Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інформаційні системи

Викладач: к.т.н., доц. Саяпіна Інна Олександрівна

План заняття:

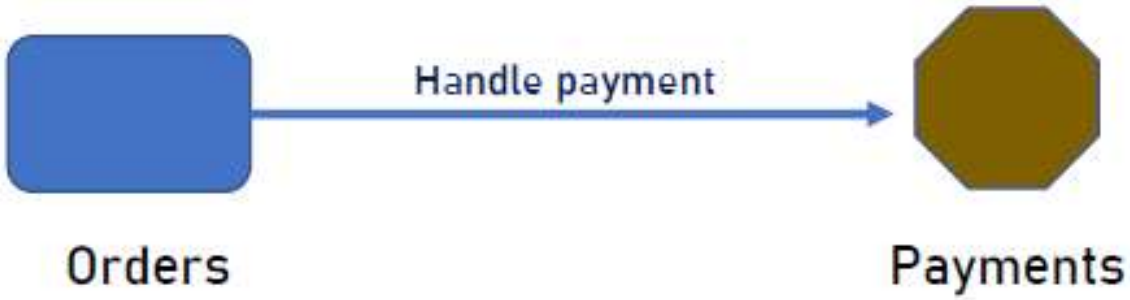
- Мікросервісна архітектура (Microservice Architecture):
  - Процес проєктування архітектури
    - Визначення шаблонів зв’язку
- Архітектура, керована подіями (Event Driven Architecture):
  - Поняття події (Event)
  - Архітектура, керована подіями (Event Driven Architecture)

Визначення шаблонів зв’язку

- Ефективний зв’язок між службами має вирішальне значення
- Важливо вибрати правильний шаблон спілкування
  - Використання неправильного шаблону може призвести до повільної роботи, некерованості системи та поганого розподілу ролей
- Основні шаблони:
  - 1-to-1 Sync
  - 1-to-1 Async
  - Pub-Sub / Event Driven

1-to-1 Async

- Сервіс викликає інший сервіс і продовжує роботу
- Не чекає на відповідь за принципом «Вистрелив і забув».
- Використовується в основному, коли перша служба хоче передати повідомлення іншій службі



1-to-1 Async

Переваги

- Продуктивність

Недоліки

- Потрібні додаткові налаштування
- Складна обробка помилок

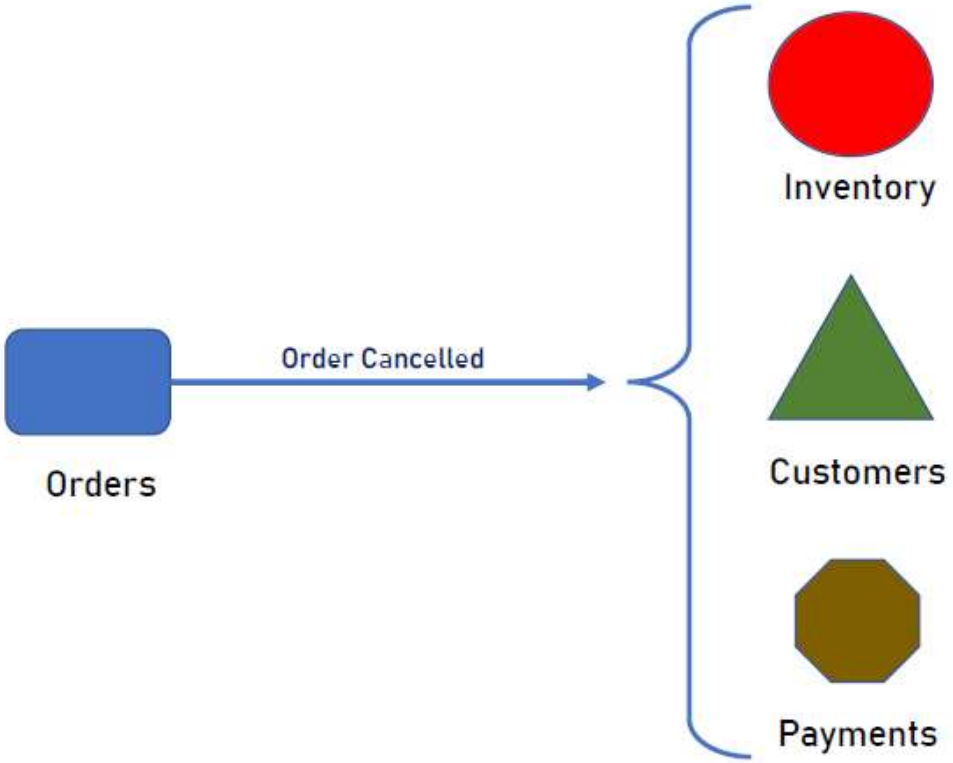
1-to-1 Async

- Реалізація

Pub-Sub / Event Driven

- Сервіс хоче повідомити інші сервіси про щось
- **Сервіс навіть не знає, скільки сервісів його слухає**
- Не чекає на відповідь «Вистрелив і забув».
- Використовується в основному, коли перший сервіс хоче повідомити про важливу подію в системі

Pub-Sub / Event Driven



Pub-Sub / Event Driven

Переваги

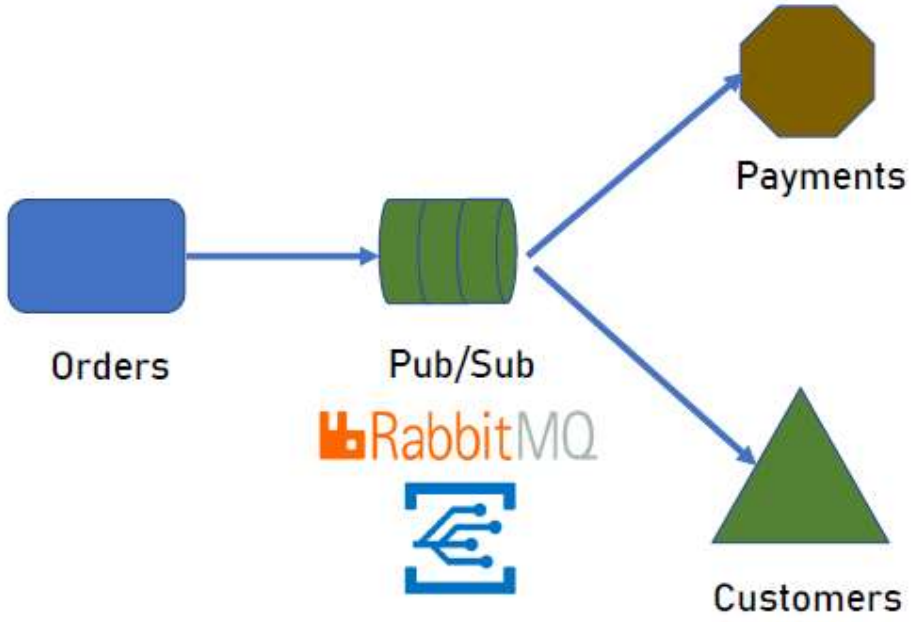
- Продуктивність
- Сповіщення кількох сервісів одночасно

Недоліки

- Потрібні додаткові налаштування
- Складна обробка помилок
- Може викликати навантаження

Pub-Sub / Event Driven

- Реалізація



Висновки. Визначення шаблонів зв’язку

- Вибір правильного шаблону зв’язку має вирішальне значення
- Впливає на:
  - Продуктивність
  - Обробку помилок
  - Потік комунікації
- Переробити заново майже неможливо

Вибір стеку технологій сервісів

- Децентралізоване управління дозволяє вибирати різні стеки технологій для кожної служби
- Зосередимося на бекенд-платформі та платформах зберігання
- Немає мети ділити на «правильно» чи «неправильно»
- Конкретне рішення приймається на основі вагомих доказів

Платформа розробки

	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

Сховище даних

- 4 типи сховища даних:
  - Реляційна база даних
  - База даних NoSQL
  - Кеш
  - Сховище об'єктів (Object Store)

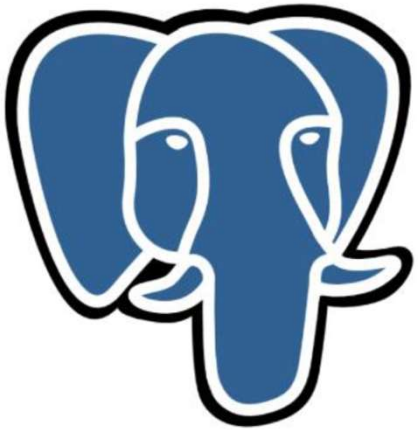
Реляційна база даних

- Зберігає дані в таблицях
- Таблиці мають чіткий набір стовпців

Column Name	Type	Nullable?
OrderId	Numeric	No
OrderDate	DateTime	No
CustomerId	Numeric	No
DeliveryAddress	String	No

