Architektura i narzędzia w systemach mikroserwisowych

Hello

Oleksii Tsyganov

Agenda

Welcome
History of Microservices
Problems of Monolith & SOA
Microservices Architecture
Problems Solved by Microservices
Designing Microservices Architecture
Deploying Microservices
Testing Microservices
Service Mesh
Logging And Monitoring
When NOT to use Microservices
Microservices and the Organization
Anti-Patterns and Common Mistakes
Breaking Monolith to Microservices
Case Study
Conclusion

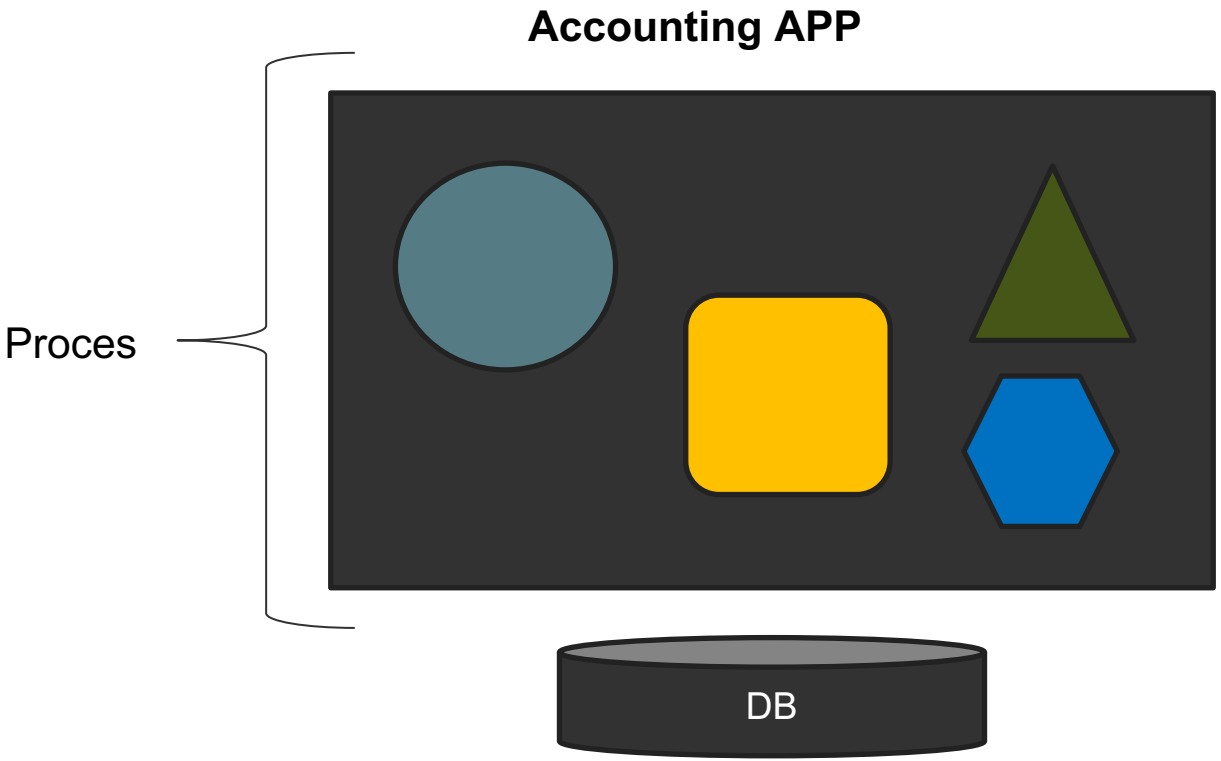
Before microservices. Monolith.

