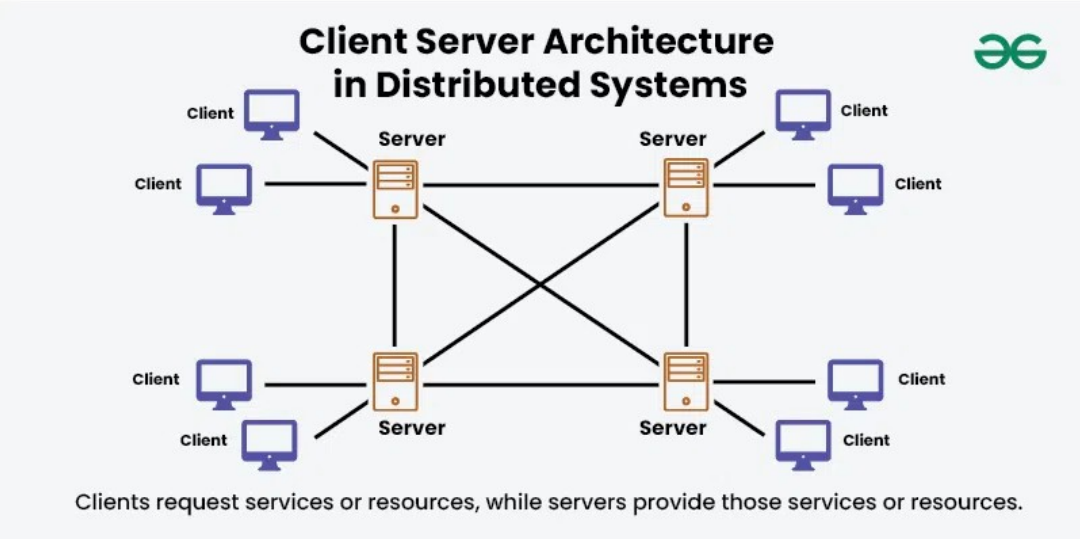
Тема 11. Архітектура розподілених систем. REST архітектура

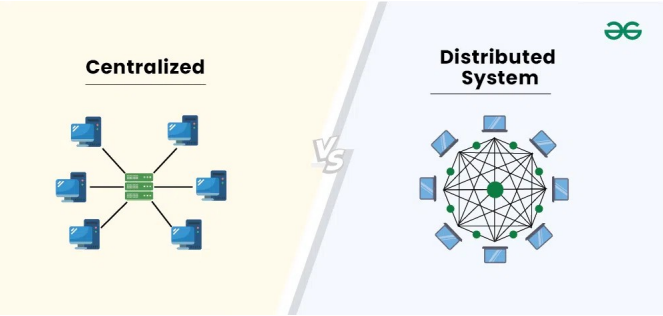
Архітектура розподілених систем

- Розподілені інформаційні системи є основою сучасних цифрових екосистем.
- На відміну від централізованих рішень, де всі процеси виконуються на одному сервері, розподілені системи передбачають виконання обчислень та зберігання даних на кількох вузлах, з’єднаних мережею.
- Цей підхід забезпечує масштабованість, стійкість до збоїв і ефективне використання ресурсів. Розподілені архітектури лежать в основі хмарних сервісів, мікросервісних систем, peer-to-peer мереж, IoT і глобальних платформ, таких як Google, Netflix чи Amazon.