Column Name	Type	Nullable?
OrderItemId	Numeric	No
OrderId	Numeric	No
ItemName	String	No
Quantity	Numeric	No

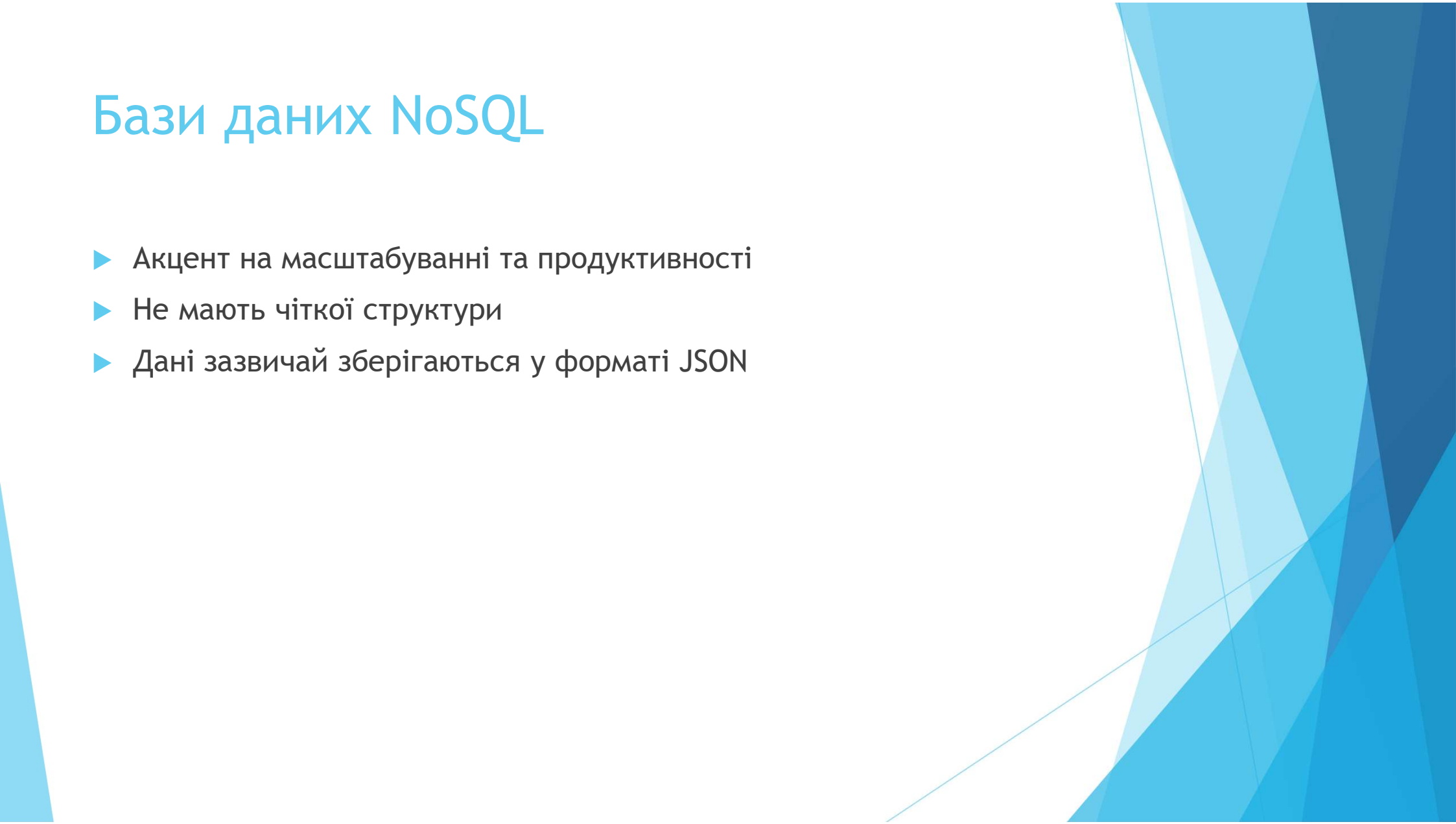
Реляційна база даних



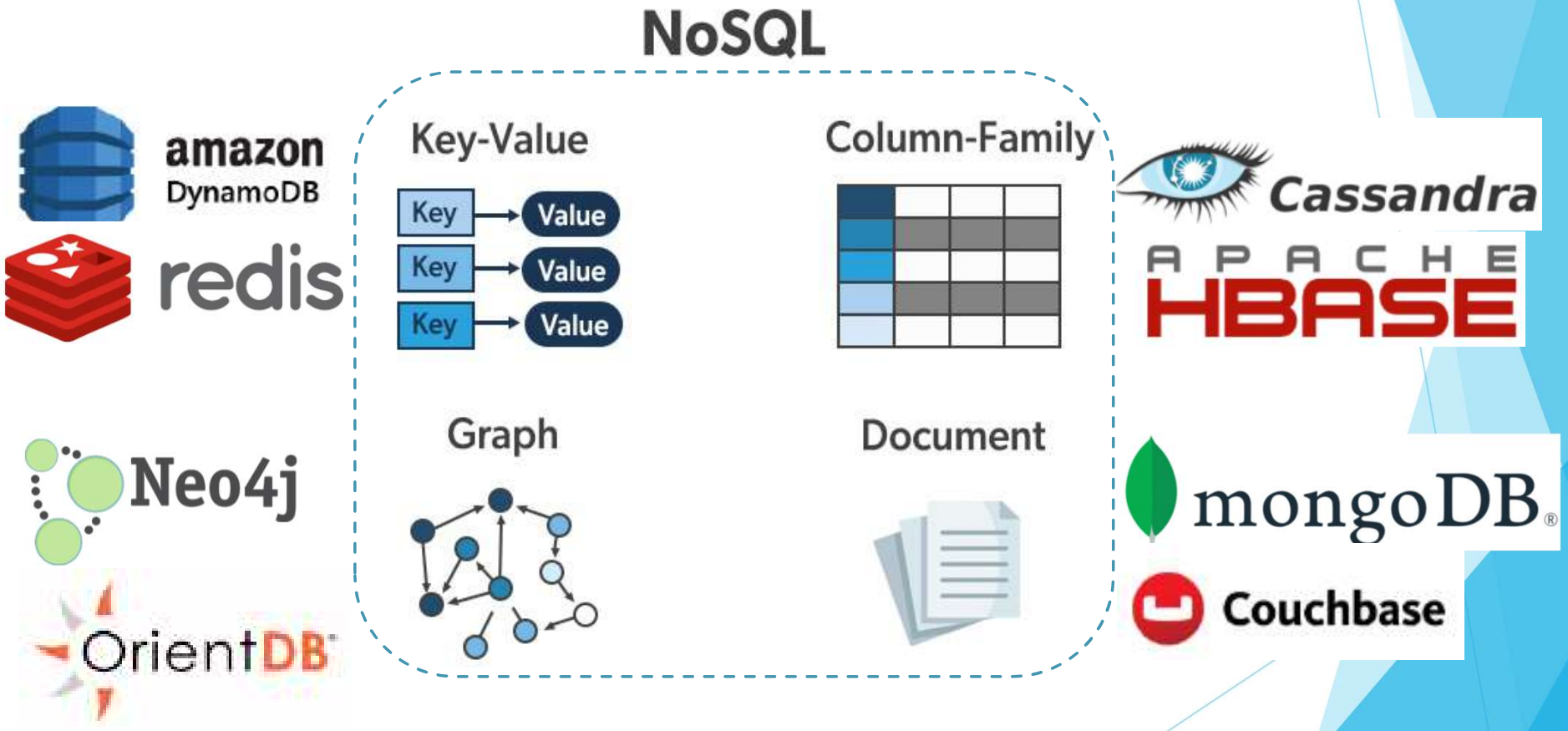
PostgreSQL

Бази даних NoSQL

- Акцент на масштабуванні та продуктивності
- Не мають чіткої структури
- Дані зазвичай зберігаються у форматі JSON

