Rozdział 8 i 9

Grupa nr 4:
Kamil Pierzchała
Jakub Dobrowolski
Paweł Malec
Michał Półchłopek
Kamila Skałba

Wzorce aplikacji cloud-native

Przyjrzymy się, czym jest aplikacja cloud-native i jakie są jej cele.

W tym rozdziale omówimy następujące tematy:

- Wyjaśnienie koncepcji aplikacji natywnych w chmurze
- Wyjaśnienie celów aplikacji w chmurze
- Wyjaśnianie wzorców projektowych chmury

Wyjaśnienie pojęcia aplikacji chmurowych

W przeszłości zasoby chmurowe były wykorzystywane jedynie jako pamięć masowa. Obecnie, wdrażanie aplikacji korporacyjnych w chmurze staje się coraz popularniejsze. Dużym jej wyzwaniem jest sprawienie, aby aplikacja posiadała cechy takie jak elastyczność, skalowalność i dostępność (główne cechy chmury obliczeniowej). A więc aplikacja, która wykorzystuje zalety i możliwości chmur to właśnie cloud-native. Do stworzenia takiej aplikacji wykorzystuje się zestaw wzorców projektowych. Elastyczność w chmurze oznacza, że zasoby obliczeniowe są wykorzystywane i uwalniane zgodnie z zapotrzebowaniem.

Właściwości aplikacji cloud-native:

Pakowanie w kontenery - aplikacje wykonywane w odizolowanych jednostkach

Dynamiczne zarządzanie - Istnieje centralny punkt, który zarządza aplikacjami poprawiając wykorzystanie zasobów i redukując z nią związane koszty.

Zorientowanie na mikroserwisy - Aplikacje w chmurze są luźno sprzężone czyli zmiana w jednym komponencie nie powinny wpływać na iny komponent. (z ang. loose coupling)

Wyjaśnienie celów aplikacji cloud-native

Cechy, które powinna posiadać aplikacja cloud-native:

- Dostępność-pożądane jest nieprzerwane działanie wszystkich funkcjonalności systemu.
- Zarządzanie danymi-dane mogą być replikowane na wiele serwerów.
- Przesyłanie komunikatów-usługi lub aplikacje komunikują się za pomocą asynchronicznego przesyłania komunikatów.
- Zarządzanie i monitorowanie-logi oraz raporty a także nowe implementacje bez przerwania działania.
- Wydajność i skalowalność-pomimo wzrostu wykorzystania zasobów utrzymuje wydajność na założonym poziomie.
- Odporność-aplikacja powinna być w stanie szybko poradzić sobie z awariami.
- Bezpieczeństwo-elementy dostępne dla użytkownika powinny być odporne na ataki.

Wyjaśnienie wzorców projektowych chmury

1. **Aplikacja złożona(mikroserwisy)** - wzorzec projektowy charakteryzujący się rozłożeniem aplikacji na mniejsze, funkcjonalne i niezależne części, (z dobrze zdefiniowanym interfejsem) które są między sobą luźno powiązane.
2. **Abstrakcja** - głównym jej założeniem jest skupienie na potrzebach klienta, a nie na istniejącej już strukturze hardware'owej. W ten sposób, zasoby chmury są użytkowane na żądanie, co charakteryzuje elastyczną skalowalność. Dzięki temu zasoby chmury mogą być dopasowywane zależnie od potrzeb klienta.
3. **Brama API** - służy jako front-end dla klientów aplikacji chmurowych. Działa jako warstwa odbierająca żądanie i wykonuje kilka mikrousług związanych z wymaganą funkcjonalnością. W zależności od typu klienta aplikacja może dostarczać różnych odpowiedzi.
4. **Wzorzec rejestru usług** - jest to baza danych zarejestrowanych usług. Kiedy tworzony jest mikroserwis jest on w niej zapisywany, a gdy umiera znika z bazy. Klient uzyskuje dostęp do bazy, następuje sprawdzenie dostępności danej usługi i udostępnienie klientowi jej lokalizacji.
5. **Serwer konfiguracji -** służy do zarządzania właściwościami aplikacji zbudowanej w oparciu o mikroserwisy. Warstwa ta odpowiada za utrzymanie tych właściwości i jeżeli są one zmieniane to zmiana ta jest odzwierciedlana w mikroserwisie (lub aplikacji) bez konieczności przebudowy lub restartu serwisu.
6. **Schemat wyłącznika** - zapobiega wielokrotnemu wykonywaniu się operacji w sytuacji, gdy najprawdopodobniej zakończą się błędem w trakcie wykonywania. Dodatkowo wzorzec ten sprawdza czy błąd został naprawiony i jeśli tak to proxy wysyła żądanie wykonania operacji, a jeśli nie licznik niepowodzeń jest zwiększany.

