Wzorce Wdrożeniowe i Operacyjne

Michał Sitko Katarzyna Surzyn Jakub Piątek Patryk Świerkowski Krzysztof Rokosz

Wzorce Wdrożeniowe

Czyli rodzice CI/CD

Po co te całe wzorce?

- Co jeśli w wewnętrznym środowisku testowym wszystko działało a na produkcji nowa wersja zaczyna się sypać?
- Co jeżeli nowa wersja nie przypadła klientom do gustu i trzeba robić rollback aplikacji na wszystkich serwerach produkcyjnych?

<==

Stosowanie dobrych wzorców wdrożeniowych pozwala w większości uniknąć tych przypadków.

Canary Server

O jaki ładny kanarek! `Segmentation fault (core dumped)` O nie...

- Nowa wersja aplikacji zostaje uruchomiona na pewnym procencie serwerów produkcyjnych
- Użytkownicy są przekierowywani na serwery ze starą wersją aplikacji
- Nowa wersja jest testowana przez automatyczne testy, testerów i chętnych użytkowników. Aplikacja jest obserwowana pod kątem wydajności, niezawodności i współpracy z istniejącymi systemami
- Nowa wersja jest propagowana na pozostałe serwery lub robiony jest rollback serwerów cannary w zależności od wyników testów

Wdrożenie za pomocą serwerów canary pozwala na przetestowanie nowej wersji aplikacji w środowisku produkcyjnym bez zaburzania dostępu do starej wersji.

Testerzy i chętni użytkownicy mogą korzystać z serwerów cannary (beta) do oceny nowej wersji.

Jeśli nowa wersja się sprawdzi ruch jest zamieniany i wszyscy użytkownicy są przekierowywani na serwery canary aż reszta serwerów zostanie zaktualizowana


Użytkownicy
Testerzy i Power Users
Proxy
Serwery ze starą wersją
Serwer 1
Serwer 2
Serwer 3
Serwery Canary (Beta)
Serwer 4
Serwer 5
Serwer 6
Użytkownicy
Testerzy i Power Users
Proxy
Serwery ze starą wersją
Serwer 1
Serwer 2
Serwer 3
Serwery Canary (Nowa Wersja)
Serwer 4
Serwer 5
Serwer 6
Deploy nowej wersji

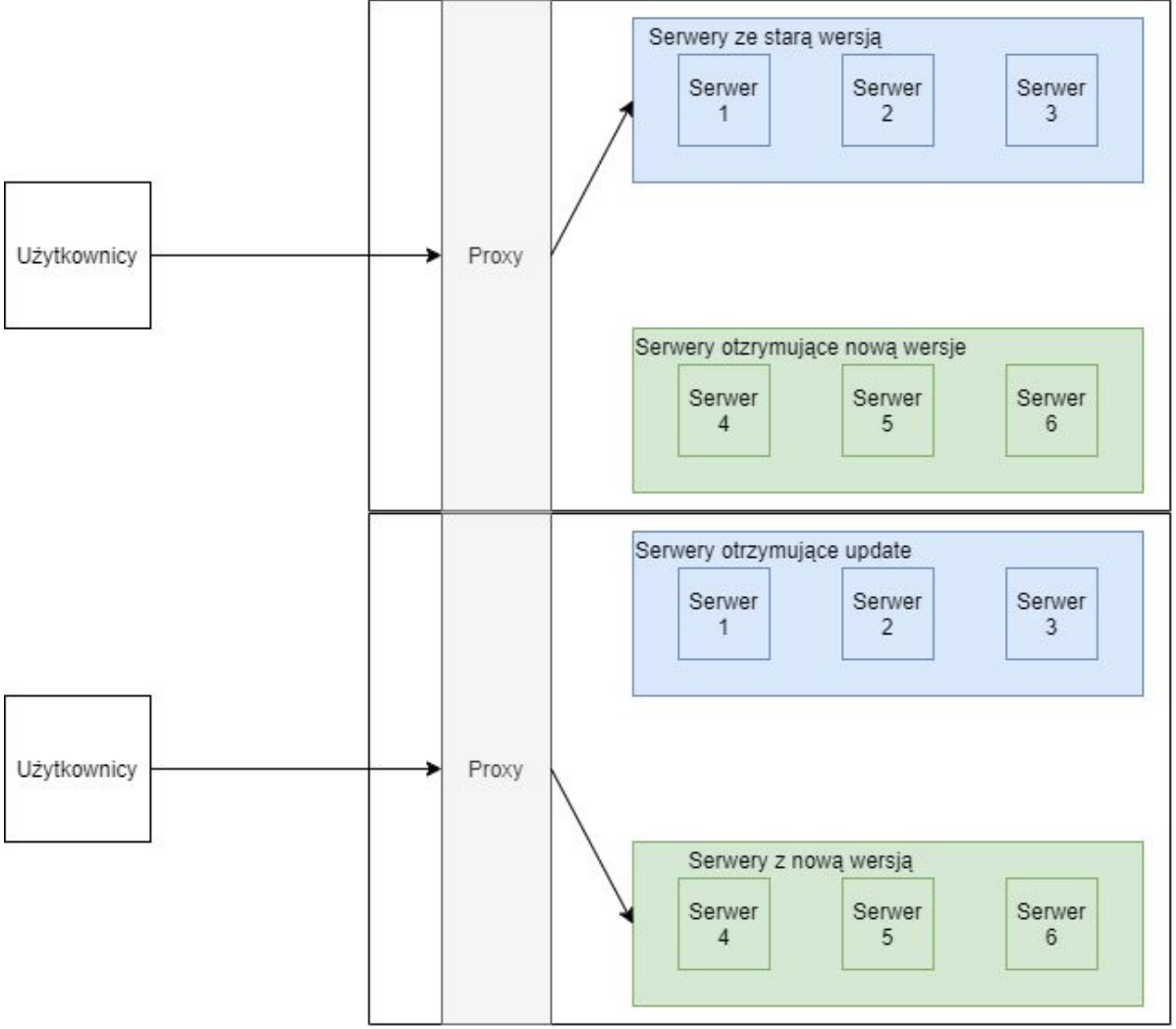
Blue/Green Deployment

Boże/Gdzie idą moje pakiety?... a... tu są

- Część serwerów wybierana jest do otrzymania nowej wersji
- Ruch użytkowników przekierowywany jest na serwery ze starą wersją
- Kiedy deploy nowej wersji na wybrane serwery jest ukończony, ruch użytkowników przekierowywany jest na serwery z nową wersją
- Pozostałe serwery ze starą wersją otrzymują update