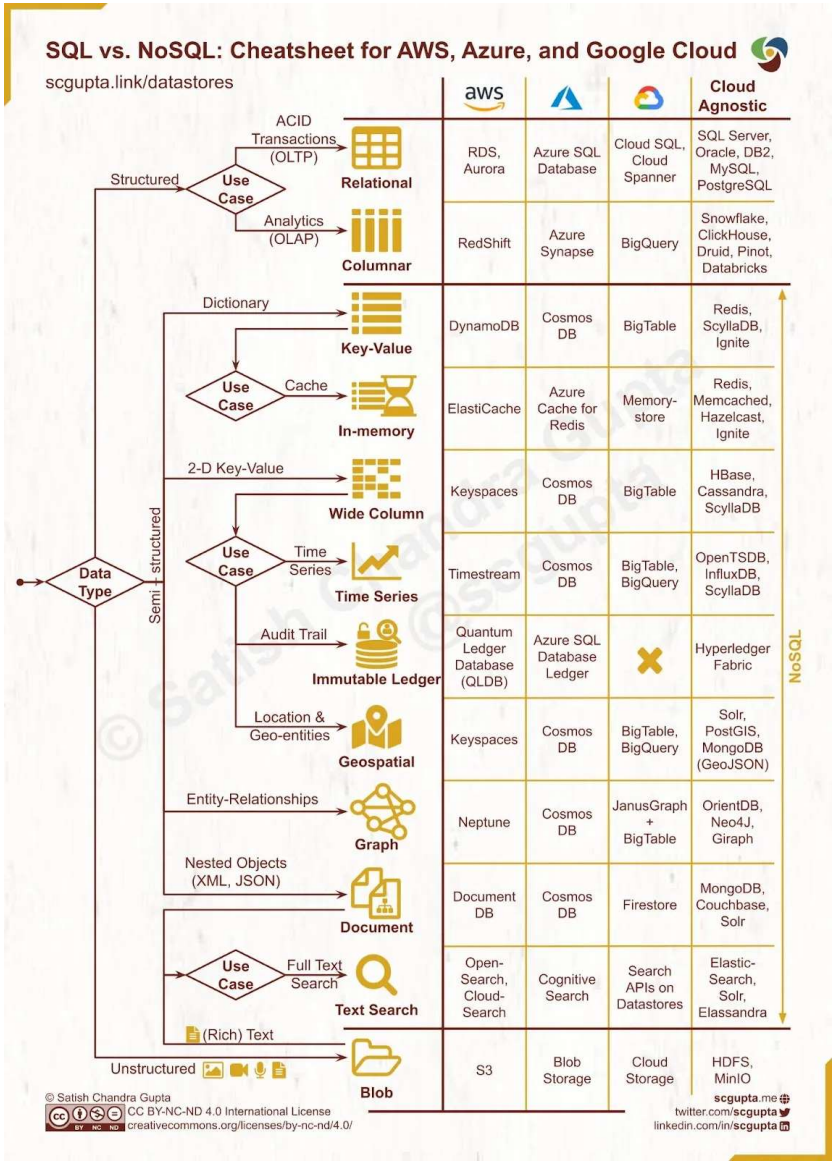


Бази даних NoSQL



Кеш

- Зберігає дані в пам'яті для швидкого доступу
- Розподіляє дані між вузлами
- Використовує власний протокол
- Зберігає серіалізовані об'єкти


SQL vs. NoSQL: Cheatsheet for AWS, Azure, and Google Cloud
scgupta.link/datastores
aws
Cloud Agnostic
Data Type
Structured
Semi - structured
Unstructured
Use Case
ACID Transactions (OLTP)
Relational
Analytics (OLAP)
Columnar
Dictionary
Key-Value
Cache
In-memory
2-D Key-Value
Wide Column
Time Series
Time Series
Audit Trail
Immutable Ledger
Location & Geo-entities
Geospatial
Entity-Relationships
Graph
Nested Objects (XML, JSON)
Document
Full Text Search
Text Search
(Rich) Text
Blob
RDS, Aurora
Azure SQL Database
Cloud SQL, Cloud Spanner
SQL Server, Oracle, DB2, MySQL, PostgreSQL
RedShift
Azure Synapse
BigQuery
Snowflake, ClickHouse, Druid, Pinot, Databricks
DynamoDB
Cosmos DB
BigTable
Redis, ScyllaDB, Ignite
ElastiCache
Azure Cache for Redis
Memory-store
Redis, Memcached, Hazelcast, Ignite
Keyspaces
Cosmos DB
BigTable
HBase, Cassandra, ScyllaDB
Timestream
Cosmos DB
BigTable, BigQuery
OpenTSDB, InfluxDB, ScyllaDB
Quantum Ledger Database (QLDB)
Azure SQL Database Ledger
Hyperledger Fabric
Keyspaces
Cosmos DB
BigTable, BigQuery
Solr, PostGIS, MongoDB (GeoJSON)
Neptune
Cosmos DB
JanusGraph + BigTable
OrientDB, Neo4J, Giraph
Document DB
Cosmos DB
Firestore
MongoDB, Couchbase, Solr
Open-Search, Cloud-Search
Cognitive Search
Search APIs on Datastores
Elastic-Search, Solr, Elassandra
S3
Blob Storage
Cloud Storage
HDFS, MinIO
NoSQL
© Satish Chandra Gupta
CC BY-NC-ND 4.0 International License
creativecommons.org/licenses/by-nc-nd/4.0/
scgupta.me
twitter.com/scgupta
linkedin.com/in/scgupta

Object storage (Сховище об’єктів)

- Зберігає неструктуровані великі дані
  - Документи
  - Фотографії
  - Файли

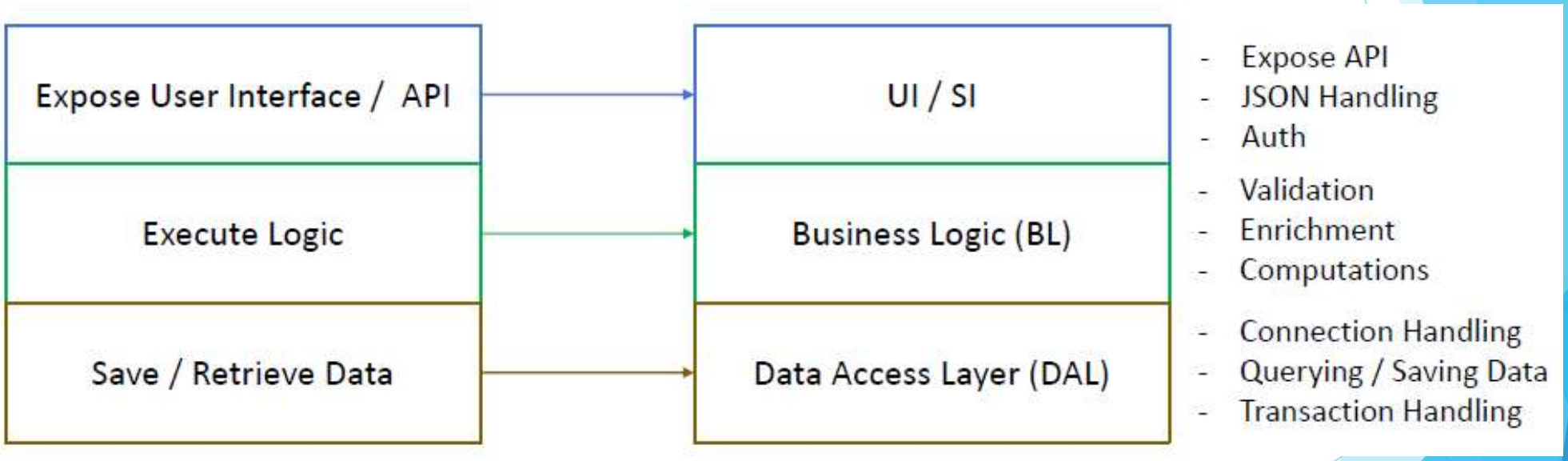
- Приклади:
  - **Amazon S3** (Simple Storage Service): служба зберігання об’єктів Amazon Web Services (AWS), яка забезпечує безпечне, довговічне та масштабоване сховище для будь-яких типів даних.
  - **Microsoft Azure Blob Storage**: рішення Microsoft Azure для зберігання об’єктів, яке дозволяє користувачам зберігати та отримувати доступ до великих обсягів неструктурованих даних.
  - **Google Cloud Storage**: рішення Google для зберігання об’єктів, яке забезпечує масштабоване, безпечне та економічно ефективне зберігання будь-яких типів даних.
  - **IBM Cloud Object Storage**.

Проєктування архітектури

- Архітектура сервісу не відрізняється від звичайного програмного забезпечення
- На основі - парадигма шарів

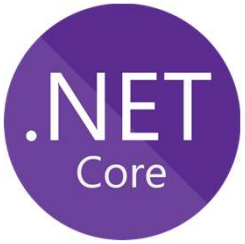
Шари

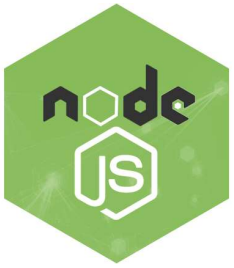
- Представляють горизонтальну функціональність

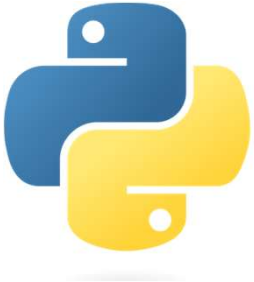


Навіщо вивчати Event Driven Architecture (EDA)?