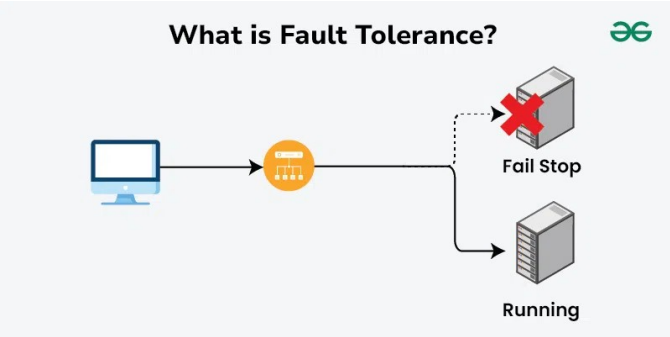
Централізована і розподілена системи

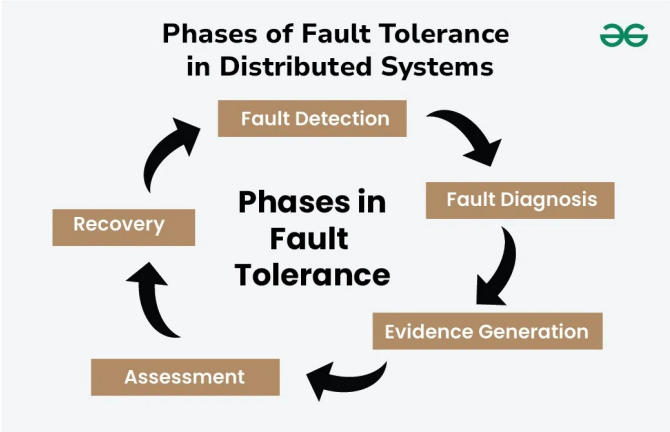
Ознака	Централізована система	Розподілена система
Архітектура	Один головний сервер керує всіма операціями	Кілька вузлів взаємодіють через мережу
Зберігання даних	У єдиній базі даних	Репліковане або розподілене між вузлами
Масштабування	Обмежене апаратними ресурсами сервера	Горизонтальне — додавання нових вузлів
Надійність	Відмова сервера = зупинка системи	Відмова окремого вузла не впливає на роботу всієї системи
Продуктивність	Висока для невеликих систем	Висока при великій кількості користувачів
Приклади	Локальні бази даних, ERP на одному сервері	Хмарні платформи, мікросервісні архітектури



Основні характеристики розподілених систем

- Прозорість розподілу — користувач не помічає, що ресурси розміщені на різних вузлах.
- Надійність — система продовжує працювати навіть при відмові окремих компонентів.
- Масштабованість — можливість додавати нові вузли без зміни логіки роботи.
- Асинхронність і паралельність — одночасне виконання запитів.
- Відкритість — стандартизовані протоколи обміну (HTTP, gRPC, MQTT).





CAP-теорема (Brewer’s Theorem)

CAP-теорема визначає три ключові властивості, які не можуть одночасно бути гарантовані у розподіленій системі:

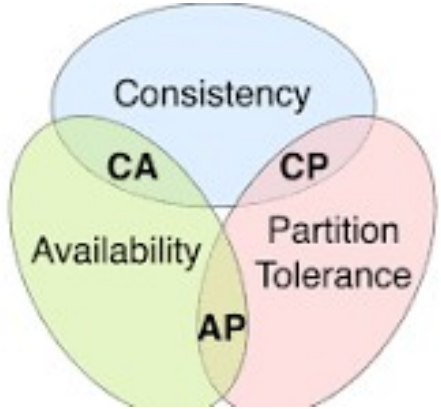
- **C — Consistency (узгодженість)**: усі вузли бачать однакові дані в один момент часу.
- **A — Availability (доступність):** кожен запит отримує відповідь, навіть якщо частина системи недоступна.
- **P — Partition tolerance (стійкість до розділення мережі**): система продовжує працювати, навіть коли частина вузлів втратила зв’язок між собою.

У розподіленій системі неможливо одночасно гарантувати всі три властивості.

При настанні мережевого розділення (P) система повинна зробити вибір між узгодженістю (C) та доступністю (A).