Celem wdrożenia Blue/Green jest minimalizacja czasu gdzie aplikacja jest niedostępna

Podczas takiego deployu nie przeprowadzane są żadne testy ale pozwala to na szybkie wypchnięcie nowej wersji na wszystkie serwery produkcyjne bez zauważalnych luk w dostępności serwisu


Użytkownicy
Proxy
Serwery ze starą wersją
Serwer 1
Serwer 2
Serwer 3
Serwery otzrymujące nową wersje
Serwer 4
Serwer 5
Serwer 6
Użytkownicy
Proxy
Serwery otrzymujące update
Serwer 1
Serwer 2
Serwer 3
Serwery z nową wersją
Serwer 4
Serwer 5
Serwer 6

A/B Testing

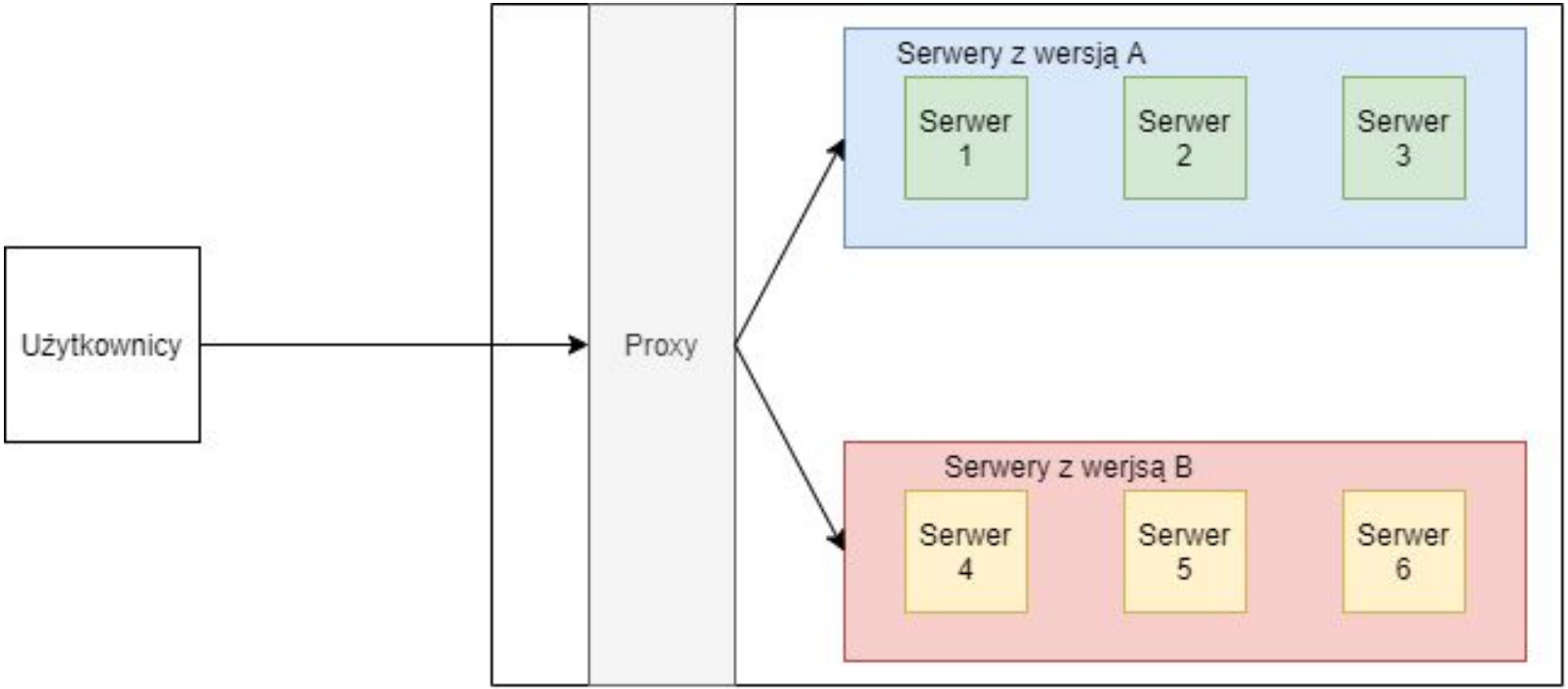
Ale Będzie krzyk o ten nowy layout

- Serwery produkcyjne są dzielone na dwa segmenty: A, które będą zawierały starą wersje aplikacji i B, które będą miały nową
- Wybrana jest pewna grupa użytkowników którzy otrzymają dostęp do wersji B
- Zbierane są opinie użytkowników o wersji B
- Jeśli opinie są pozytywne reszta serwerów produkcyjnych otrzymuje update podczas którego wszyscy użytkownicy są kierowani na serwery B

Testy A/B służą zwykle do sprawdzenia opinii użytkowników o zmianach w aplikacji. Zwykle chodzi o zmiany dotyczące frontendu.

W opisanym modelu A/B, A jest starą wersją aplikacji a B nową.