- Є еволюцією архітектури мікросервісів
- Знаходить дуже широку популярність
- Є платформенно незалежною








- Дуже зручна у масштабуванні

Наслідки погано спроєктованої EDA

- Повільність
- Неможливість обслуговування
- Висока вартість утримання

Наслідки погано спроєктованої EDA

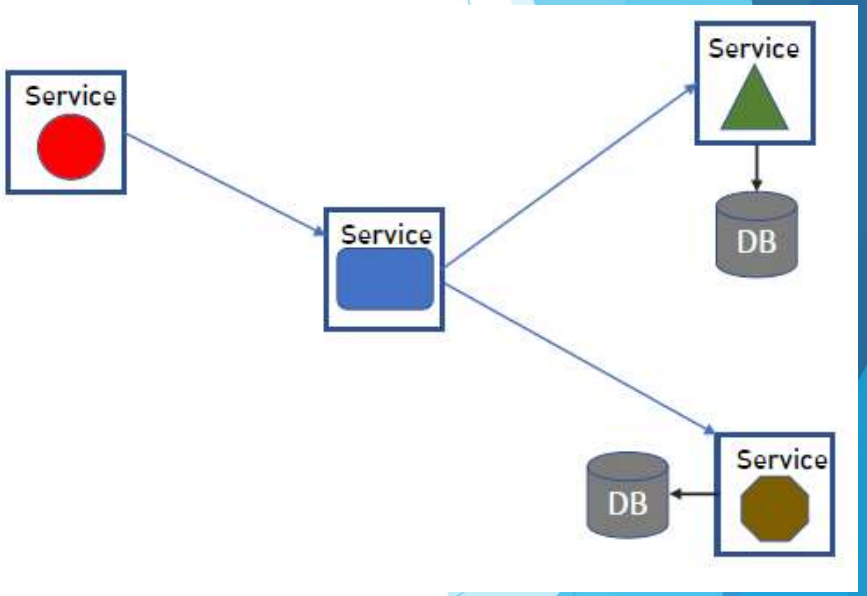
- Подія - основна одиниця архітектури, керованої подіями
- Події або Керована подіями архітектура є наслідком еволюції інших архітектур

Мікросервісна архітектура

- В основі слабко зв’язані сервіси
- Кожний сервіс є окремим процесом
- Легкі протоколи зв'язку
- Відсутність залежності від платформи при комунікації між службами
- Замінює дві застарілі архітектури:
  - Монолітну (Monolith)
  - Сервіс-орієнтовану (Service-Oriented Architecture - SOA)

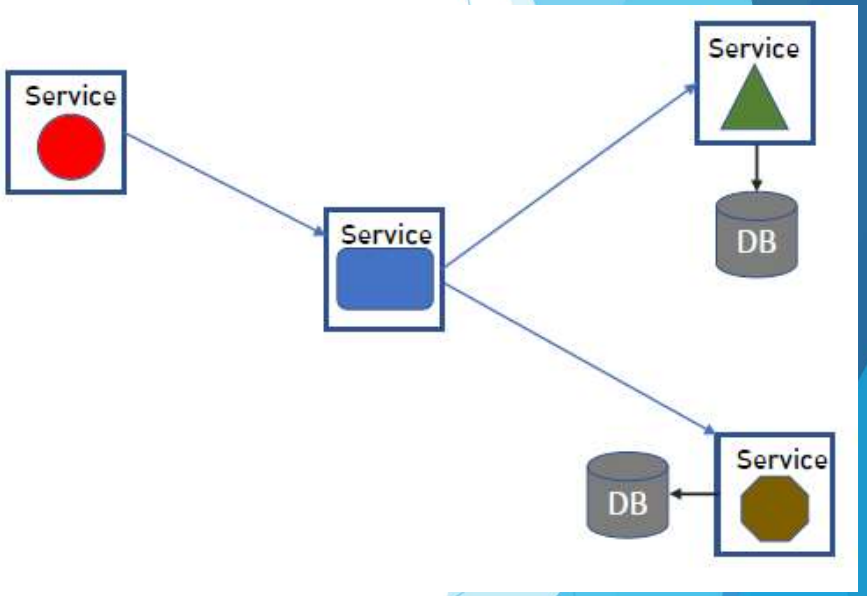
Мікросервісна архітектура

- розподілені сервіси
- деякі з них мають власну базу даних, хоча це не обов'язково
- вони спілкуються один з одним за допомогою мережі.
- кожен сервіс розгортається, як незалежний процес



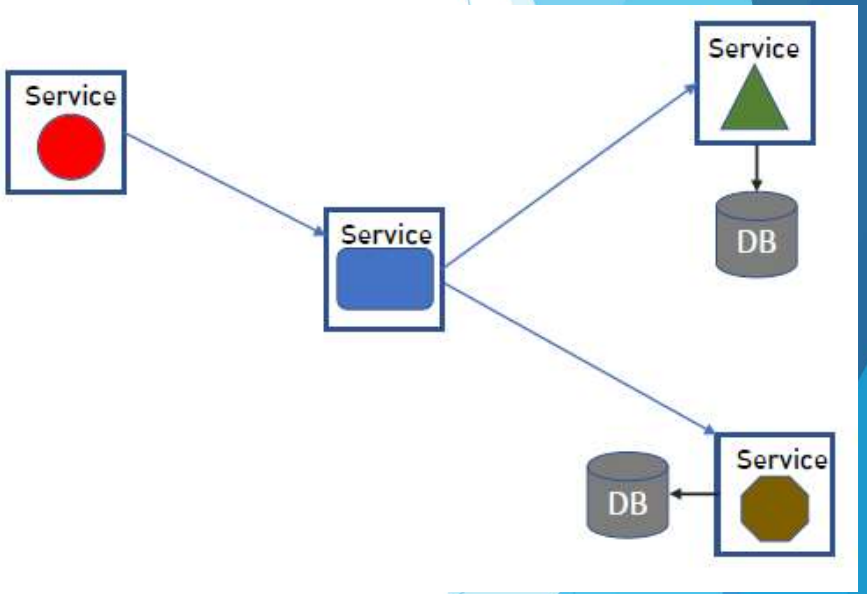
Комунікація між мікросервісами

- Одна з найважливіших частин архітектури мікросервісів
- Диктує продуктивність, масштабованість, можливість впровадження та інше
- Керована подіями архітектура бере на себе комунікацію між сервісами



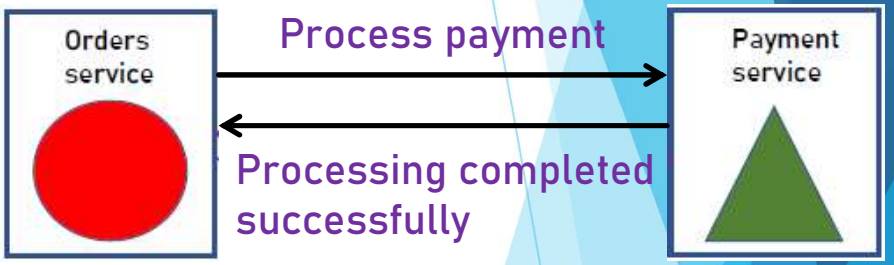
Команда та запит

- Класичний зв'язок між сервісами:
  - Відправити команду
  - Надіслати апит на дані



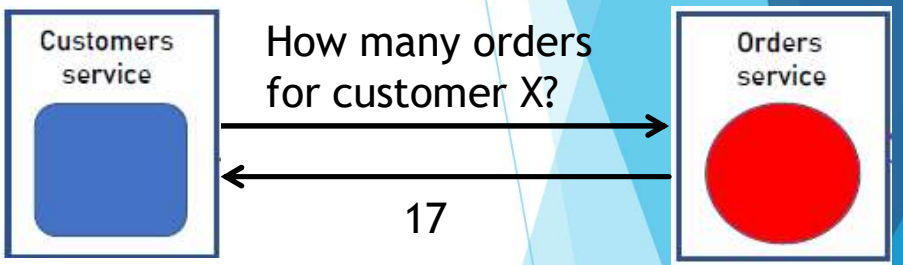
Команда

- Сервіс просить інший сервіс щось зробити
- Приклад. Два сервіси: сервіс замовлень і сервіс оплати, і сервіс замовлень просить платіжний сервіс обробити платіж.
- На команду може бути відповідь, як правило, індикатор успіху чи невдачі. Це опціонально.



