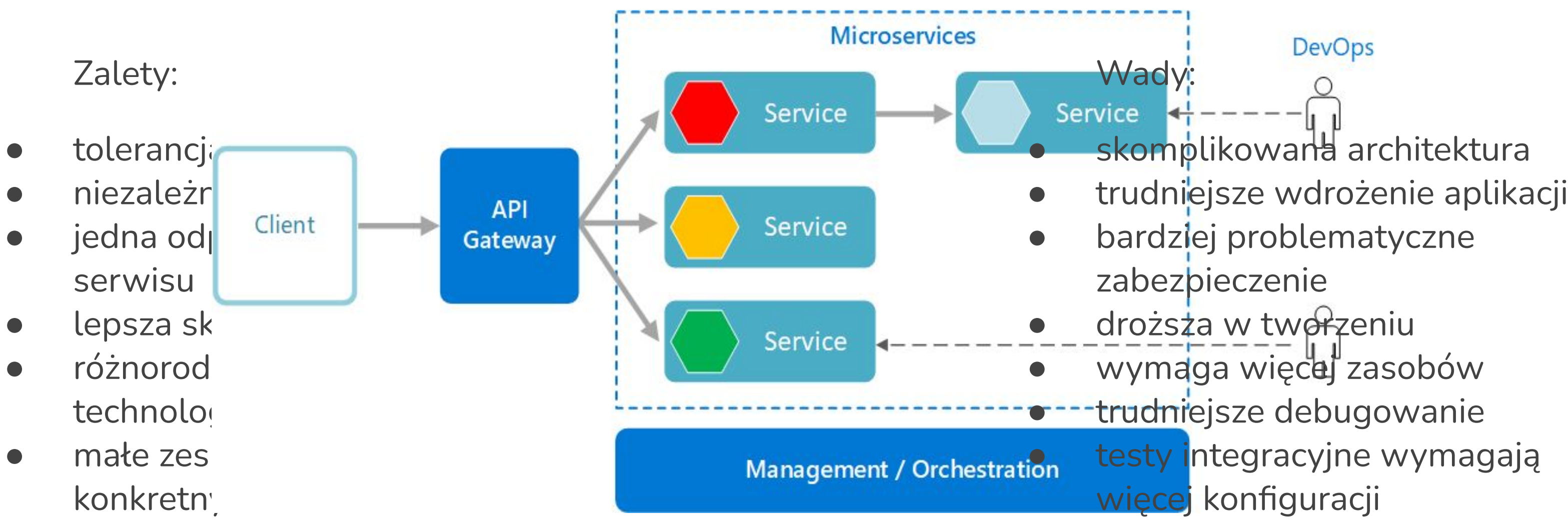
- Sieć jest niezawodna
- Opóźnienia nie istnieją
- Nieskończona przepustowość
- Sieć jest bezpieczna
- Topologia sieci się nie zmienia
- Istnieje jeden administrator
- Koszt przesyłu danych wynosi 0
- Sieć jest jednorodna

Wzorce projektowania chmury

Composite application (microservices)

Zalety:

- tolerancj
- niezależn
- jedna od
  serwisu
- lepsza sk
- różnorod
  technolo
- małe zes
  konkretn



Wady:

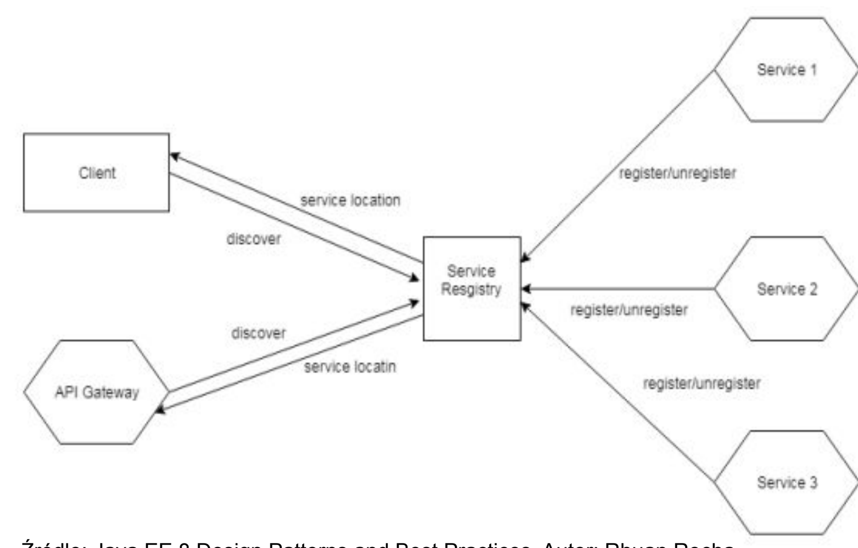
- skomplikowana architektura
- trudniejsze wdrożenie aplikacji
- bardziej problematyczne zabezpieczenie
- droższa w tworzeniu
- wymaga więcej zasobów
- trudniejsze debugowanie
- testy integracyjne wymagają więcej konfiguracji

Źródło: https://docs.microsoft.com/en-us/azure/architecture/includes/images/microservices-logical.png

Service registry

Adresy IP kontenerów i maszyn wirtualnych są dynamiczne i często ulegają zmianie. W konsekwencji lokalizacje serwisów rezydujących w kontenerach również ulegają zmianie. Jak wiadomo instancje mikroserwisów są tworzone i usuwane na bieżąco. Powstaje więc pytanie jak klienty, korzystające z mikroserwisów mogą poradzić sobie z tym problemem? Z pomocą przychodzi wzorzec projektowy Service Registry.

Service Registry jest bazą danych wszystkich tworzonych serwisów. Po utworzeniu, mikroserwisy są zapisywane do tej bazy danych (register) natomiast po ich wyłączeniu są z niej usuwane (unregister). Aplikacje klienckie uzyskują dostęp do Service Registry, który odpowiedzialny jest za sprawdzenie, czy mikroserwis jest dostępny oraz za udostępnienie jego lokalizacji klientowi.



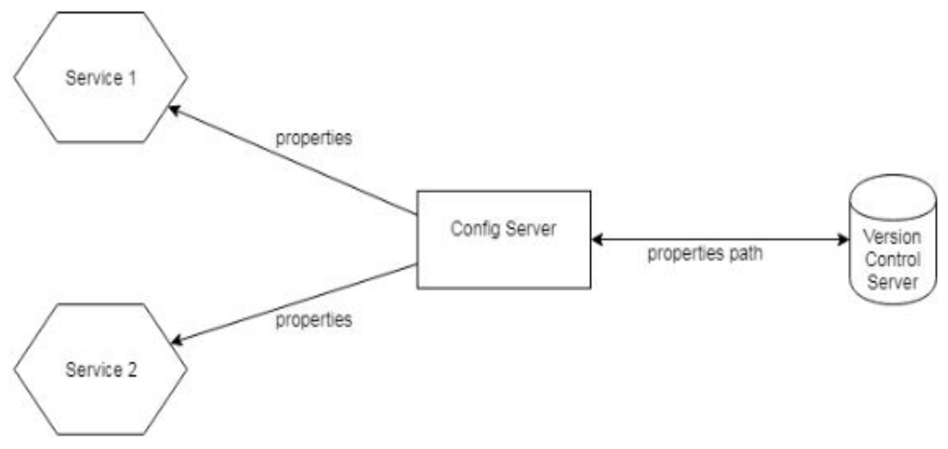
Źródło: Java EE 8 Design Patterns and Best Practices, Autor: Rhuan Rocha

Abstraction

- nastawienie na potrzeby klienta, a nie obecną sprzętową architekturę
- traktowanie warstwy sprzętowej w sposób abstrakcyjny
- wysoka skalowalność warstwy sprzętowej, użycie zgodne z zapotrzebowaniem