Jedyny proces

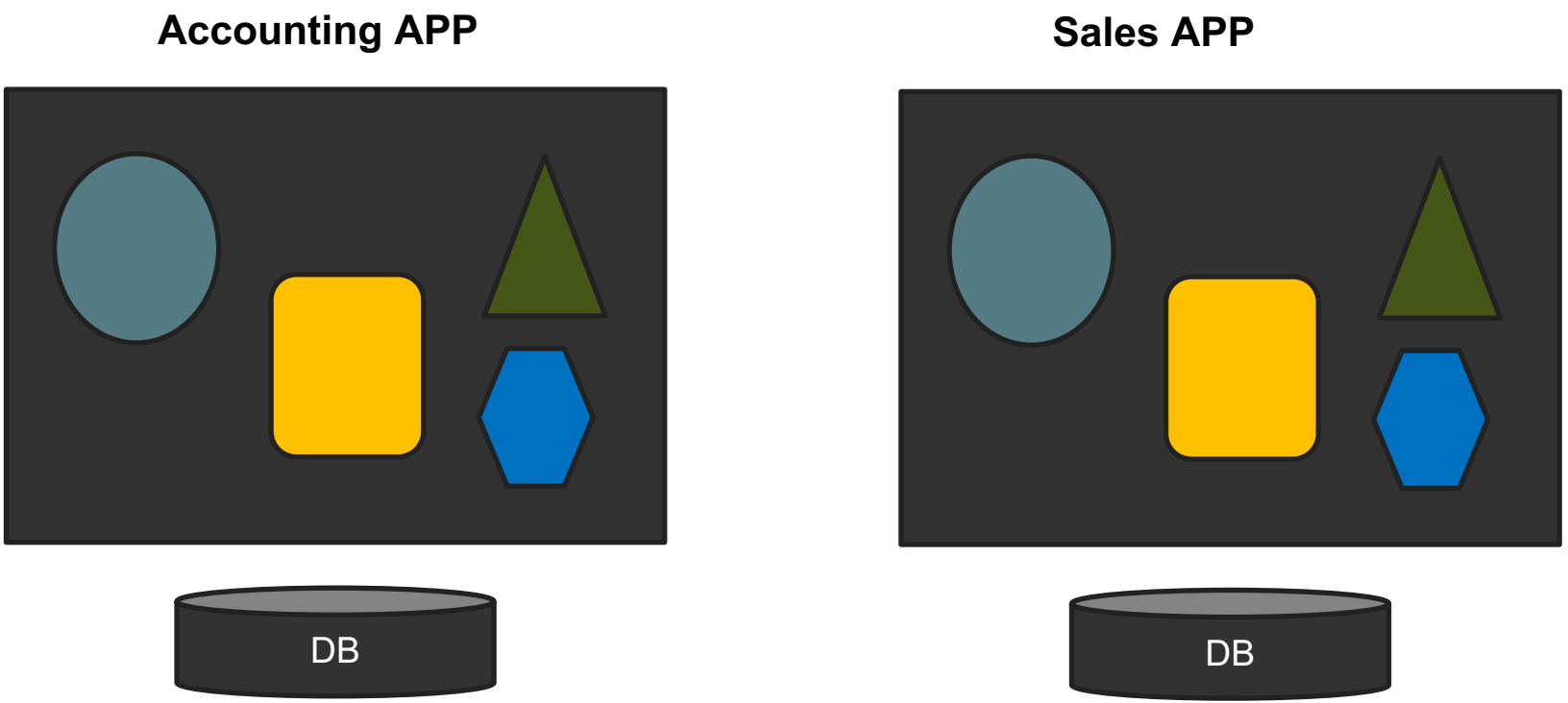
Związki pomiędzy wszystkimy klasami

Implementowana jako silos

Before microservices. Monolith



Before microservices. Monolith



Monolith. Pros.