Inne podejście do testów A/B to stworzenie dwóch nowych wersji aplikacji o różnym wyglądzie i sprawdzenie która wersja zbiera więcej pozytywnych i deploy takiej wersji na reszte produkcji lub integracje jej najlepiej odbieranych elementów do następnej wersji


Użytkownicy
Proxy
Serwery z wersją A
Serwer 1
Serwer 2
Serwer 3
Serwery z werjsą B
Serwer 4
Serwer 5
Serwer 6

Wzorce Operacyjne

Koncept wzorców operacyjnych

1. Poprawne zidentyfikowanie problemu
2. Definicja potrzebnych operacji do rozwiązania problemów
3. wykonanie operacji w celu rozwiązania problemu

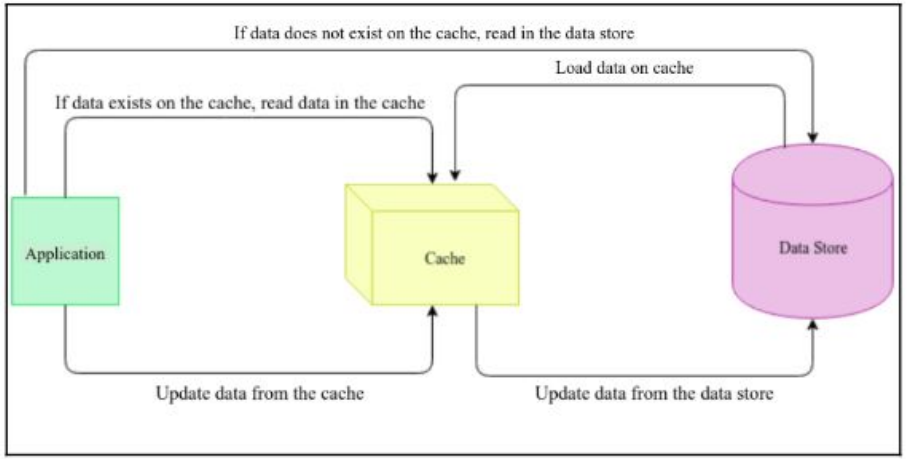
Rozwiązywanie problemów powinniśmy podzielić na następujące etapy:

1. Plan rozwiązania problemu
2. Zaprogramowanie rozwiązania bez przejmowania się wykonanym rozwiązaniem
3. Optymalizacja problemu

Przykładowe wzorce operacyjne

- **Cache-aside** - ładuje dane do pamięci cache z bazy danych na żądanie
- **CQRS** - oddziela operacje, które odczytują dane z operacji, które aktualizują dane przy użyciu oddzielnych interfejsów.
- **Event Sourcing** - działa w trybie append-only. Rejestruje akcje podejmowane w domenie.
- **Index table** - tworzy indeksy pomiędzy polami w bazie danych, które wywoływane są poprzez zapytania
- **Materialized view** - generuje wstępnie wypełnione widoki danych w co najmniej jednej bazie danych, gdy dane nie są idealnie sformatowane dla operacji wymagających zapytań
- **Sharding** - dzieli dane z baz danych w zestaw części
- **Static content hosting** - wdraża zawartość statyczną w oparciu o chmurę. Może dostarczać dane bezpośrednio do klienta.

Cache-aside



Wzorzec rekomendowany jest, gdy dane nie są często aktualizowane. Polega on na zapisaniu informacji do pamięci cache na żądanie i trzymanie informacji do czasu aż nie będzie potrzebna aktualizacja danych lub nie wygaśnie czas żywotności pamięci cache. Zaletą pamięci cache jest szybkość dostępu do pamięci i brak potrzeby ciągłego odpytywania bazy danych. W przypadku, gdy pamięć cache jest pusta dane są aktualizowane z bazy danych. Wadą wzorca jest to, że jeżeli dane zostaną zaktualizowanego z innego źródła możemy mieć różnice w pamięci cache i stanem faktycznym.

CQRS

Command and Query Responsibility Segregation jest wzorcem segregującym dane odczytu z operacji zapisu, a następnie tworzy oddzielny interfejs na te operacje. Zmusza nas do utworzenia modelu, który będzie odczytywał dane i osobnego modelu, który te dane zapisze, ale używając tylko tych danych, które są wymagane do zapisu wszystkich informacji.

Model powinien być wykorzystywany, gdy potrzebujemy wykonać konkretną biznesową rolę, która wykracza poza pojedynczy model encji w systemie, a dane te są aktualizowane częściej, lecz wzrasta ryzyko wykonania błędnych zapisów. Model warto wykorzystać również, gdy system ma problemy z łączeniem danych lub operacje wykonywane są równolegle.

Event sourcing

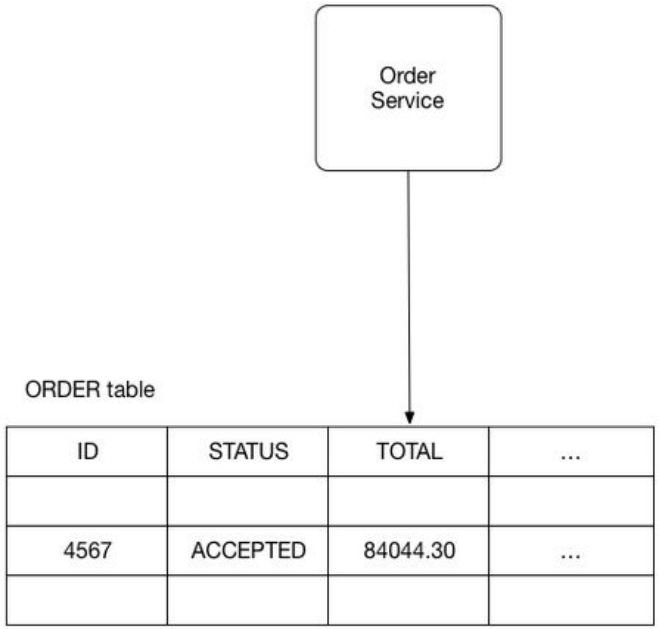
W zwykłej operacji aplikacji zwykle oczekujemy że aplikacja odczyta dane z magazynu, modyfikuje je i zapisze z powrotem do magazynu. Takie podejście może jednak powodować problemy.