Запит

- Сервіс запитує у іншого сервісу дані
- На запит завжди є відповідь , що містить дані



Команда та запит

- Основні характеристики:

Команда

- Щось виконує
- Зазвичай синхронна дія
- Іноді повертає відповідь
- Викликаючий сервіс повинен знати хто обробляє команду

Запит

- Отримує дані
- Майже завжди синхронна дія
- Завжди повертає відповідь
- Викликаючий сервіс повинен знати хто обробляє запит

Проблеми з командами та запитами

Команда	Запит
Щось виконує	Отримує дані
Зазвичай синхронна дія	Майже завжди синхронна дія
Іноді повертає відповідь	Завжди повертає відповідь
Викликаючий сервіс повинен знати хто обробляє команду	Викликаючий сервіс повинен знати хто обробляє запит

- Продуктивність
  - Синхронна дія = служба, що викликає, очікує завершення команди / запиту => Потенціальне зниження продуктивності
- Зчеплення
  - Викликаючий сервіс викликає певну службу => Якщо викликаний сервіс змінюється, викликаючий сервіс також має змінитися => Більше роботи, більше обслуговування
- Масштабованість
  - Служба, що викликає, повинна знати, хто обробляє команду => викликається конкретний сервіс і якщо він наразі зайнятий, продуктивність буде погіршуватися => масштабувати сервіс можливо, але непросто

Подія

- Вказує на те, що в системі щось сталося

- На подію не потрібна відповідь

Основні характеристики події

- Вказує на те, що в системі щось сталося
- Асинхронна
- Ніколи не повертає відповідь
- Викликаючий сервіс не знає, хто обробляє подію

Два види подій

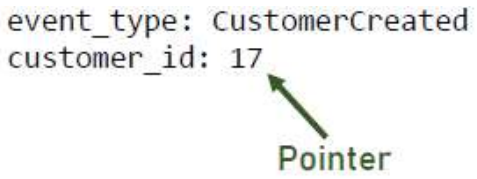
Повна (Complete)

- Містить усі необхідні дані
- Зазвичай це дані сутності
- Для обробки події не потрібні додаткові дані
- Приклад

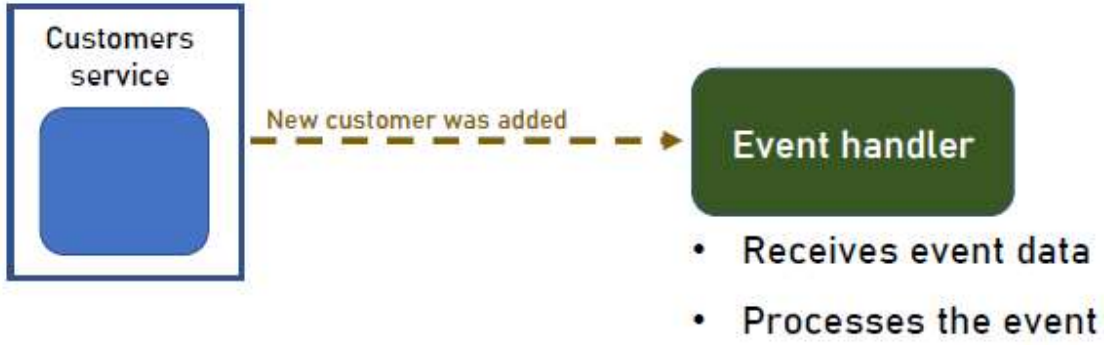
```
event_type: CustomerCreated
customer_id: 17
first_name: David
last_name: Jones
join_date: 2022-03-15
```

Вказівник (Pointer)

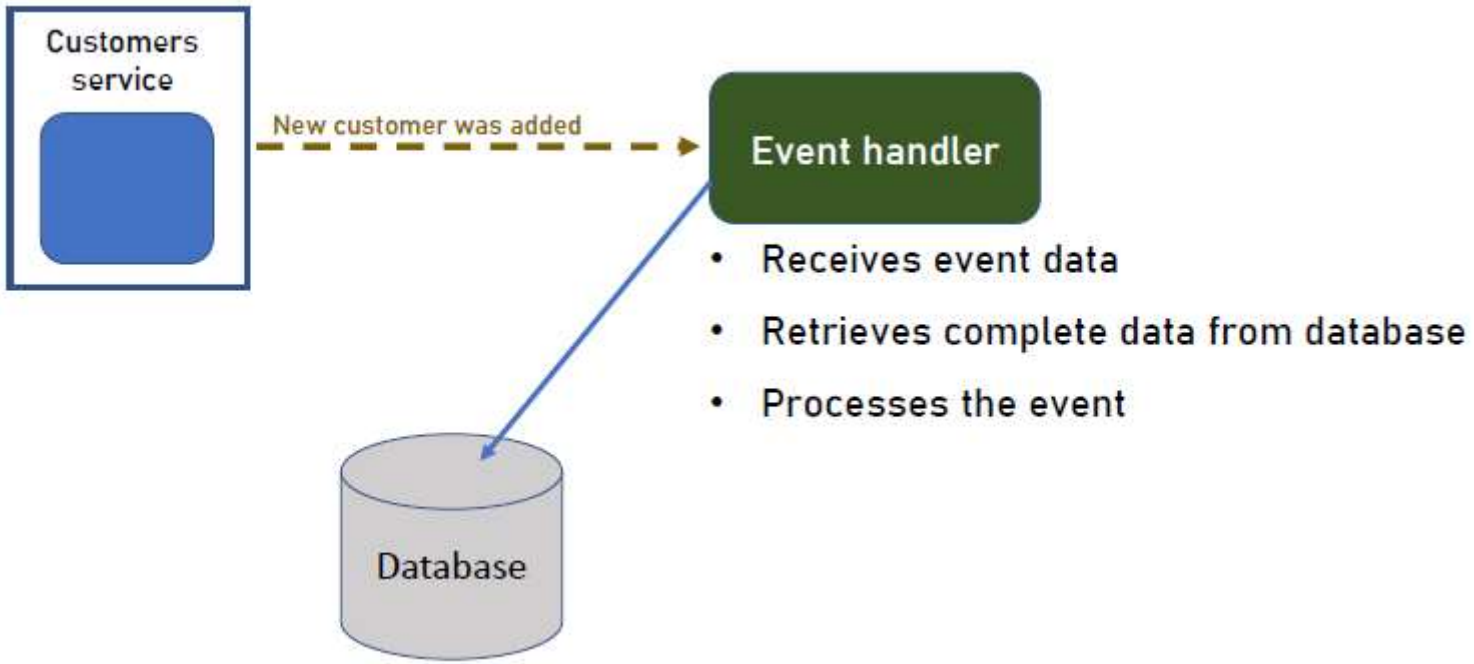
- Містить вказівник на повні дані сутності
- Повні дані зазвичай зберігаються в базі даних
- Обробнику подій потрібен доступ до бази даних, щоб отримати повні дані
- Приклад



Обробка повної події



Обробка вказівника події



Повна подія vs Вказівник події

- Коли що використовується?

Повна (Complete)

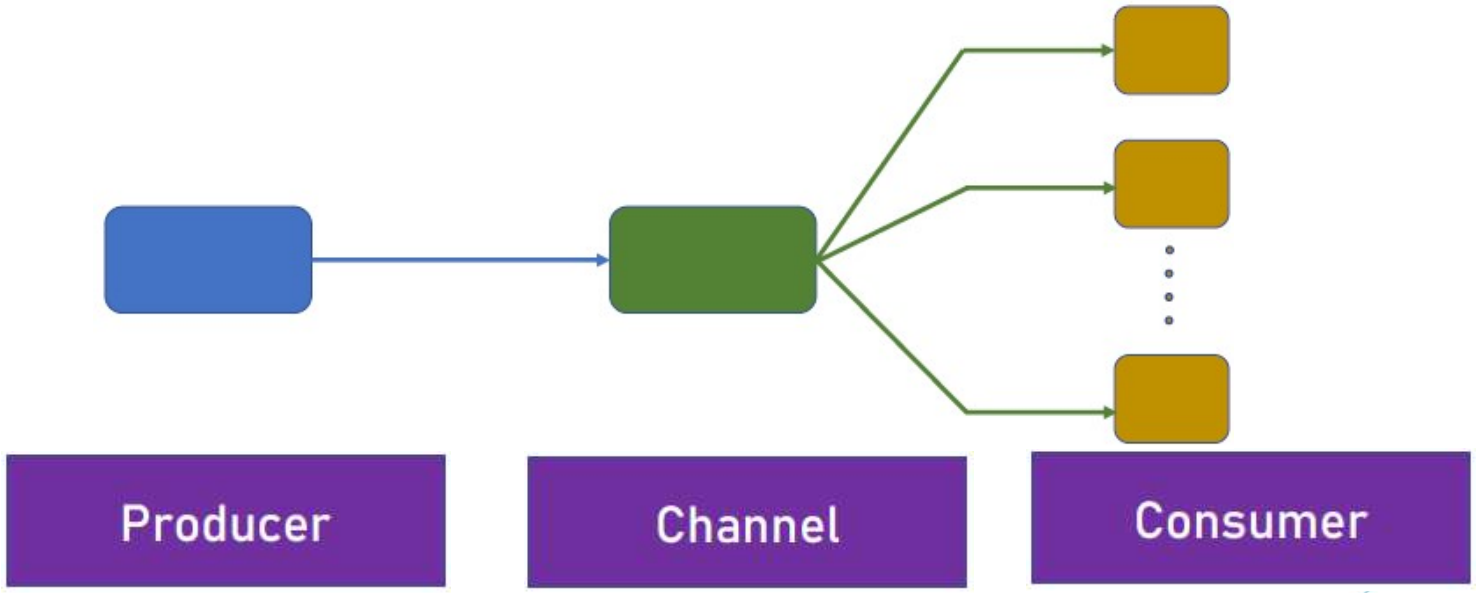
- Зручніший підхід
- Робить подію повністю автономною
- Може використовуватись за межами системи