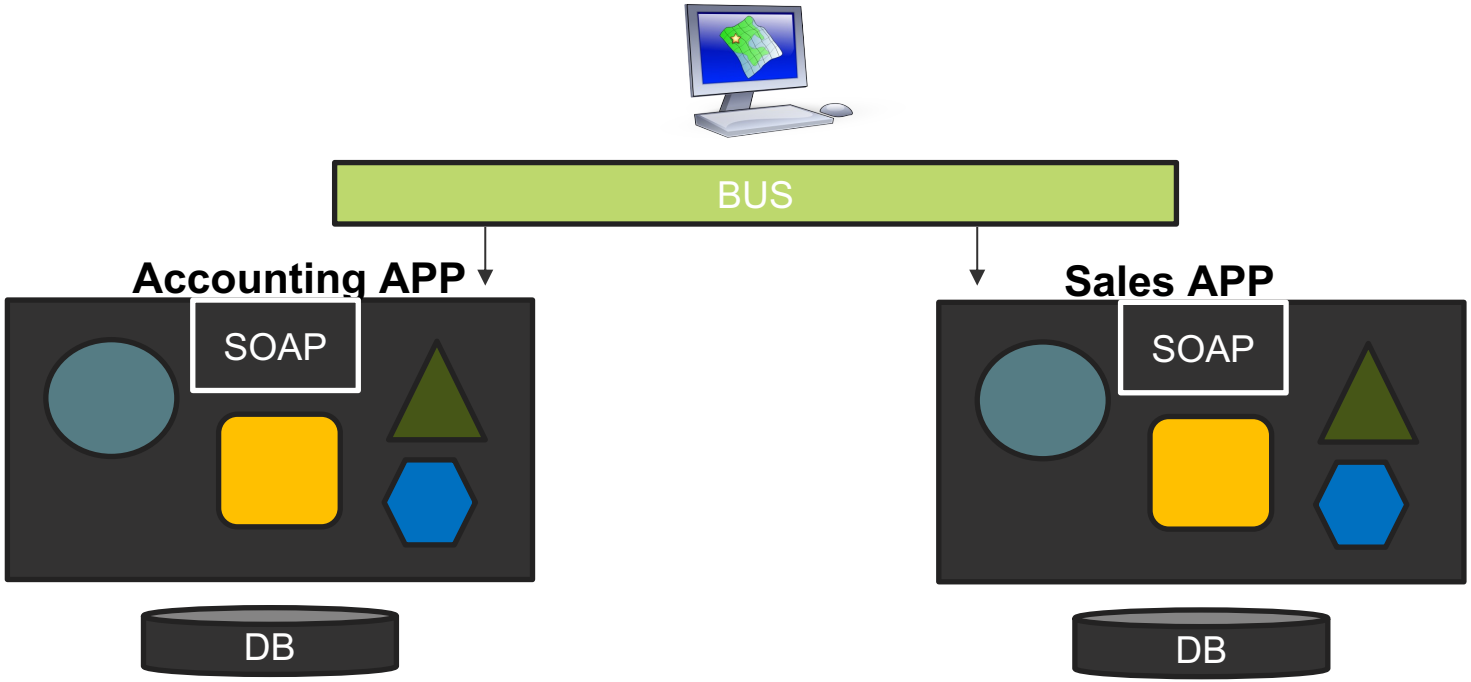
Łatwo projektowac

Wydajność

Before microservices. Service Oriented Architecture

- Aplikację są serwisami, które wystawiają swoją funkcjonalność “w świat”
- Dzielenie się I dawanie
- Serwisy wystawiają metadane zeby zanznaczyć swoją funkcjonalność
- Zwykle implementowane za pomocą SOAP I WSDL
- Implemetowane za pomocą Enterprise Service Bus (ESB)

Before microservices. Service Oriented Architecture



SOA. Pros.

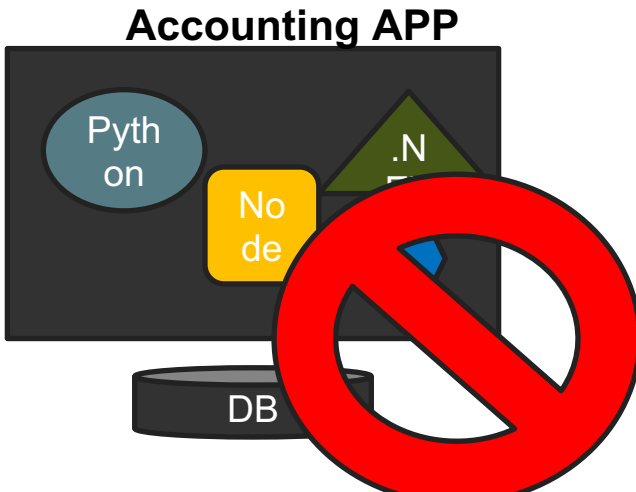
Udostępnianie danych I funkcjonalności

Polyglotowe podejscie

Problems with Monolit and SOA

Jedyna platforma technologiczna:

- wszystkie komponenty muszą być dewelopowane na tej samej platformie
- nie zawsze technologia jest lepsza dla zadania
- upgrade jest problemem



Problems with Monolit and SOA

Nielastyczne wdrażenie:

- Zawsze deplojujemy całą aplikacje
- Nie ma możliwosci deploju częściowego
- Update tylko jednego komponentu wymaga deploymentu całości “codebase”
- Wymyszanie uciążliwego procesu testow całości
- Dlugie cykly deploymentowe

Problems with Monolit and SOA

Nieefektywne korzystanie z mocy obliczeniowej:

- W monolicie CPU i RAM dzielone pomiędzy wszystkimi komponentami
- Nie ma możliwości przydzielenia większej liczby zasobów dla komponenta systemu

Problems with Monolit and SOA

Skala i skomplikowaność:

- “Codebase” jest skomplikowany i duży
- Mała zmiana może spowodać problemy z innymi komponentami
- Testowanie nie zawsze wykrywa bugi
- Wsparcie dla produktu jest bardzo trudne
- Bardzo trudne utrzymywanie
- Przestarzały system

Problems with Monolit and SOA

Skomplikowana i droga ESB:

- ESB jest głownym komponentem
- Szybko staje przeciążony i drogi
- Próbuje robić „wszystko”
- Bardzo skomplikowany w utrzymaniu

Problems with Monolit and SOA

Brak narzędzi:

- dla SOA potrzebne krótke cyklu developmentu
- potrzeba w szybkim testowaniu
- nie ma narzędzi wspierających powyżej wskazane
- nie osiągnięto żadnego oszczedzania czasu w porównaniu z monolitem

Arhitektura mikroserwisowa

- Problemy z monolitem I SOA przeprowadziły do nowego paradygmatu
- Muśi być modularny, z prostym API
- Pojawił się w 2011, ale realnie otrzymał życie w 2014 roku.
- Martin Fowler “Microservices” - standart “de-facto”

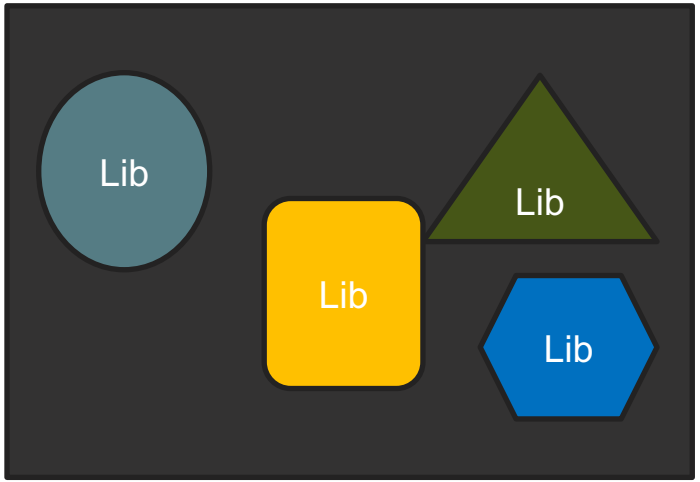
Cechy microserwisów

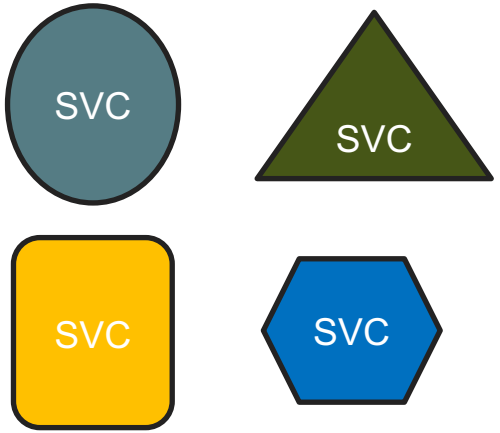
- Componentization via Services
- Organized Around Business Capabilities
- Products not Projects
- Smart Endpoints and Dumb Pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for Failure
- Evolutiaonary Design

Componentization via Services

- design modulowy zawsze jest dobrym pomysłem
- komponenty są częściamy ktorzy tworzą products
- Modularność może być osiągnięta poprzez:
    - biblioteki – wywolane bezpośrednio w procesie
    - serwisy – wywolane poza procesem (API, RPC)
- W mikroserwisach oczywyscie preferujemy serwisy a nie biblioteki