Mechanizm wyłącznika automatycznego - posiada trzy odrębne stany: closed, open i half-open.

7. **Dwanaście czynników** - metodologia oparta na dwunastu czynnikach prowadzących do udanej realizacji projektu SaaS (Software as a Service). Czynniki te to:

- **Codebase** - jest to każde repozytorium, w którym jest przechowywany kod, dla każdej aplikacji istnieje jedno takie repozytorium. Musi być ono zarządzane poprzez system kontroli wersji.
- **Zależności** - każda zależność użytkowana przez projekt musi zostać zadeklarowana i wyizolowana z kodu. Jednym z przydatniejszych narzędzi do zarządzania paczkami jest np. Maven.
- **Config** - jest to wszystko co różni się pomiędzy poszczególnymi wdrożeniami np. porty, sposoby uwierzytelnienia dostępu do baz danych etc. Metoda dwunastoczynnikowa wymaga odseparowania konfiguracji od kodu, gdyż konfiguracje mogą różnić się pomiędzy deployami, a kod powinien pozostać taki sam.
- **Usługi wspierające** - są to zewnętrzne komponenty/usługi użytkowane przez aplikację tj. bazy danych, repozytoria plików, usługi komunikacyjne (przesyłu wiadomości, maili). Każdy z tych komponentów musi być dostępny poprzez adres URL lub lokalizację i uwierzytelnienie jakie zawiera konfiguracja aplikacji. W ten sposób żadne zmiany w lokalizacji danej usługi nie wpłyną na kod aplikacji.

- **Build,release,run** - jest to proces transformacji kodu w środowisko. Build odpowiada za kompilację i wygenerowanie wykonywalnej paczki. Release polega na zastosowaniu konfiguracji aplikacji w pakiecie wykonywalnym, wynik release jest gotowy na przeskanowanie w środowisku, które zawiera konfigurację. Run natomiast inicjalizuje aplikację w konkretnym środowisku.
- **Procesy** - metodologia dwunastoczynnikowa wymusza wyjałowienie procesów i komponentów, oznacza to, że żadne z nich nie powinno zawierać informacji w pamięci któregokolwiek komponentu aplikacji. Jeśli wyniknie potrzeba przechowania jakichkolwiek danych do późniejszego wykorzystania, z pomocą przychodzi baza danych.
- **Port-binding** - aplikacja implementowana za pomocą metody dwunastoczynnikowej jest całkowicie samowystarczalna i niezależna od użycia zewnętrznych serwerów. Musi on eksportować usługę HTTP poprzez port-binding co oznacza, że łączy się ze światem za pomocą adresu URL. W ten sposób jedna aplikacja może być wspierającym lub zewnętrznym zasobem drugiej.
- **Współbieżność** - aplikacja implementowana w tej metodologii musi być skalowalna pod względem wykonywania procesów w trybie równoległym.