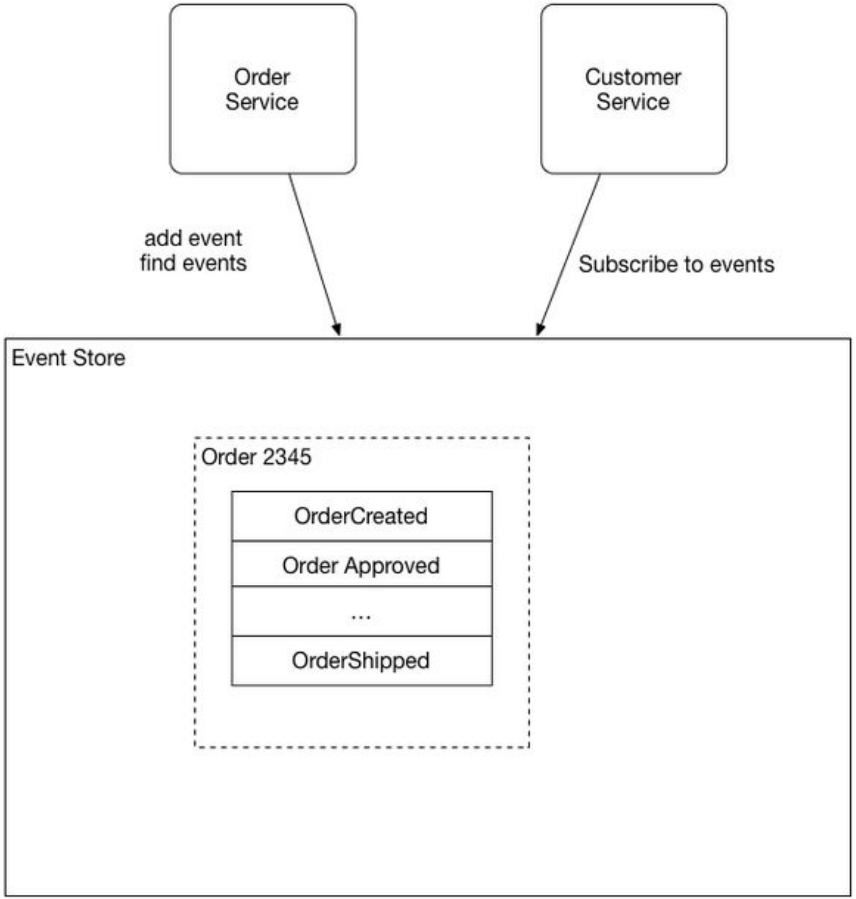
W systemie z wieloma użytkownikami mogą wystąpić konflikty danych a operacje bazodanowe zwykle są dość powolne. Dodatkowo historia zmian może zostać szybko zatracona

Rozwiązaniem jest Event Sourcing, gdzie zmiany danych są zapisywane do magazynu który przechowuje modyfikacje wykonane na danych jako ciąg operacji które należy wykonać na danych z magazynu aby odtworzyć z nich zmodyfikowane dane. Taki magazyn może służyć również jako historia zmian.

Działanie magazynu Event Sourcing

1. Otrzymaj od jednego użytkownika serię poleceń modyfikujących dane
2. Dołącz otrzymane komendy do historii operacji w kolejności chronologicznej
3. Wyemituj wydarzenie o modyfikacji danych do wszystkich subskrybentów
4. Zwróć nowe polecenia wszystkim żądającym subskrybentom
5. Co jakiś czas wykonaj snapshot logu poleceń i wykonaj go na bazie danych, jednocześnie rozpoczynając nowy log zaczynający się od poleceń otrzymanych po snapshocie
6. Przenieś snapshot do archiwum


Order
Service
ORDER table
ID
STATUS
TOTAL
...
4567
ACCEPTED
84044.30
...


Order
Service
Customer
Service
add event
find events
Subscribe to events
Event Store
Order 2345
OrderCreated
Order Approved
...
OrderShipped

Index table

- Wzorzec wykorzystywany do tworzenia indeksów dla tabel w bazie danych.
- Wzorzec może poprawić wydajność zapytań, umożliwiając aplikacjom szybsze lokalizowanie danych do pobrania z bazy danych.
- Daje możliwość odczytywania danych przy użyciu atrybutów, które nie zawierają indeksu jako filtru w zapytaniu.

Index table

Do konstruowania tabeli indeksów powszechnie stosuje się trzy strategie :

1. Strategia polegająca na zduplikowaniu danych w każdej tabeli indeksów, ale zorganizowaniu ich według różnych kluczy.
2. Tworzenie tabel indeksów uporządkowanych według różnych kluczy i odwoływanie się do oryginalnych danych przy użyciu klucza głównego zamiast jego duplikowania.
3. Strategia polegająca na tworzeniu częściowo znormalizowanych tabel indeksowych zorganizowanych według różnych kluczy, które duplikują często pobierane pola.

Index table

1

ID	NAME	AGE
1	Usuario A	20
2	Usuario B	22
3	Usuario C	20
4	Usuario D	19
5	Usuario E	22

INDEX(AGE)	ID	NAME	AGE
19	4	Usuario D	19
20	1	Usuario A	20
20	3	Usuario C	20
22	2	Usuario B	22
22	5	Usuario E	22

2

ID	NAME	AGE
1	Usuario A	20
2	Usuario B	22
3	Usuario C	20
4	Usuario D	19
5	Usuario E	22

INDEX(AGE)	ID
19	4
20	1
20	3
22	2
22	5

3

ID	NAME	AGE
1	Usuario A	20
2	Usuario B	22
3	Usuario C	20
4	Usuario D	19
5	Usuario E	22

INDEX(AGE)	ID	NAME
19	4	Usuario D
20	1	Usuario A
20	3	Usuario C
22	2	Usuario B
22	5	Usuario E