Componentization via Services





Organized Around Business Capabilities

Tradycyjne projekty:

- mają zespoły z poziomową odpiwiedzialnościa: UI, DB, API
- Bardzo powolna komunikacja pomiędzy zespolami
- Nie używają tej wspólnej terminologii
- **Nie mają wspólnych celow**

Organized Around Business Capabilities

Zalety:

- Szybki development
- Dobrze oznaczone granicę pomiedzy serwisami, a znaczy również ludzie wiedzą co im trzeba robić

Products not Projects

Tradycyjne projekty:

- Celem jest dostarczenie pracującego kodu
- Nie ma trwalych relacji z clientem
- Często w ogóle nie znają klienta
- Po dostarczeniu kodu zespół przechodzi do kolejnego projektu

Products not Projects

Mikroserwisowe projekty:

- Celem jest dostarczenie pracującego produktu
- Produkt wymaga stalego wsparci I bliskiej współpracy z klientem
- Zespół odpowiedzialny po dostarczeniu serwisu

“You build it, you run it”
W. Vogels, AWS CTO

Smart Endpoints and Dumb Pipes

SOA Projekty uzywają:

- ESB
- WS-* protokoły: WS-Security, WS-messaging, WS-discovery

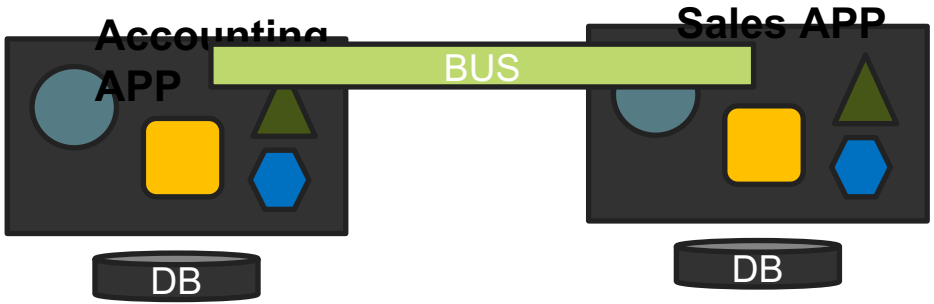
Pomiedzy serwisowa komunikacja stała zbyt skomplikowana i trudna do utrzymania

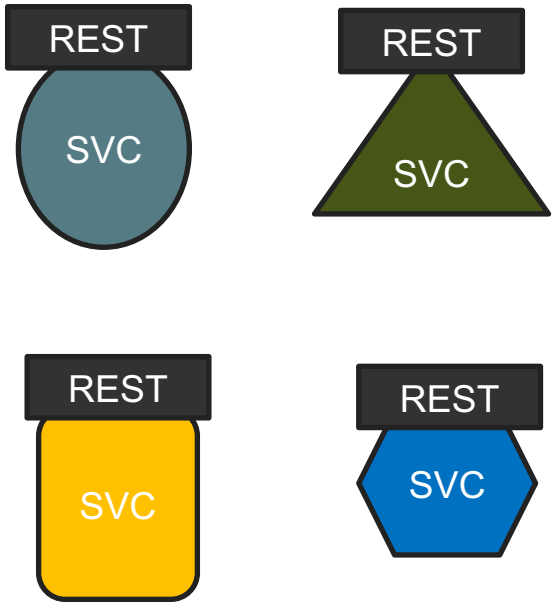
Smart Endpoints and Dumb Pipes

Systemy microserwisowe:

- Uzywają dumb pipes – prostych protokołów
- Nie wymyślają nowego, używają to co WEB proponuje teraz
- Zwykłe – REST API

Smart Endpoints and Dumb Pipes





Smart Endpoints and Dumb Pipes

Ważne:

- Bezposrednie połączenie pomiędzy serwisami nie jest dobrą opcją
- Lepsze rozwiązanie – używanie service discovery lub gateway
- Więcej protokolów w ostatnie lata (GraphQL, gRPC) – są skomplikowane