- **Dyspozycyjność** - w tej metodologii powinna istnieć możliwość by proces został rozpoczęty lub zakończony w dowolnym czasie. Proces zakończony bez jakiegokolwiek impaktu oznacza jego pozytywne zakończenie z zapisaniem stanu i uwolnieniem przydzielonych mu zasobów.
- **Parzystość dev/prod** - oznacza, że środowiska produkcyjne i testowe powinny być utrzymywane w jak najbardziej zbliżonych stanach co ułatwia proces wdrażania i unikania błędów podczas przesyłania zmian z developmentu na produkcję.
- **Logi** - powinnny być traktowane jako ciąg zdarzeń w szczególności w środowisku chmurowym, gdzie procesy rozpoczynają się i są kończone co chwilę.
- **Procesy administracyjne** - metoda dwunastoczynnikowa wskazuje na to, że procesy utrzymania powinny być zautomatyzowane i wykonywane na czas. Z racji tego, że aplikacja działa w wielu środowiskach wymaga użycia tych samych narzędzi do wykonania poszczególnych zadań co znacznie zmniejszy problemy z parzystością pomiędzy środowiskami.

Wzorce bezpieczeństwa

W tym rozdziale przyglądniemy się wzorcom bezpieczeństwa i jak mogą one pomóc nam implementować bezpieczniejsze aplikacje. Będziemy omawiać

- Główne koncepty wzorców bezpieczeństwa
- Idea wzorca pojedynczego logowania (ang. single-sign-on)
- Mechanizmów uwierzytelniania
- Wyjaśnienie czym jest przechwytywanie uwierzytelniania

Podstawowe założenia wzorców bezpieczeństwa

Aplikacje mają ścisły związek z danymi i ich przechowywaniem. Wiele firm musi przechowywać delikatne dane i zapewniać kontrolę dostępu. Integralnym elementem bezpieczeństwa, a co za tym idzie bezpiecznych aplikacji jest informacja dotycząca bezpieczeństwa, które opiera się na następujących podstawowych zasadach