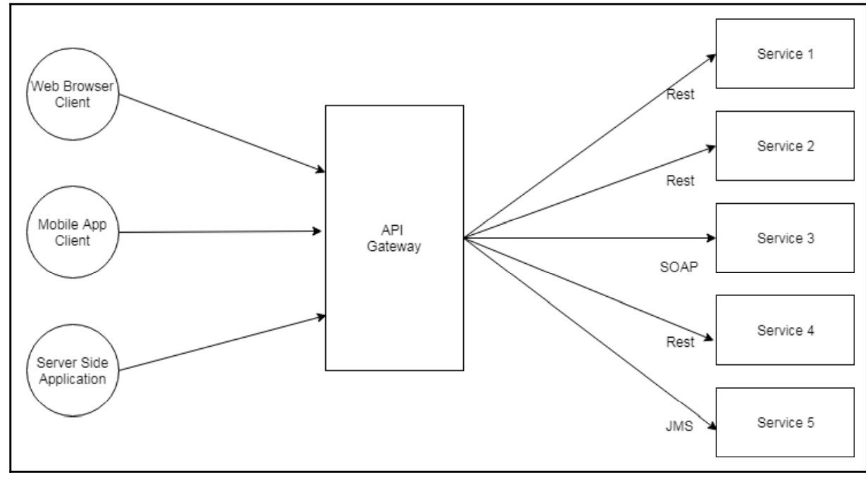
Config server

Config server jest warstwą odpowiedzialną za dostarczanie konfiguracji do mikroserwisów oraz przechowywanie konfiguracji. Pozwala ona pozbyć się problemów związanych z koniecznością restartowania kontenera po zmianie którejś z danych konfiguracyjnych np. hasła do bazy danych. Po starcie Config Server’a dane konfiguracyjne mikroserwisów pobierane są ze ścieżki określonej w systemie kontroli wersji.



Źródło: Java EE 8 Design Patterns and Best Practices, Autor: Rhuan Rocha

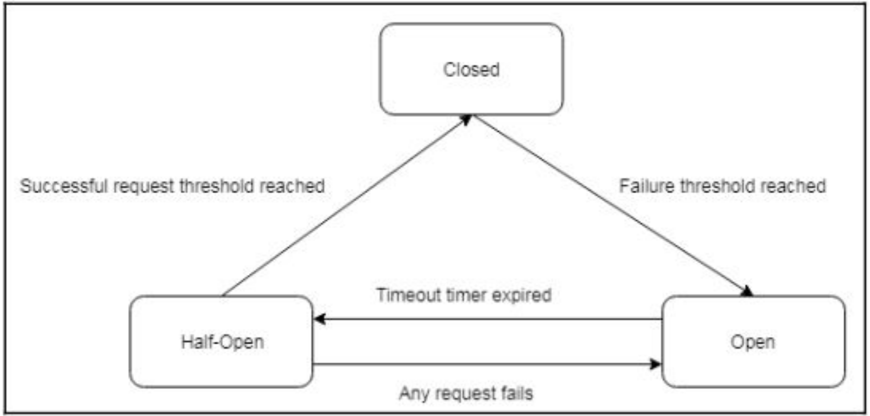
Api Gateway



Źródło: Java EE 8 Design Patterns and Best Practices, Autor: Rhuan Rocha

Api Gateway służy jako tzw. “frontend” dla klientów chmurowych. W skrócie jest to publiczne API, dzięki któremu jednym zapytaniem użytkownicy mogą wykonywać bardziej skomplikowane operacje, mimo że wymaga to komunikacji z wieloma aplikacjami. Można na ten wzorzec spojrzeć jak na proxy, które przyjmuje żądanie i wysyła odpowiednie żądania/żądanie do właściwych serwisów. Czasami stworzonych API jest więcej w zależności od typu klienta (np. osobne dla urządzeń mobilnych i przeglądarkowych).

Circuit-Breaker



Źródło: Java EE 8 Design Patterns and Best Practices, Autor: Rhuan Rocha

Mechanizm ma nazwę pochodzącą od obwodu elektrycznego, ponieważ działa w bardzo podobny sposób. Sam wzorzec jest jak proxy, które decyduje czy dane zapytanie ma się przetworzyć. Wzorzec ten w przypadku wykrycia błędów z zapytaniami stara się unormować sytuację, aniżeli pozwalać na dalsze zalewanie serwisu zapytaniami, na które serwis nie odpowie lub odpowie po bardzo długim czasie.