Co wygrajemy używając Smart Endpoint oraz Dumb Pipes:

- Przyszpeszamy development aplikacji
- Robimy aplikacje prostą do obslugi

Decentralized Governance

Klasyczne projekty:

- Jest standart do prawie wszystkiego

Mikroserwisy:

- Każdy zespół robi swoję decyzje
- Każdy zespół jest odpowiedzialny za swój serwis
- System poliglota

Decentralized Data Management

Klasyczne projekty:

- Jedna baza danych – przechowuje wszystkie dane komponentów systemu

Mikroserwisy:

- Każdy serwis może mieć swoją bazę danych

Infrastructure Automation

Azure DevOps

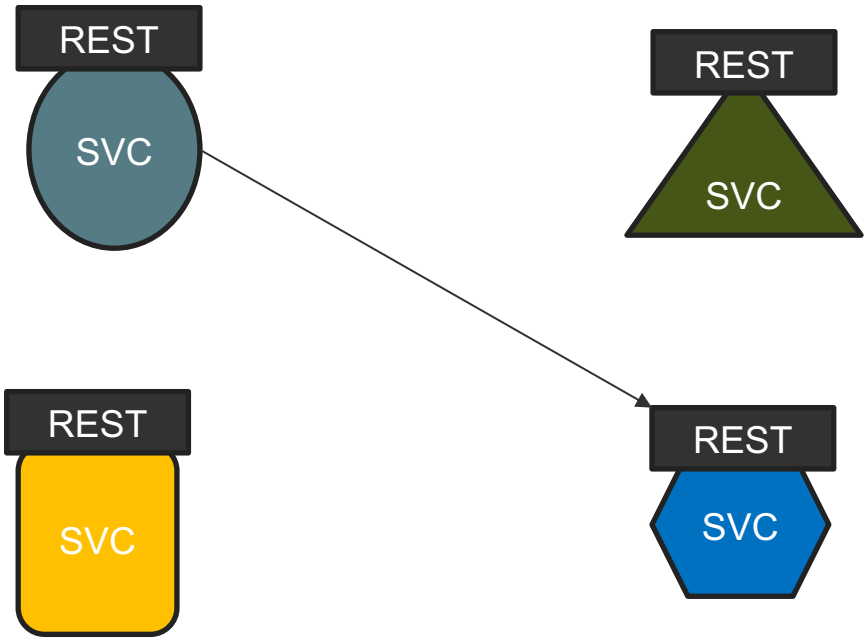
GitLab

Jenkins

Design for Failure

- W arhitekturze mikroserwisowej dużo procesów I dużo ruchu sieciowego
- Dużo może pójści nie tak
- Kod muśi ogarnać takie błędy w prawidłowy sposób
- Trzeba logować I monitorować

Design for Failure



Catch Exception

Retry

Log exception

Evolutionary Design

- Przejscie do microserwisów powinno być postępowe
- Nie trzeba rozwalać wszystkiego I zaczynać od nowa
- Zmieniamy kazdą część postępowo

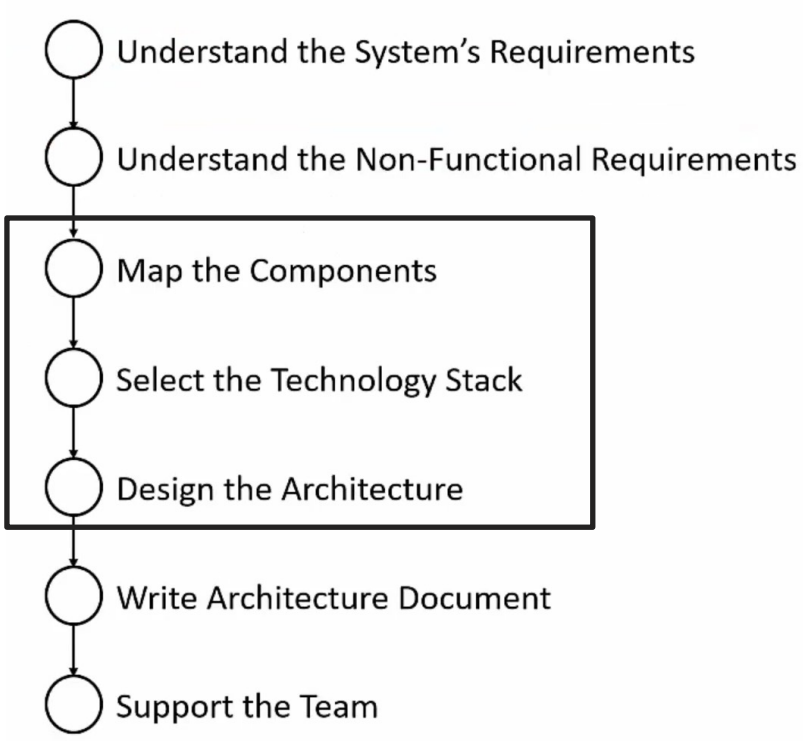
Problemy rozwiązywane za pomocą mikroserwisów