- Poufność
- Integralność
- Dostępność
- Niezaprzeczalność

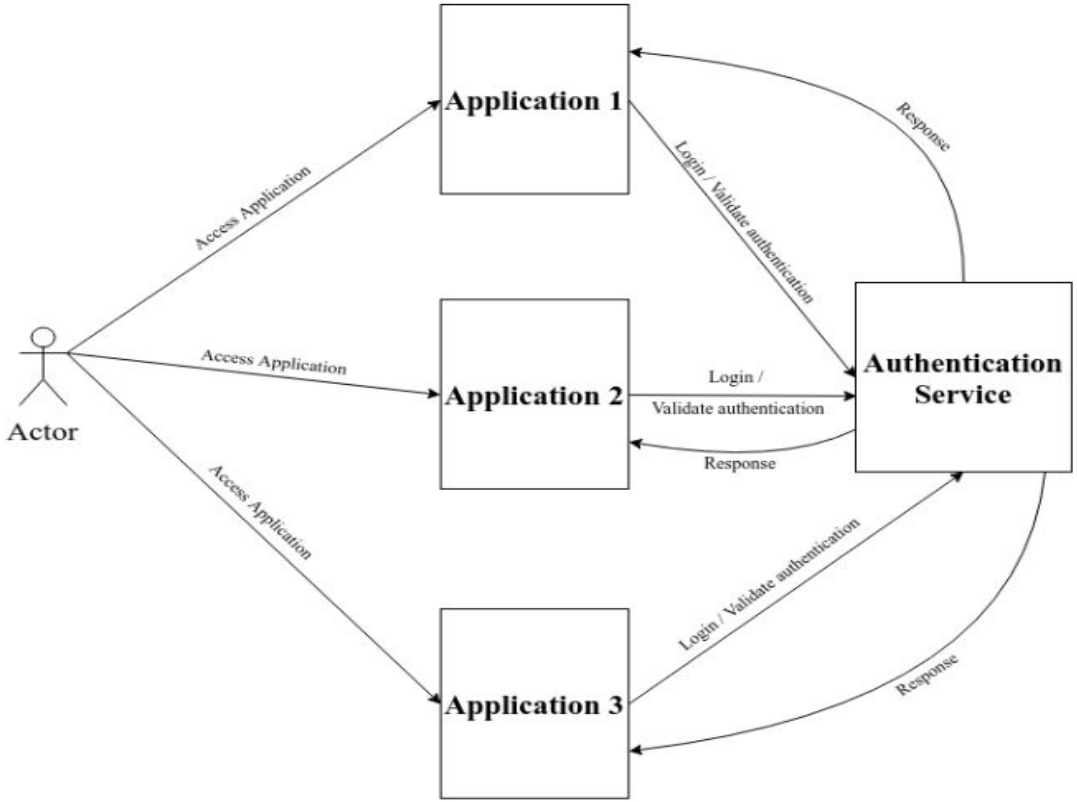
Czym są wzorce bezpieczeństwa

Wzorce bezpieczeństwa są zestawem rozwiązań dla częstych problemów z bezpieczeństwem. Duża część tych wzorców bezpieczeństwa służy do rozwiązywania problemów związanych z uwierzytelnianiem, które jest związane z zasadami poufności i integralności.

Dzięki tym wzorcom programista może napisać oprogramowanie z wysokim poziomem bezpieczeństwa i dla znanych problemów zastosować rozwiązania które są już przetestowane i zweryfikowane

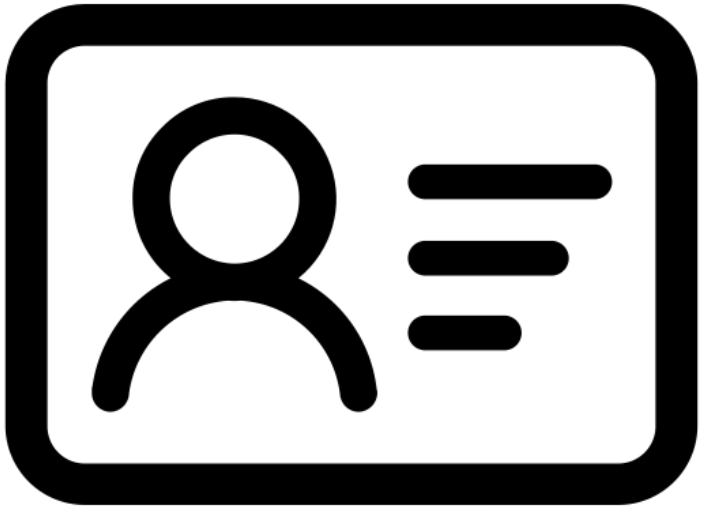
Wzorzec pojedynczego logowania

Wzorzec ten skupia się na stworzeniu usługi uwierzytelniania współdzieloną z kilkoma aplikacjami w domenie. Przykładem takiego uwierzytelniania jest Google gdzie możemy zaobserwować, że po zalogowaniu się np. do Gmail jesteśmy jednocześnie zalogowani do YouTube



Mechanizmy autentykacji

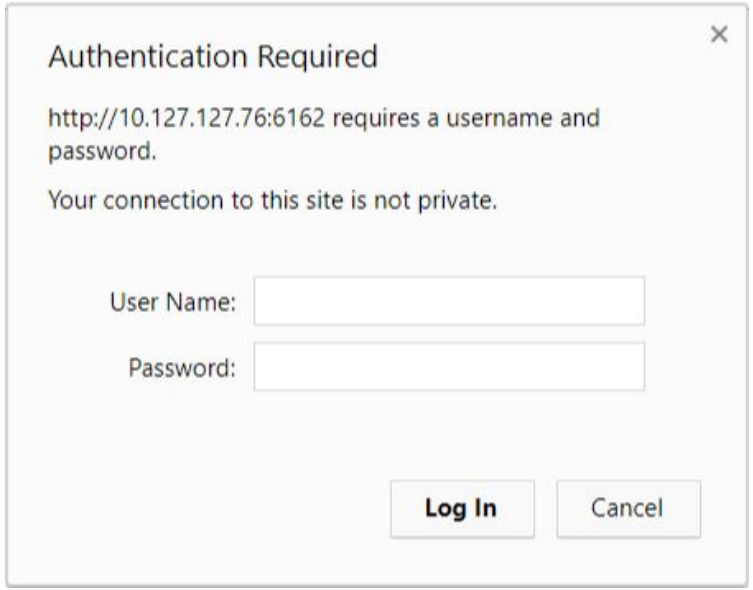
- Podstawowa autentykacja
- Autentykacja za pomocą formularza
- Digest
- Autentykacja za pomocą klucza
- Autentykacja dwustronna