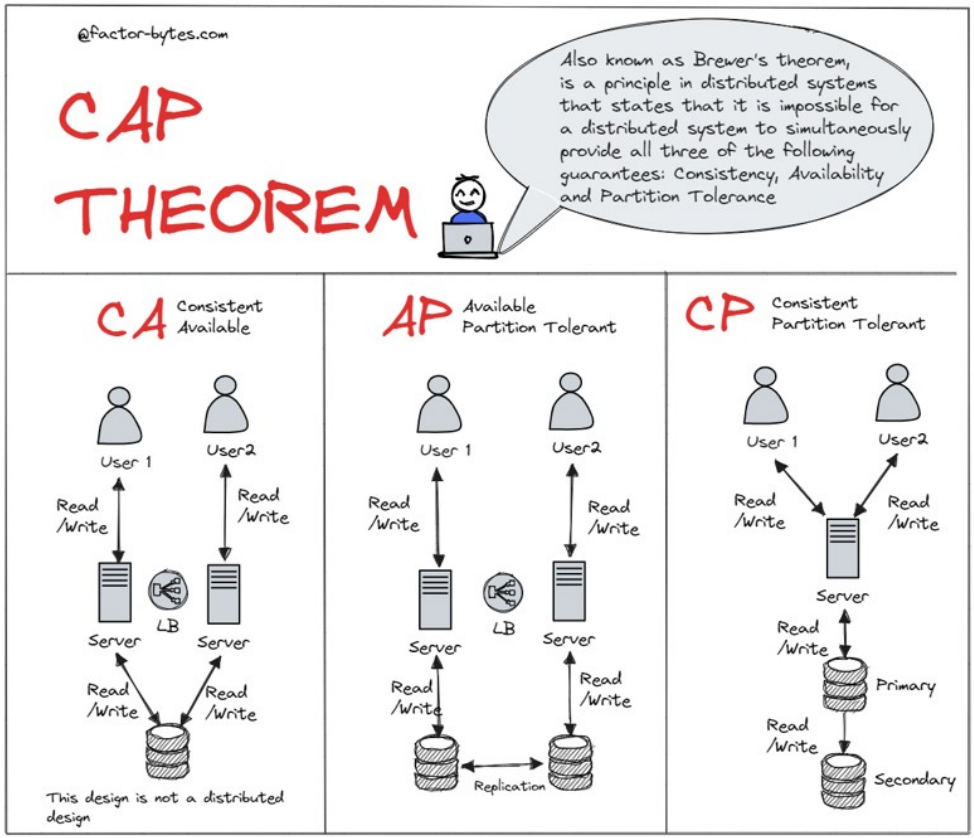
CAP-теорема, сформульована Еріком Брювером (Eric Brewer) у 2000 році, пояснює компроміс, з яким стикається кожна розподілена система.
В умовах відмови мережі неможливо одночасно зберегти повну узгодженість даних і абсолютну доступність сервісів.
Тому архітектори систем свідомо обирають пріоритет залежно від бізнес-вимог:

- фінансові системи тяжіють до CP,
- соціальні платформи — до AP,
- а внутрішні корпоративні рішення можуть бути CA, якщо працюють у стабільному середовищі.



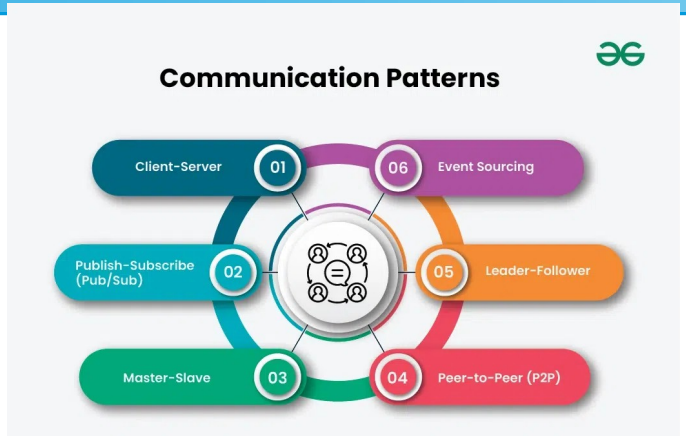
CAP-теорема (Brewer’s Theorem)