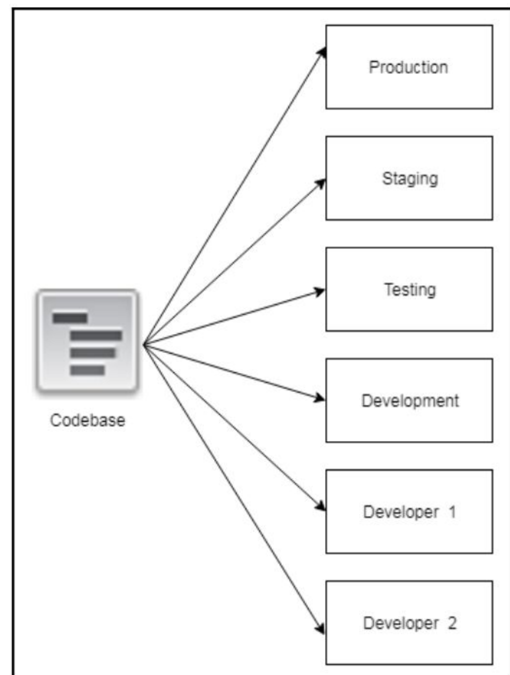
Wyróżnia on 3 tryby działania:

- Closed - serwis działa prawidłowo i zapytania są przekazywane do serwisu. Jeżeli napotka timeout, to licznik niepowodzeń się zwiększa. W momencie gdy licznik przekroczy pewien limit, stan zamienia się na half-open.
- Open - serwis automatycznie na każde kolejne zapytanie zwraca wyjątek.
- Half-Open - ogranicza liczbę przekazywanych zapytań do serwisu. Jeżeli kończą się powodzeniem, to licznik się zeruje i nasz obwód przechodzi w stan closed. W przypadku gdy jakiekolwiek zapytanie się nie powiedzie, obwód przechodzi na stan open na jakiś określony czas.

Twelve factory

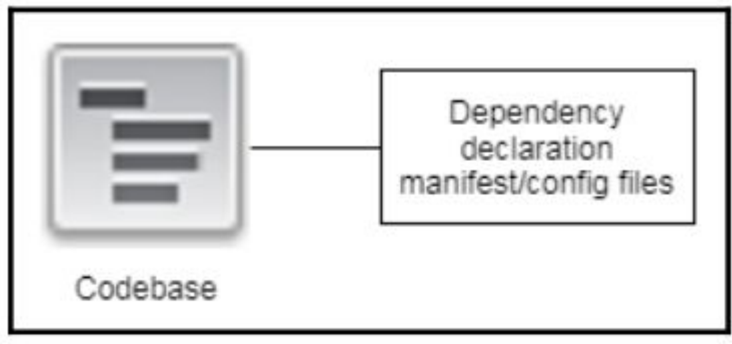
Codebase

- jedno repozytorium na aplikacje
- części wspólne kodu jako biblioteki
- repozytorium musi być zarządzane systemem kontroli wersji np. Git
- zaletą tego podejścia jest możliwość prostego budowania aplikacji w wersji deweloperskiej, przejściowej lub produkcyjnej



Źródło: Java EE 8 Design Patterns and Best Practices
Autor: Rhuan Rocha

Dependencies



Źródło: Java EE 8 Design Patterns and Best Practices, Autor: Rhuan Rocha

- zależności muszą być zadeklarowane i odizolowane od kodu
- w praktyce wiąże się to z korzystaniem z narzędzi do zarządzania projektem np. Maven, Gradle, SBT
- dla przykładu w Mavenie jest plik pom.xml gdzie deklaruje się zależności projektu

Config

Pliki konfiguracyjne zawierają m.in:

- dane dostępu do bazy danych (adres, port, dane logowania oraz schemat)
- ustawienia cache’owania pamięci
- dane dostępu do kolejek (adres, port oraz dane logowania)