Podstawowa autentykacja

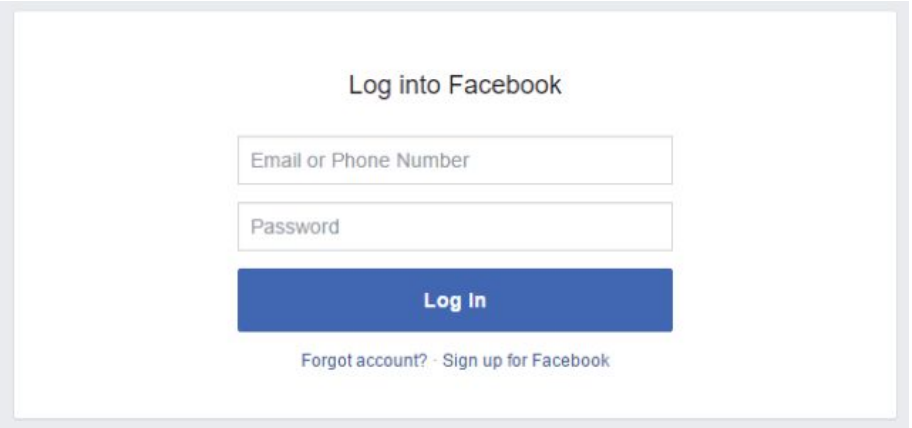
Jedną z form autentykacji jest autentykacja określona jako w “Basic authentication”. Nie wymaga dużego wysiłku ze strony dewelopera. Aplikacje używające tej formy wyświetlają “dialog box” pozwalający na wprowadzenie danych użytkownika.

Podatny na ataki typu “man-in-the-middle”, ze względu na przesyłanie danych w formie czystego tekstu



Autentykacja za pomocą formularza

Forma podobna do “podstawowej autentykacji”, ze względu na sposób przesyłania danych. Pozwala jednak na większą swobodę w kwestii dostosowywania wyglądu graficznego interface’u.

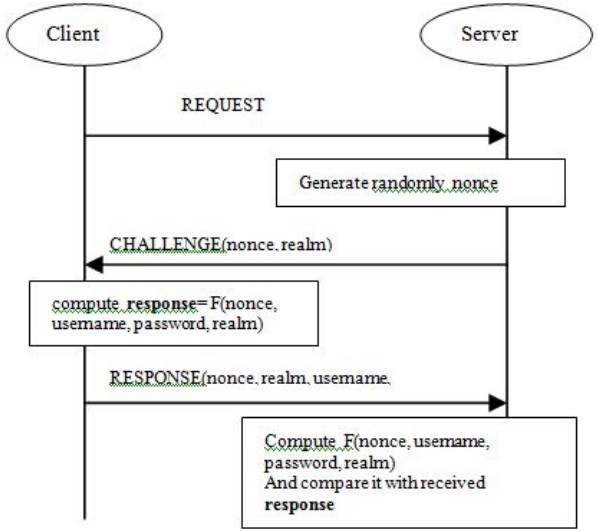


Podatny na ataki typu “man-in-the-middle”, ze względu na przesyłanie danych w formie czystego tekstu

Digest authentication

```
HA1 = MD5(username:realm:password)
HA2 = MD5(method:digestURI)
response = MD5(HA1:nonce:HA2)
```

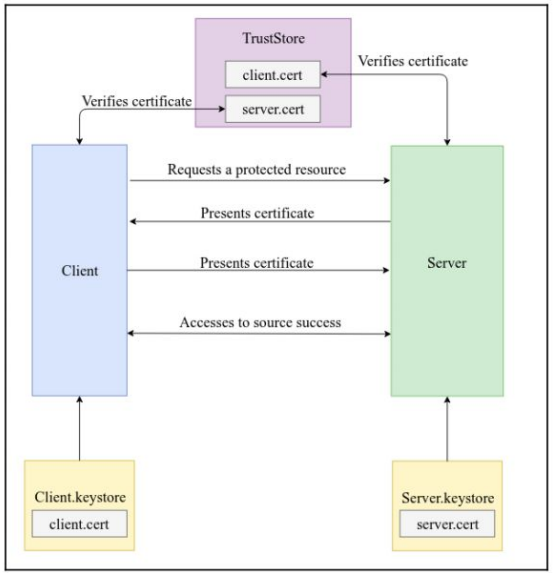
Wykorzystuje jednokierunkową funkcję hashującą w celu zaszyfrowania przesyłanych danych. Dzięki temu zabiegowi przechwycenie danych w trakcie przesyłania nie stanowi potencjalnego zagrożenia dla systemu.