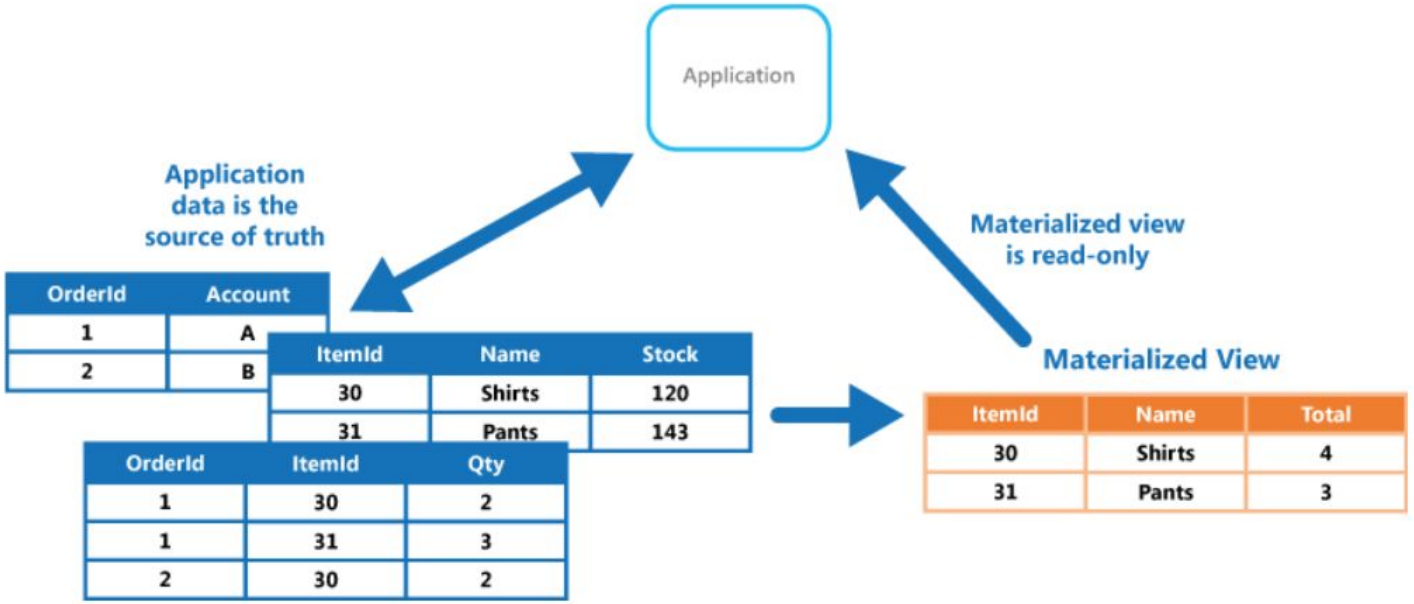
Materialized view

Czasami potrzebujemy zobaczyć dane połączone z różnych fizycznych lokalizacji. Często powoduje to problemy z wydajnością, spowalniając odczyt danych. W związku z tym, aby zwiększyć wydajność możemy utworzyć wstępnie wypełniony widok danych z wielu fizycznych lokalizacji - służy do tego wzorzec materialized view. Materialized view pattern charakteryzuje się tym, że wszelkie dane są odczytywane bez wykonywania łączeń lub obliczeń.

Należy zauważyć, że dane uzyskane w taki sposób nigdy nie są aktualizowane w widoku. Gdy rzeczywiste dane zostaną zaktualizowane, widok będzie musiał zostać odbudowany.

Dlatego zaleca się korzystanie z materialized view gdy dane są rzadko modyfikowane i nie są dynamiczne.

Materialized view



Sharding

Czasami tabela w magazynie danych lub sam magazyn z czasem rośnie do dużych rozmiarów które powodują problem z dodawaniem i odczytywaniem danych.

W takim wypadku rozwiązaniem problemu może być Sharding czyli podzielenie magazynu danych na odłamki. Każdy taki odłamek zawierać będzie identyczną strukture co monolityczny magazyn ale wpisy są dzielone pomiędzy odłamkami

Każdy odłamek dostanie część danych zwykle opierający się na jakiejś charakterystyce danych np państwo z którego pochodzi zamówienie. Dane które posłużą nam do podzielenia monolita na odłamki staną się kluczem (shard key lub partiction key). Istnieją trzy zwykle stosowane metody wyboru takiego klucza

Wzorce Shardingu

Lookup strategy

Range strategy

Hash strategy

Lookup Strategy

W tej strategii implementowana jest mapa która kieruje żądanie do fragmentu zawierającego te dane przy użyciu klucza fragmentu. W aplikacji wielodostępnej wszystkie dane jednostki mogą być przechowywane razem we fragmencie przy użyciu identyfikatora jednostki jako klucza fragmentu. Wiele jednostek może współużytkować ten sam fragment, ale dane dla jednej jednostki nie zostaną rozłożone na wiele fragmentów.

Range strategy

W tej strategii implementowana jest metoda która umożliwia grupowanie powiązanych elementów, które są uporządkowane według klucza poszczególnego fragmentu. Klucze fragmentu w tej strategii są sekwencyjne. Jest to przydatne w przypadku aplikacji, które regularnie znajdują elementy za pomocą zapytań zwracających zestaw zakresu danych w poszczególnym fragmencie.