- Jedyna platforma technologiczna
- Nieelastyczny deployment
- Nieskuteczne zuzycie zasobów
- Duża skala I skomplikowanośc
- Drogie szyny serwisowe (ESB)
- Brak narzędzi

Projektowanie arhitektury mikroserwisowej

- Trzeba użyć metodologji
- Do not rush
- Więcej planuj, koduj mniej

Projektowanie arhitektury mikroserwisowej



Projektowanie arhitektury mikroserwisowej. Mapowanie komponentów

- Najbardziej ważny krok calego procesu
- Definiuje jak system będzie wyglądal w przyszlości
- Ustalony – nie łatwo zmienić

Mapowanie – definiowanie elementów systemu, z których tworzy się cały system.

Komponenty = Serwisy

Projektowanie arhitektury mikroserwisowej. Mapowanie komponentów

Mapowanie muśi bazować sie na:

- **Wymaganiach biznesowych**
- Autonomia funkcjonalna
- Jednostki danych
- Autonomia danych

Projektowanie arhitektury mikroserwisowej. Mapowanie komponentów

Mapowanie muśi bazować sie na:

- Wymaganiach biznesowych
- **Autonomia funkcjonalna**
- Jednostki danych
- Autonomia danych

Projektowanie arhitektury mikroserwisowej. Mapowanie komponentów

Mapowanie muśi bazować sie na:

- Wymaganiach biznesowych
- Autonomia funkcjonalna
- **Jednostki danych (zamowienia, towary)**
- Autonomia danych

Projektowanie arhitektury mikroserwisowej. Mapowanie komponentów

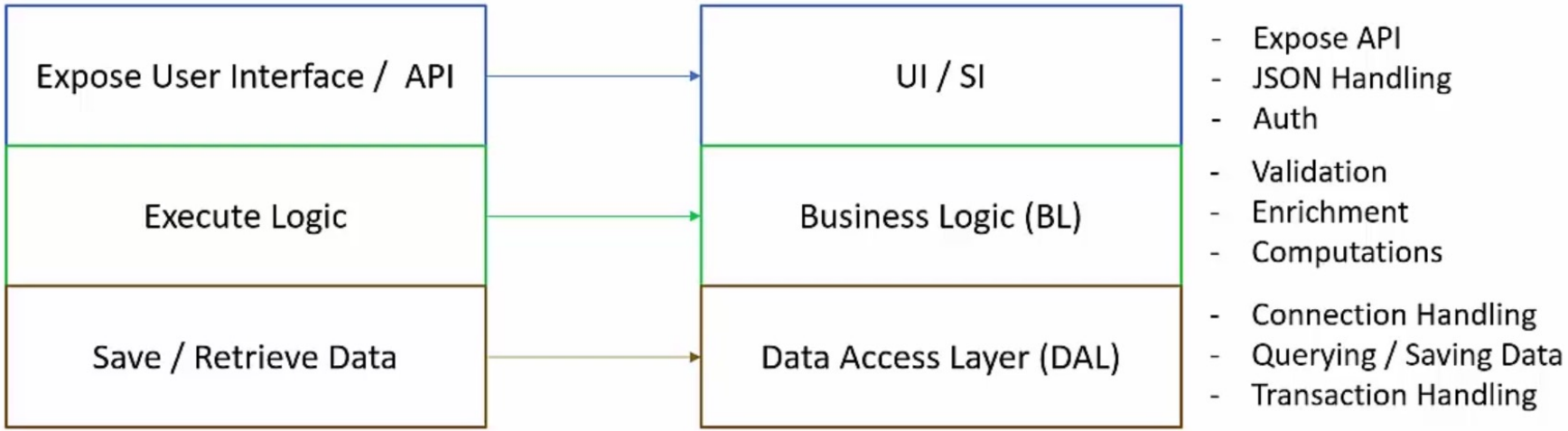
Mapowanie muśi bazować sie na:

- Wymaganiach biznesowych
- Autonomia funkcjonalna
- Jednostki danych (zamowienia, towary)
- **Autonomia danych**

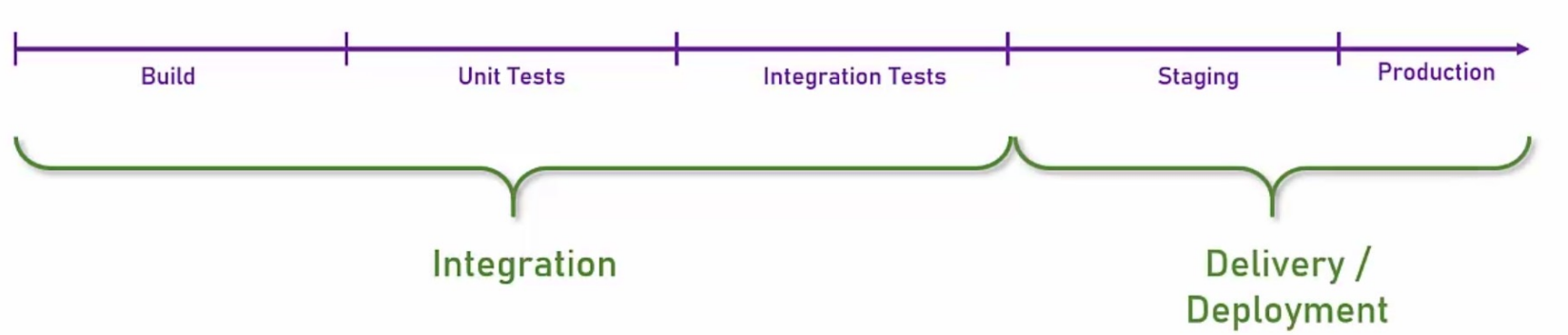
Projektowanie arhitektury mikroserwisowej. Wybor stosu technologicznego

	App Types	Type System	Cross Platform	Community	Performance	Learning Curve
.NET	All	Static	No	Large	OK	Long
.NET Core	Web Apps, Web API, Console, Service	Static	Yes	Medium and growing rapidly	Great	Long
Java	All	Static	Yes	Huge	OK	Long
node.js	Web Apps, Web API	Dynamic	Yes	Large	Great	Medium
PHP	Web Apps, Web API	Dynamic	Yes	Large	OK -	Medium
Python	All	Dynamic	Yes	Huge	OK -	Short

Projektowanie arhitektury mikroserwisowej.



Deploying Microservices. Integration and Delivery



Pozwala:

- Przyspieszyć release
- Zwiększyć niezawodność
- Na raportowanie

Deploying Microservices. Kontenery

Klasyczny deployment:

Kod skopiowany I zbuildowany na prodzie

Są problemy/bugi znalezione na prodzie, zamiast test/stage srodowisk

Deploying Microservices. Kontenery

Kontenery:

Cienki model opakowania

Pakuje aplikacje, zależności oraz pliki konfiguracyjne?

Może być kopiowany pomiedzy serwerami

Używa hostowego systemu operacyjnego

Deploying Microservices. Kontenery

Dla czego “tak” dla kontenerów:

Przewidywalność

Wydajność

Gęstość

Dla czego “nie” dla kontenerów:

izolacja

Praktyki

- Infrastruktura mikroserwisowa na przykładzie K8s
- In-memory DBs
- Queues and brokers

Przydatne linki

https://martinfowler.com/articles/microservices.html

https://encyclopedia2.thefreedictionary.com/WS*+protocols

Dzieki

Pytania, zadania domowe:

oleksii.tsyganov@gmail.com