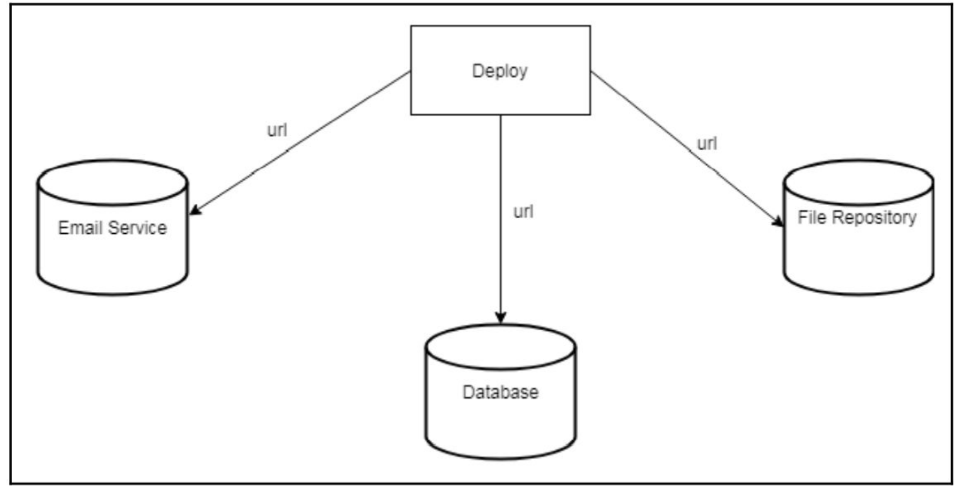
Jego charakterystyka to głównie:

- separacja konfiguracji od kodu - kod pozostaje niezmienny dla różnych środowisk w porównaniu do konfiguracji
- zalecane wstrzykiwanie konfiguracji w zależności od środowiska
- pliki konfiguracyjne wymagają odpowiedniego zabezpieczenia

Codebase
config
Production
config
Staging
config
Testing
config
Development
config
Developer 1
config
Developer 2

Źródło: Java EE 8 Design Patterns and Best Practices, Autor: Rhuan Rocha

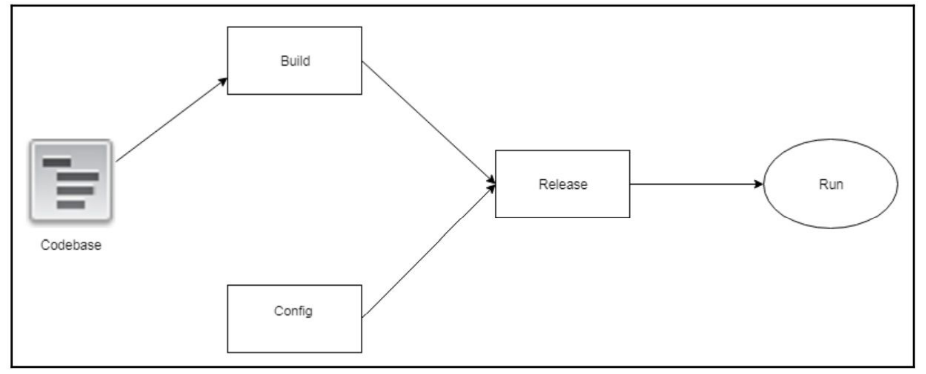
Backing services



Źródło: Java EE 8 Design Patterns and Best Practices, Autor: Rhuan Rocha

- Są to dodatkowe serwisy używane przez aplikację takie jak baza danych, serwis komunikatów, repozytorium plików czy serwis email.
- Każdy z nich jest postrzegany jako zasób przez twelve-factor.
- Każda aplikacja musi być dostępna poprzez dane podane w konfiguracji (adres URL lub lokalizacja). Dzięki temu zmiana lokalizacji serwisu nie wymaga zmian w kodzie.