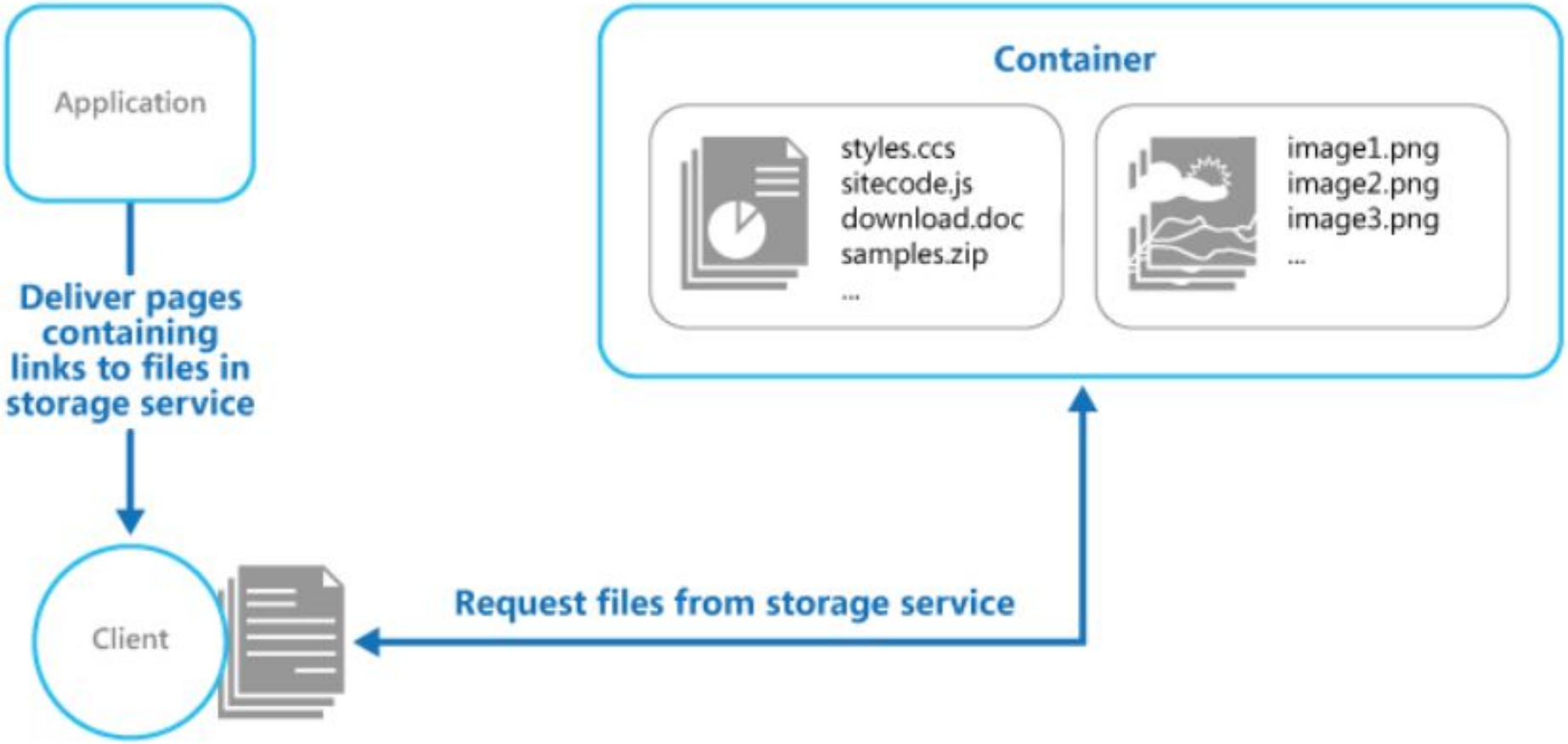
Hash Strategy

W tej strategii implementowana jest metoda wykorzystująca funkcję skrótu, która umożliwia równomierne rozłożenie danych między fragmentami. Celem tej strategii jest zmniejszenie ryzyka nieproporcjonalnie obciążonych fragmentów.

Static content hosting

Aplikacje internetowe zazwyczaj zawierają pewne elementy zawartości statycznej. Ta statyczna zawartość może zawierać strony HTML i inne zasoby. Chociaż serwery WWW są zoptymalizowane pod kątem renderowania dynamicznego i buforowania danych wyjściowych, nadal muszą obsługiwać żądania pobierania zawartości statycznej. Zużywa to cykle przetwarzania, które często można by lepiej wykorzystać.

W większości środowisk hostingu w chmurze można umieścić niektóre zasoby aplikacji i strony statyczne w usłudze magazynu danych. Usługa magazynu może obsługiwać żądania dotyczące tych zasobów, zmniejszając obciążenie zasobów obliczeniowych obsługujących inne żądania sieci Web. Koszt instancji magazynu hostowanego w chmurze jest zwykle znacznie niższy niż w przypadku instancji przetwarzających dane.


Application
Deliver pages containing links to files in storage service
Client
Request files from storage service
Container
styles.ccs
sitecode.js
download.doc
samples.zip
...
image1.png
image2.png
image3.png
...

Wzorce Zarządzania

Po co nam wzorce zarządzania

Coraz więcej aplikacji tworzonych jest z infrastrukturą chmury jako docelowym środowisku. Czy to w AWS czy Azure, nasza aplikacja pewnie będzie uruchomiona w data center daleko od lokalizacji.