Вказівник (Pointer)

- Використовується коли:
  - Дані великого обсягу
  - Потрібно переконатися, що дані є актуальними
  - Припускається, що база даних є єдиним «джерелом істини»

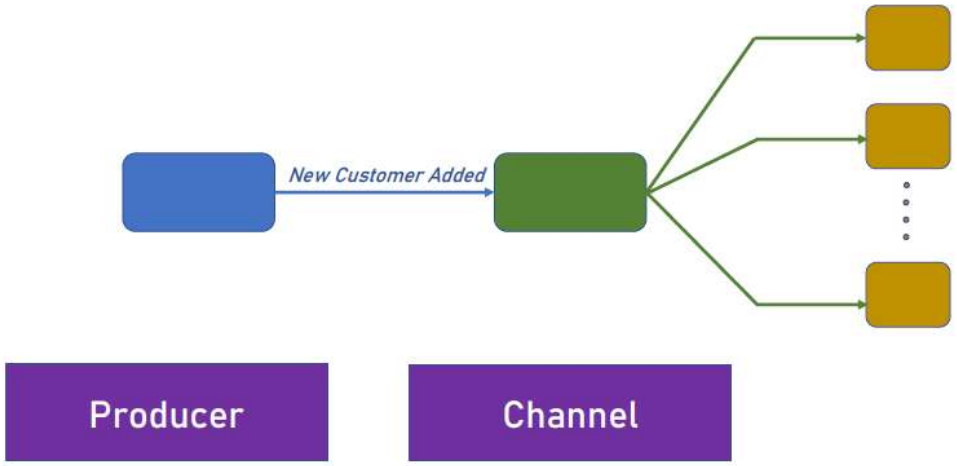
Event Driven Architecture (EDA)

- Парадигма архітектури програмного забезпечення, яка використовує події як засіб зв’язку між службами
- Має три основні компоненти:



Producer

- Компонент/служба, що надсилає подію
- Часто називають Publisher
- Зазвичай подія повідомляє про щось, що зробив компонент
- Приклади:
  - Обслуговування клієнтів: --> Подія додавання нового клієнта
  - Служба інвентаризації: --> Товар розпродано



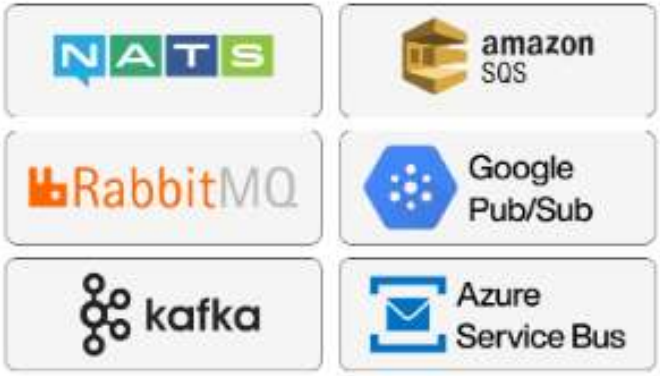
Producer

- Спосіб виклику каналу, що застосовується, залежить від каналу
- Зазвичай використовується спеціальний SDK, розроблений постачальником каналу
- Використовує якийсь мережевий виклик, зазвичай зі спеціалізованими портами та власним протоколом
- Наприклад: RabbitMQ слухає порт 5672 і використовує протокол AMQP
- Producer може бути розроблений за допомогою будь-якої мови програмування

Channel

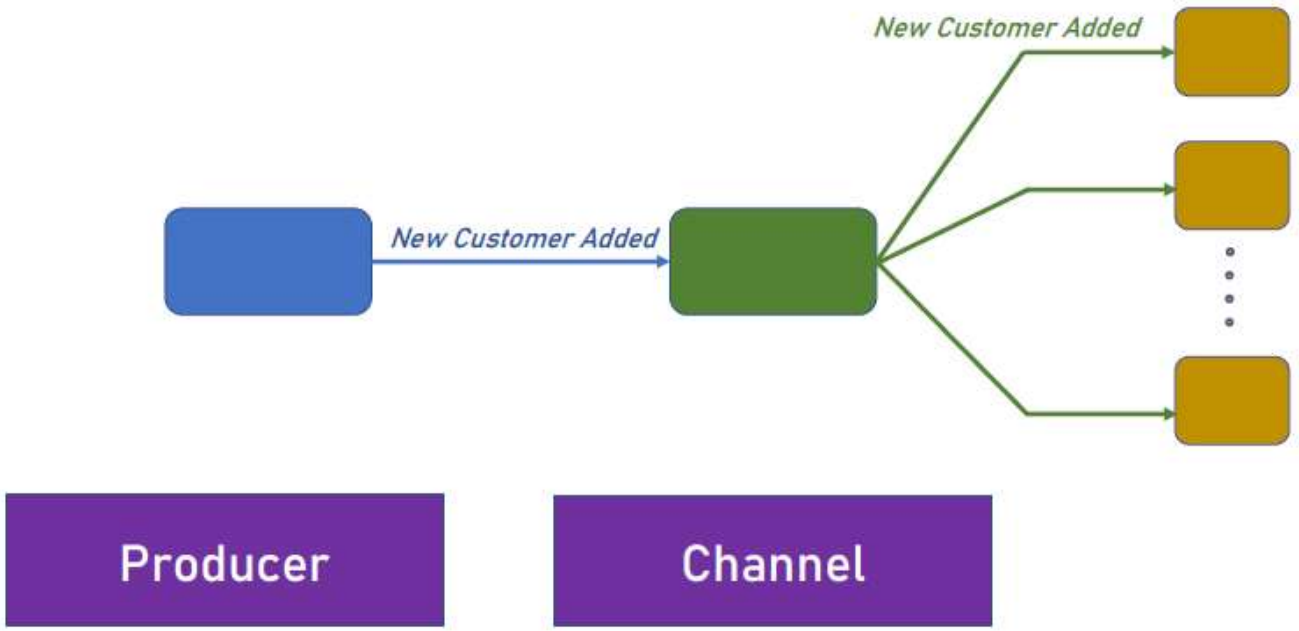
- Найважливіший компонент керованої подіями архітектури
- Відповідає за розповсюдження подій між зацікавленими сторонами
- Канал розміщує подію в спеціалізованій черзі (Queue), яку часто називають Topic або Fanout
- Споживачі (Consumers) слухають цю чергу і ловлять подію
- Програмні системи, які діють як посередники для зв'язку між розподіленими програмами, службами та системами, називаються **брокерами повідомлень (Message Brokers)**. Вони призначені для керування маршрутизацією, перетворенням і доставкою повідомлень між різними компонентами розподіленої програми.