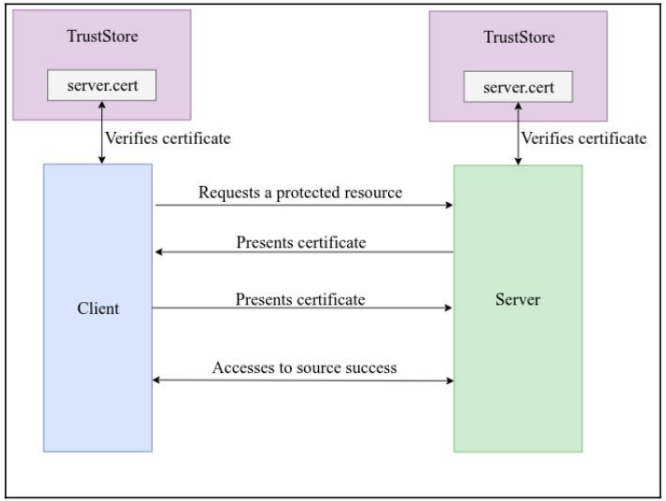
Autentykacja za pomocą klucza

Ta forma autentykacji nie wymaga od użytkownika podania danych za pomocą formularza. Autentykacja polega na wykorzystaniu pary kluczy i kryptografii asymetrycznej.

Dwustronna autentykacja



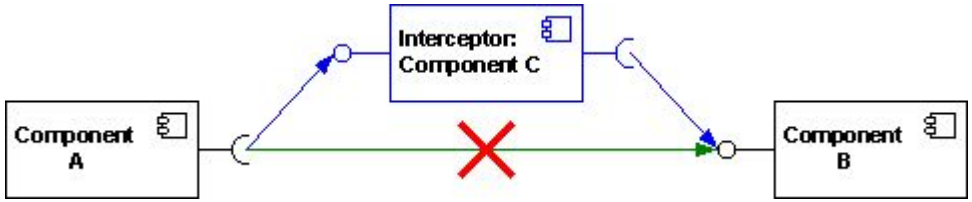
Mechanizm, w którym zarówno serwer wymaga autentykacji od klienta jak i klient od serwera.



certificate-based : obie strony dokonują atentykacji za pomocą certyfikatu.

username/password-based : serwer zapewnia certyfikat, natomiast autentykacja klienta przebiega za pomocą loginu i hasła.

Authentication interceptor



Interceptor pattern (wzorzec przechwytywania) - to zaawansowana technika programowania pozwalająca na przechwytywanie wywołania obiektu i przetworzeniu jego algorytmów przed lub po wywołaniu.

Wzorzec jest oparty o założenia programowania obiektowego, co pozwala pozwala na zmiany w trakcie działania bez ingerencji w jego logikę.

Java EE 8 pozwala nam skorzystać z interceptora EJB(Enterprise JavaBeans) lub CDI(Context and Dependency Injection).

Zmiany które powoduje są ‘transparent and used automatically’, tzn. reszta aplikacji nie wie że coś zostało dodane lub zmienione, dzięki czemu może pracować jednym ciągiem.

Authentication interceptor (przechwytywacz uwierzytelniania) - technika wykorzystująca wzorzec przechwytywania. Korzysta z interceptora EJB lub CDI.

Dzięki temu możemy skorzystać z mechanizmu autoryzacji przed lub po biznesowej części programu. Daje nam to możliwość walidacji bez ingerencji części autoryzacyjnej z logiką biznesową.

Dziękujemy za uwagę!