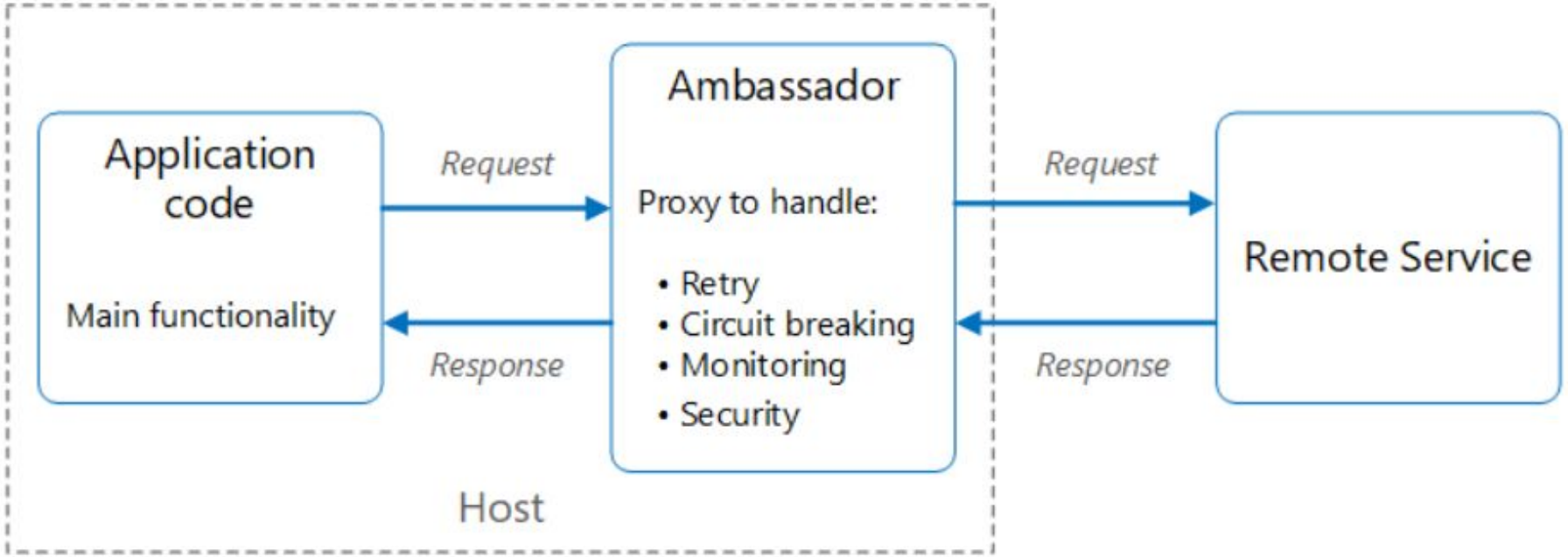
Dzięki wzorcom zarządzania możemy w łatwy sposób monitorować i zarządzać stanem naszej aplikacji, modyfikować konfigurację pewnych elementów bez czasochłonnego redeployu. Dodatkowo możemy oddzielić logikę biznesową od logiki monitoringu dla poprawienia przejrzystości kodu.

Ambasador

Czasami trzeba zaktualizować config routingu aplikacji albo po prostu monitorować ruch. Ale zmiana kodu aplikacji w celu osiągnięcia tych efektów może być trudna lub niemożliwa. Być może aplikacja jest po prostu stara i pełna legacy code albo biblioteka sieciowa którą stosuje nie udostępnia pewnej funkcjonalności.

Dlaczego więc nie wyrzucić tego wszystkiego poza aplikacje?

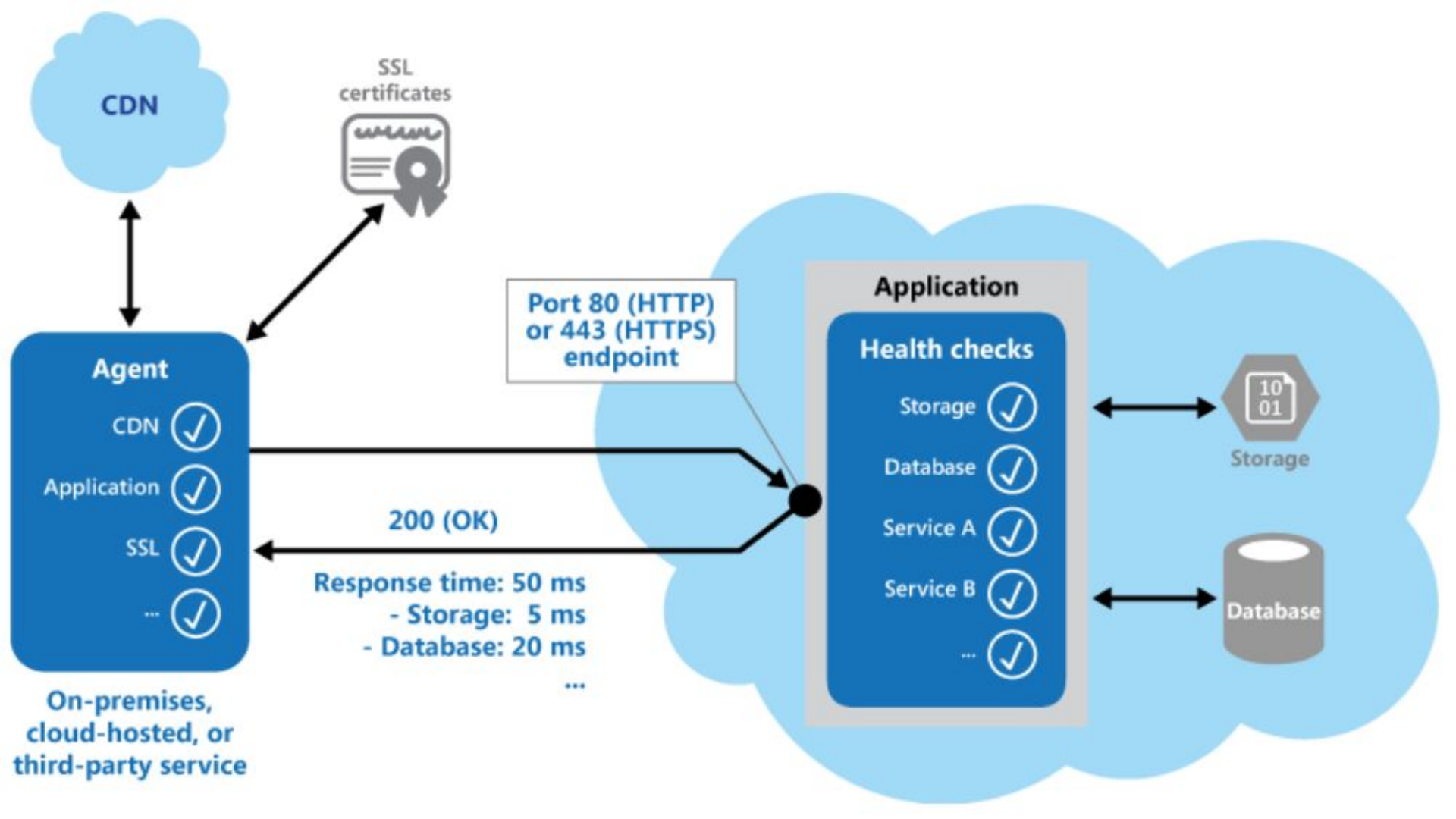
Ambasadora można traktować jako serwer proxy poza procesem, który jest zlokalizowany wspólnie z klientem. Dzięki temu sprawy sieciowe mogą być przepuszczane przez amasadora który zajmie się sprawami sieciowymi w współczesny sposób bez potrzeby modyfikowania samej aplikacji