Найрозповсюджені



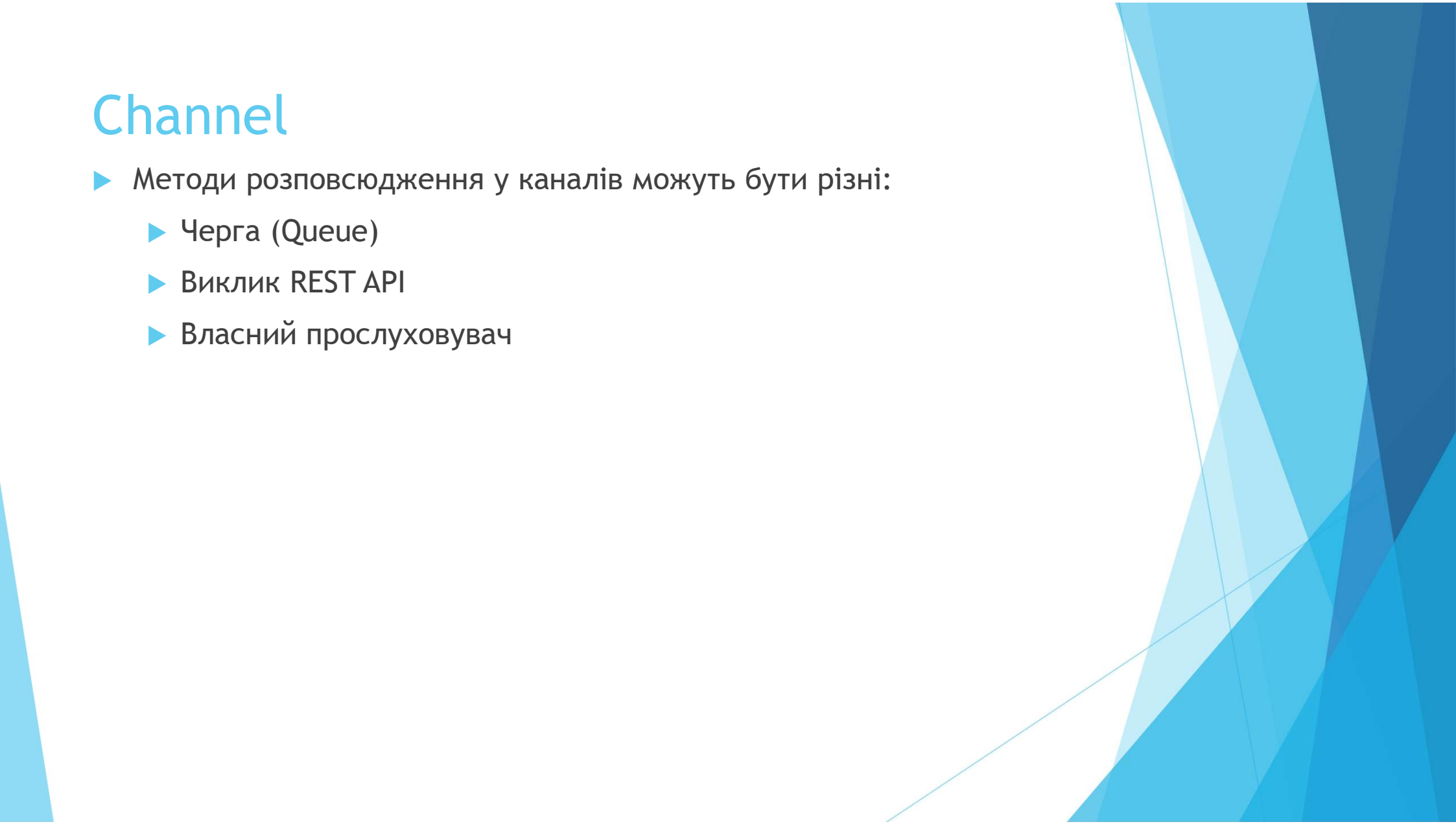
Channel

- Канал розповсюджує події для споживачів (Consumers)



Channel

- Методи розповсюдження у каналів можуть бути різні:
  - Черга (Queue)
  - Виклик REST API
  - Власний прослуховувач

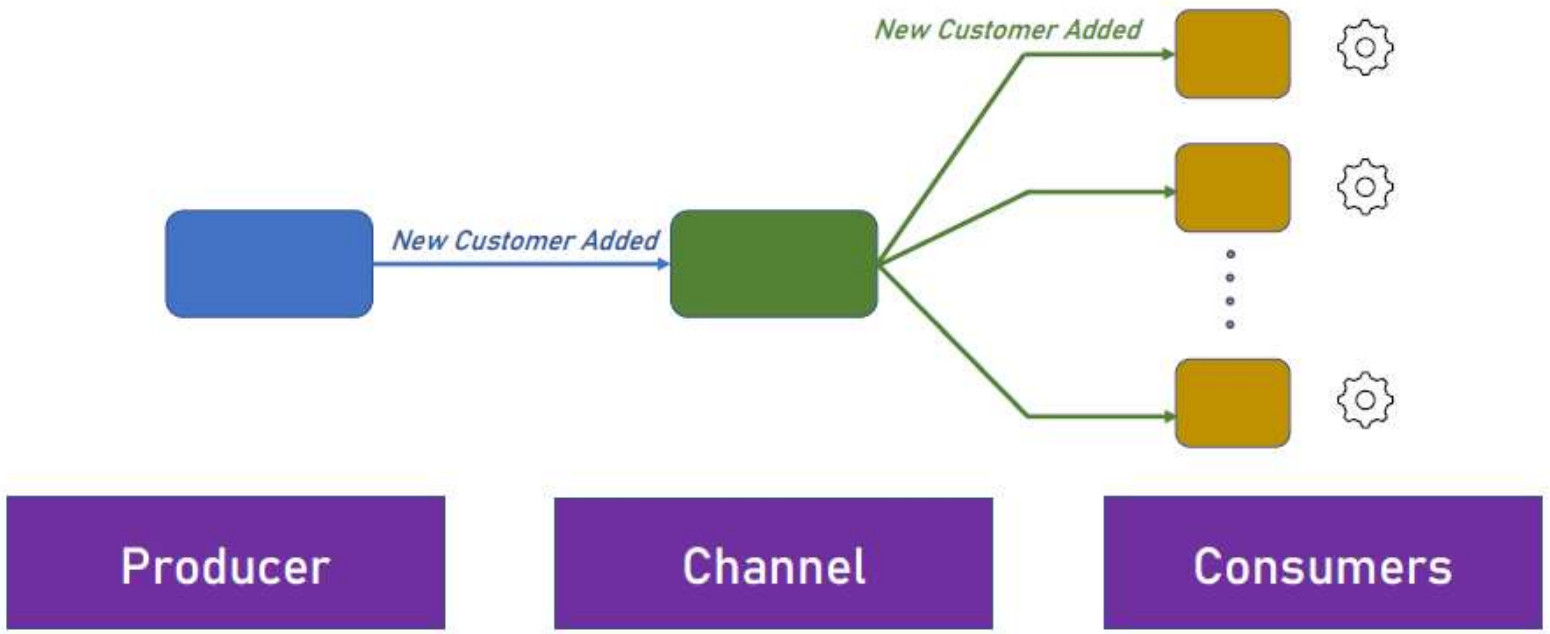


Consumer

- Компонент, який отримує подію, надіслану Producer`ом і розповсюджену каналом
- Може бути розроблений будь-якою мовою, сумісною з бібліотеками каналу
- Обробляє подію
- Іноді надсилає зворотнє повідомлення про завершення обробки (Ack)