Build, Release, Run



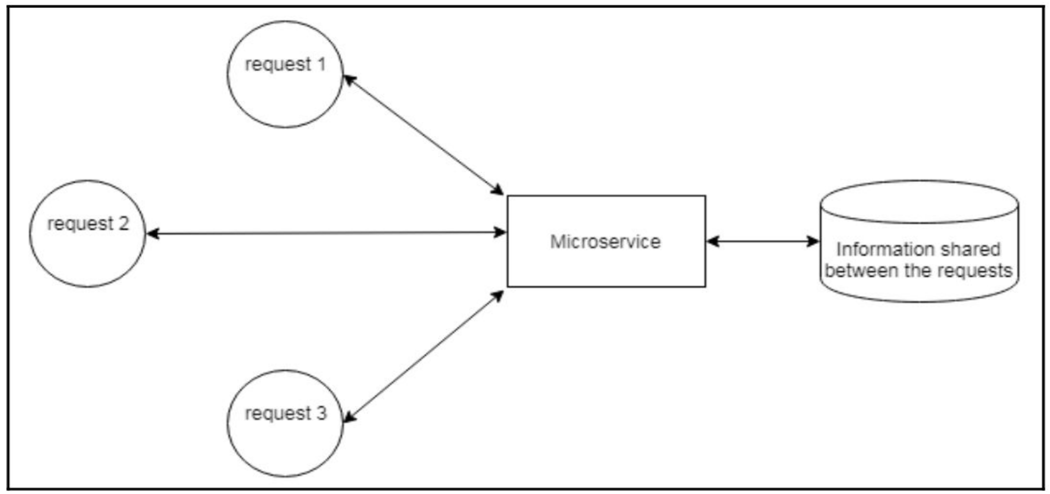
Źródło: Java EE 8 Design Patterns and Best Practices, Autor: Rhuan Rocha

Proces transformacji kodu, do danego środowiska powinien zostać podzielony na trzy etapy:

- **Build** - kompilacja i pakowanie kodu źródłowego do postaci wykonywalnej np. WAR, EAR.
- **Release** - moment wstrzykiwania konfiguracji do postaci wykonywalnej. W rezultacie mamy wykonywalną aplikację wraz z konfiguracją dla danego środowiska.
- **Run** - inicjalizacja aplikacji na danym środowisku.

Cały ten proces ułatwia zarządzanie i automatyzację procesu wdrażania danej aplikacji.

Processes



Źródło: Java EE 8 Design Patterns and Best Practices, Autor: Rhuan Rocha

Metodologia twelve-factor mówi jasno, że każdy proces ma być bezstanowy i nie powinien przechowywać informacji. Jeżeli są jakieś informacje, które chcielibyśmy zachować, powinniśmy skorzystać z bazy danych. Taka technika pozwala zapobiec problemom związanym z przetwarzaniem równoległym, gdzie otrzymywane zapytania mogą być przetwarzane przez inny proces lub nawet serwer.

Port binding

- brak zależności aplikacji od zewnętrznych serwerów, np. Tomcat, Apache
- dostęp do aplikacji za pomocą URL i przydzielonego portu
- każdy serwis może działać jako serwis zastępczy

Concurrency

- możliwość skalowania aplikacji poprzez uruchamianie wielu instancji serwisów
- aplikacja musi być stateless
- szybsze uruchamianie aplikacji dzięki konteneryzacji

Disposability

- serwisy powinny móc wystartować i zatrzymać się w każdej chwili
- zatrzymanie procesu nie powinno mieć żadnego wpływu na działanie całej aplikacji, powinno zwolnić zasoby, w razie konieczności powinien być zapisany stan procesu

Dev/prod parity

Wszystkie środowiska tj. testowe, deweloperskie i produkcyjne powinny być jak najbardziej do siebie zbliżone. Pozwala to uniknąć błędów w trakcie przenoszenia zmian z środowisk testowych na produkcyjne

Logs

W tradycyjnym środowisku, logi mogą być przechowywane w plikach. Jednakże mogą pojawić się problemy takie jak brak miejsca na dysku.

Pliki z logami mogą zostać utracone podczas zmiany rozmiaru w chmurze. Aby temu zapobiec należy traktować logi jako strumień zdarzeń. Dzięki temu logi możemy przechowywać w dowolnym miejscu.

Admin processes

Wszelkie migracje danych, skrypty inicjalizujące, czyszczenie cache itp. powinny być zautomatyzowane ze względu na używanie różnych środowisk. Pozwala to na uniknięcie różnic między nimi i zmniejszenie ilości potencjalnych błędów.

Dziękujemy za uwagę!