Application code
Main functionality
Request
Response
Ambassador
Proxy to handle:
• Retry
• Circuit breaking
• Monitoring
• Security
Request
Response
Remote Service
Host

Health Endpoint

Dobrą praktyką, a często wymaganiem biznesowym, jest monitorowanie aplikacji internetowych i usług backend, aby upewnić się, że są one dostępne i działają poprawnie. Jednak czasami monitorowanie usług działających w chmurze jest trudniejsze niż monitorowanie usług lokalnych

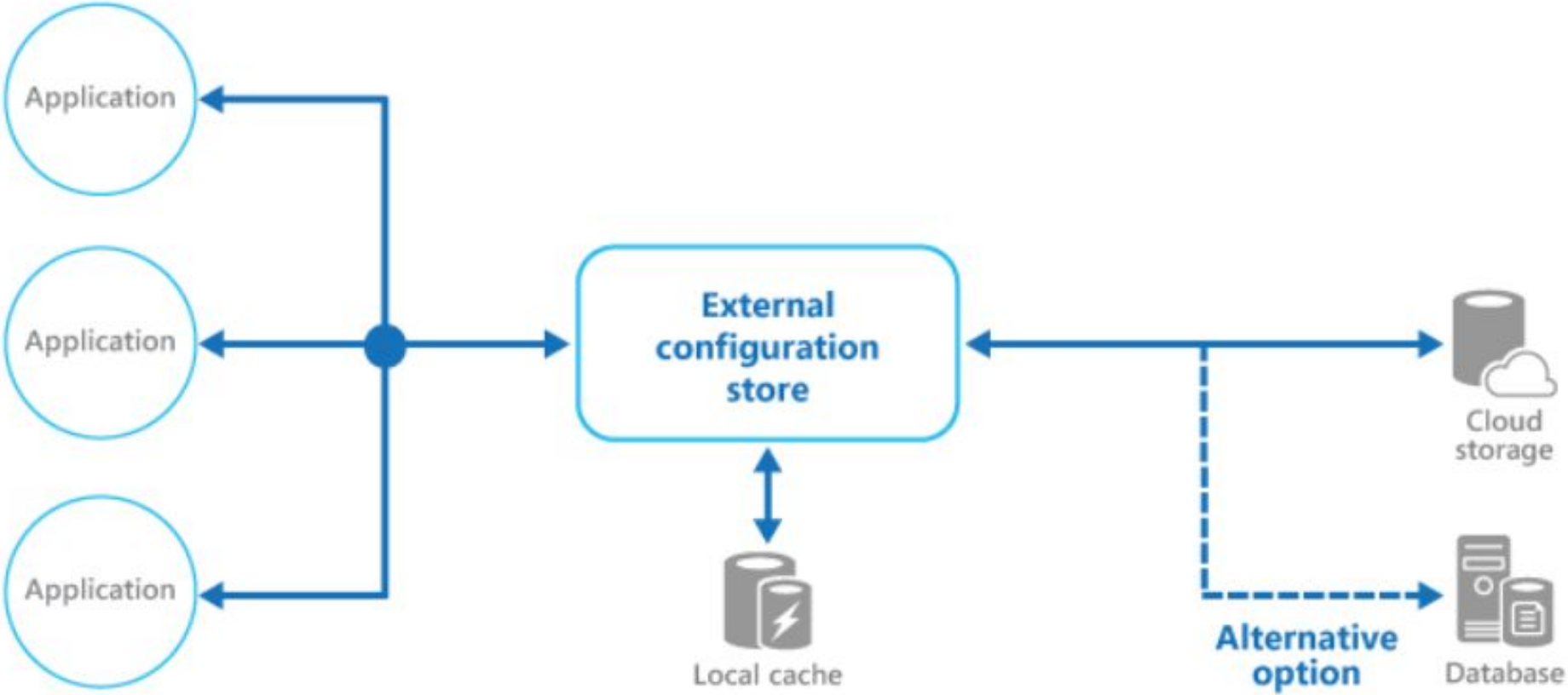
Wzorzec Health Endpoint polega na implementacji w aplikacji chmurowej endpointu który po otrzymaniu rządania, przeprowadzi pełną diagnostykę aplikacji i zwróci jej stan i pewne metryki, np czasy odpowiedzi, do drugiej części wzorca, czyli aplikacji monitorującej która oprócz sprawdzenia chmury, sprawdzi inne elementy systemu jak np, certyfikaty SSL czy config sieciowy


CDN
SSL certificates
Agent
CDN
Application
SSL
...
On-premises, cloud-hosted, or third-party service
Port 80 (HTTP) or 443 (HTTPS) endpoint
200 (OK)
Response time: 50 ms
- Storage: 5 ms
- Database: 20 ms
...
Application
Health checks
Storage
Database
Service A
Service B
...
10 01
Storage
Database

External Configuration Store

Większość środowisk wykonawczych aplikacji przechowuje konfigurację w plikach wdrożonych wraz z aplikacją. W niektórych przypadkach można edytować te pliki, aby zmienić zachowanie aplikacji po jej wdrożeniu. Jednak zmiany w konfiguracji wymagają ponownego wdrożenia aplikacji, co często skutkuje niedopuszczalnymi przestojami i innymi kosztami administracyjnymi.

Wzorzec magazynu konfiguracji zewnętrznej tworzy zewnętrzne repozytorium konfiguracji, udostępniając interfejs umożliwiający aplikacjom odczyt plików konfiguracyjnych. Dzięki temu możemy aktualizować pliki konfiguracyjne bez konieczności ponownego wdrażania aplikacji, możemy również udostępniać konfiguracje między kilkoma aplikacjami


Application
Application
Application
External configuration store
Local cache
Cloud storage
Alternative option
Database