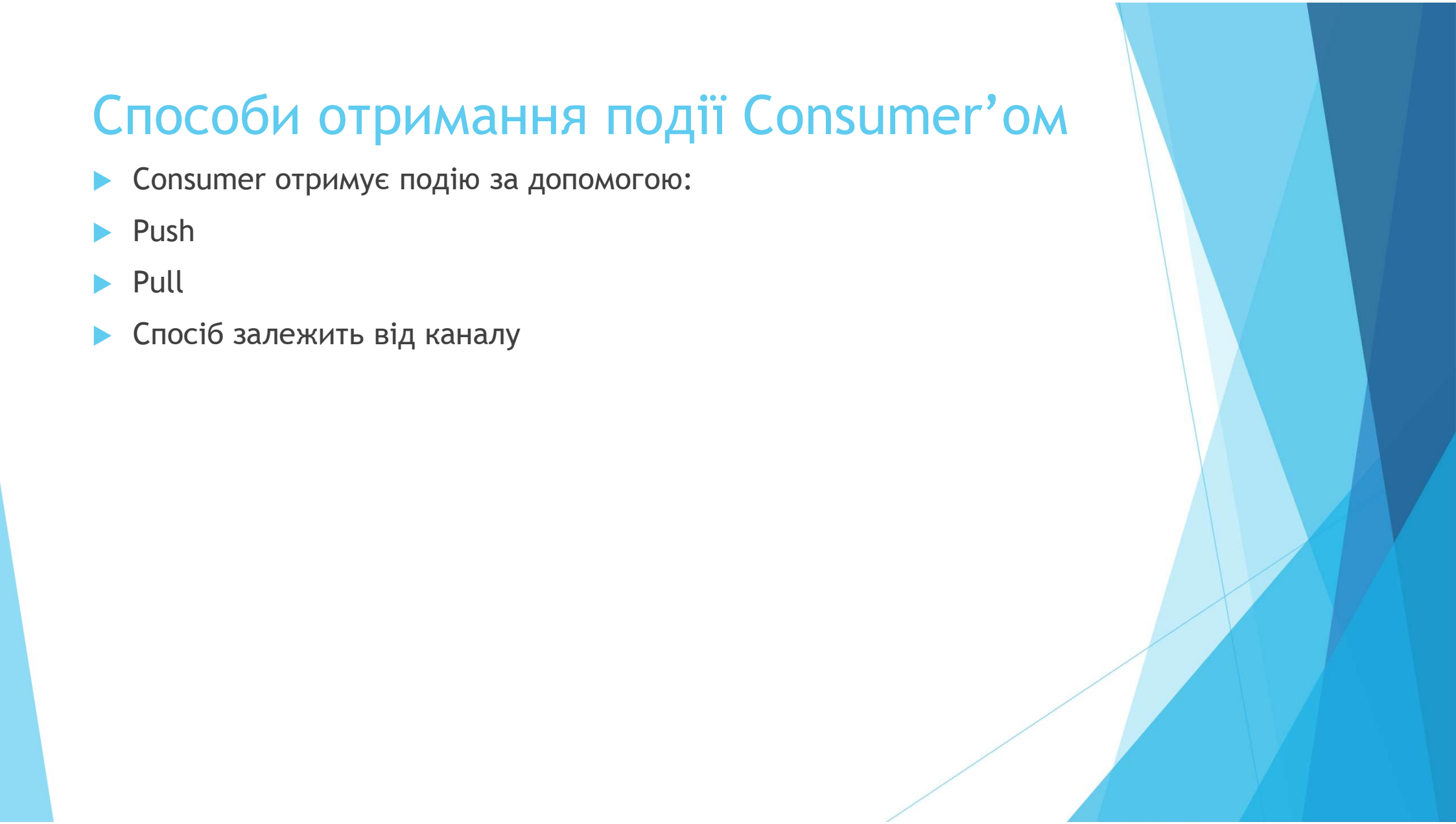
Consumer

- Consumer отримує та обробляє події



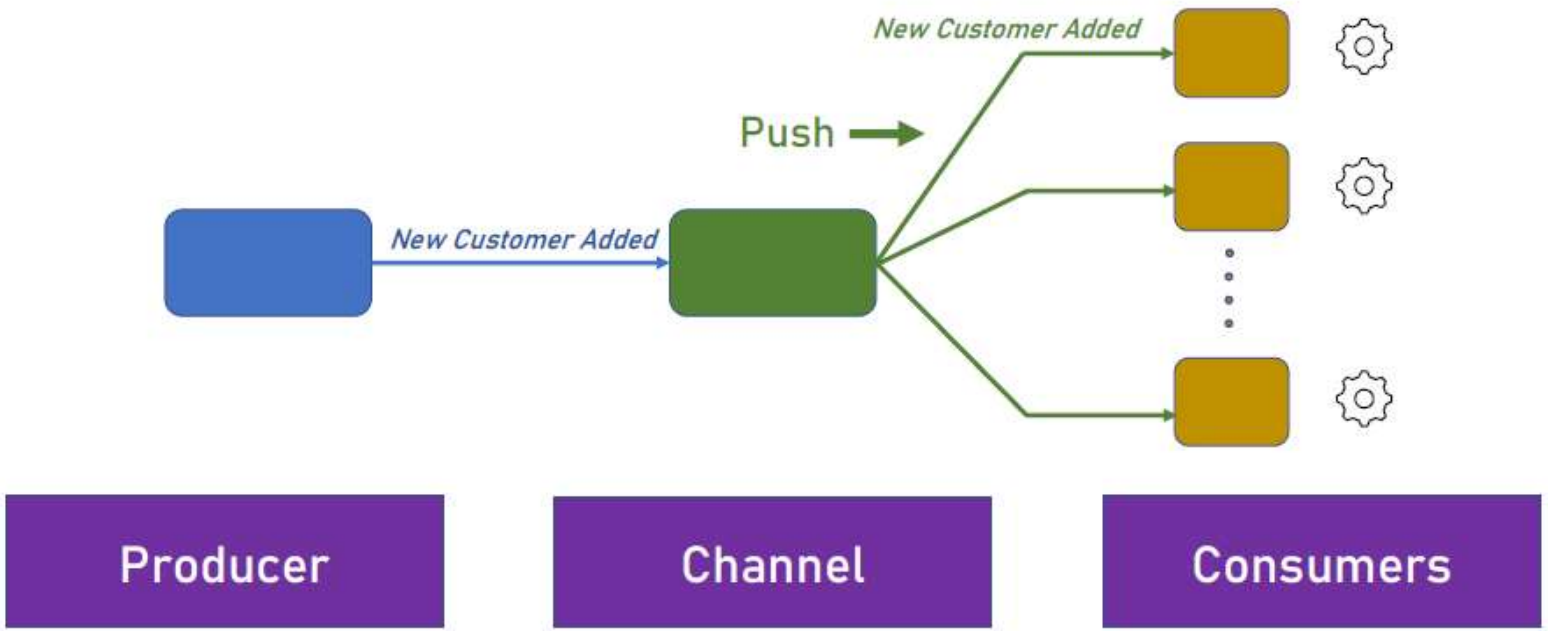
Способи отримання події Consumer’ом

- Consumer отримує подію за допомогою:
- Push
- Pull
- Спосіб залежить від каналу



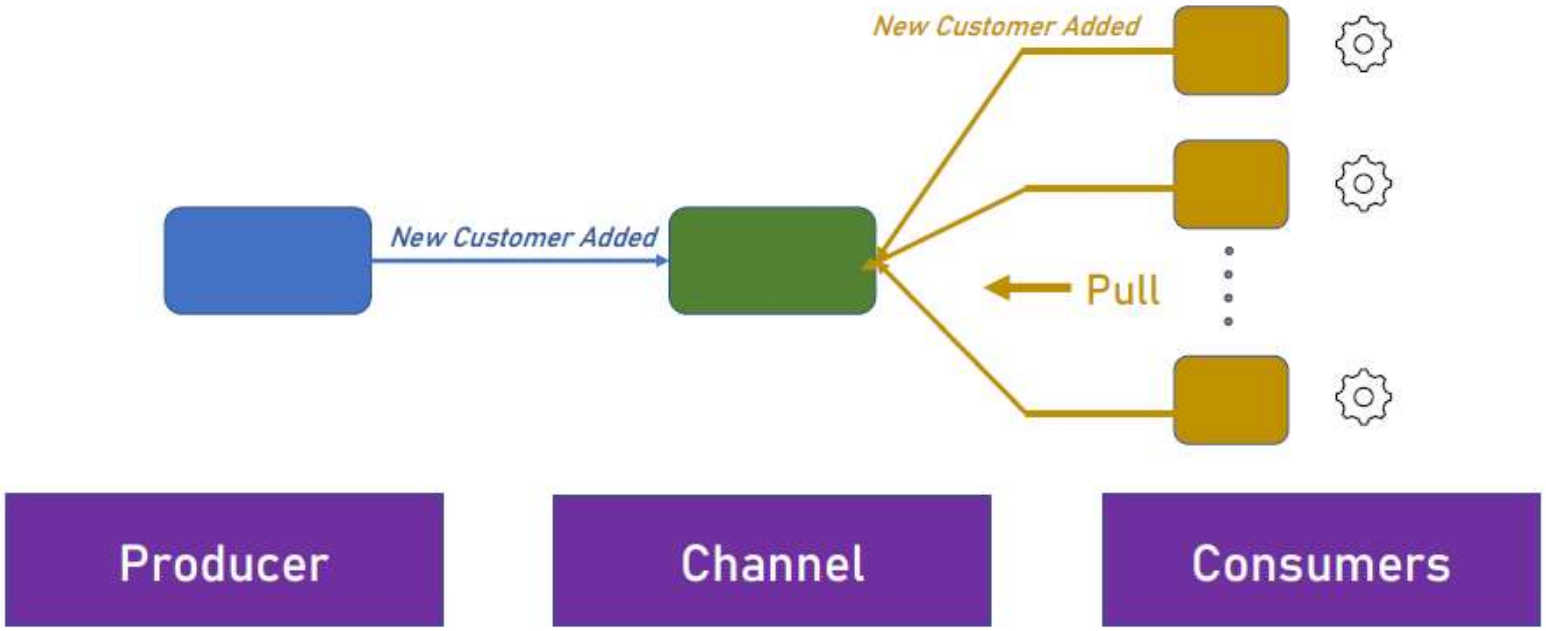
Push метод

- Consumer отримує подію за допомогою push - канал надсилає подію Consumer'ам



Pull метод

- Consumer отримує подію за допомогою pull - час від часу Consumer'и завертаються до каналу і питають чи є нові події для обробки



Переваги EDA

- Вирішені проблеми мікросервісів з
  - Продуктивністю
    - EDA — це асинхронна архітектура
    - Канал не чекає відповіді від споживача
    - Відсутність вузьких місць продуктивності
  - Зчепленням
    - Producer надсилає події на канал,
    - канал розподіляє події по темах (Topic) / чергах (Queue),
    - обидва не знають, хто слухає подію(за винятком WebHooks)
    - Немає зчеплення
  - Масштабованістю
  - Багато споживачів можуть слухати події з каналу, за потреби можна додати більше, Канал та Producer не зачіпаються при масштабуванні, Повна масштабованість