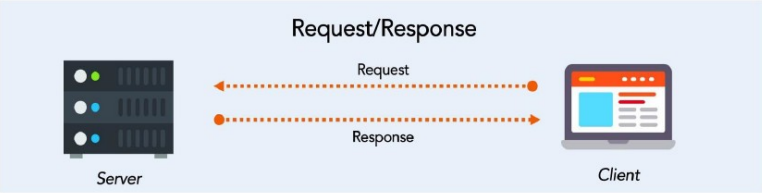
Тип системи	Пріоритет	Приклад
CA (Consistency + Availability)	Немає розділення мережі. Узгодженість і швидкість у стабільній мереж	ERP-системи в локальній інфраструктурі, монолітні веб-додатки
CP (Consistency + Partition tolerance)	Важлива точність даних. Узгодженість важливіша за доступність	Банківські системи, ZooKeeper, Etcd, Google Spanner
AP (Availability + Partition tolerance)	Важлива безперервність роботи. Краще повернути застарілі або часткові дані, ніж не відповісти	DNS, CDN-мережі, Amazon Dynamo, Cassandra, Netflix Eureka

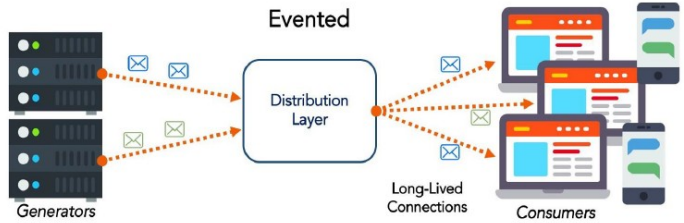


Типи архітектур розподілених систем

- Client-Server — класична модель, де клієнт звертається до сервера за послугами.
- Peer-to-Peer (P2P) — усі вузли рівноправні, напр. BitTorrent або блокчейн.
- Service-Oriented Architecture (SOA) — сервіси взаємодіють через стандартизовані інтерфейси.
- Event-Driven Architecture (EDA) — компоненти обмінюються повідомленнями через події.
- Microservices — подальший розвиток SOA, із незалежним масштабуванням кожного компонента.







Принципи побудови розподілених систем

- Декомпозиція — поділ системи на окремі сервіси або вузли.
- Комунікація — використання протоколів обміну (HTTP, TCP/IP, RPC).
- Синхронізація — координація станів між сервісами.
- Реплікація — копіювання даних для підвищення доступності.
- Толерантність до збоїв — використання кластерів і балансувальників навантаження.

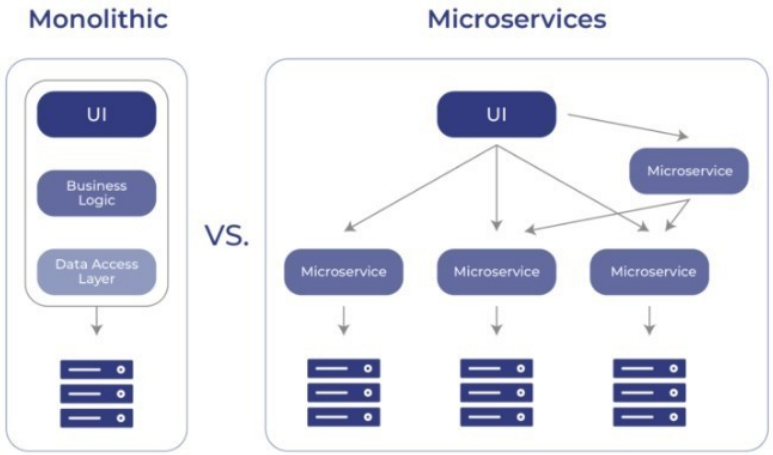
Декомпозиція

Декомпозиція — це принцип поділу великої системи на менші, незалежні компоненти (сервіси або вузли), кожен із яких виконує чітко визначену функцію.

Кожен компонент повинен бути самостійним, мати власний життєвий цикл і можливість розгортання без впливу на інші частини системи. Це дає змогу зменшити складність, підвищити гнучкість і масштабованість.

Приклади:

- поділ монолітного веб-додатка на сервіси “користувачі”, “замовлення”, “платежі”;
- мікросервісна архітектура Amazon або Netflix.



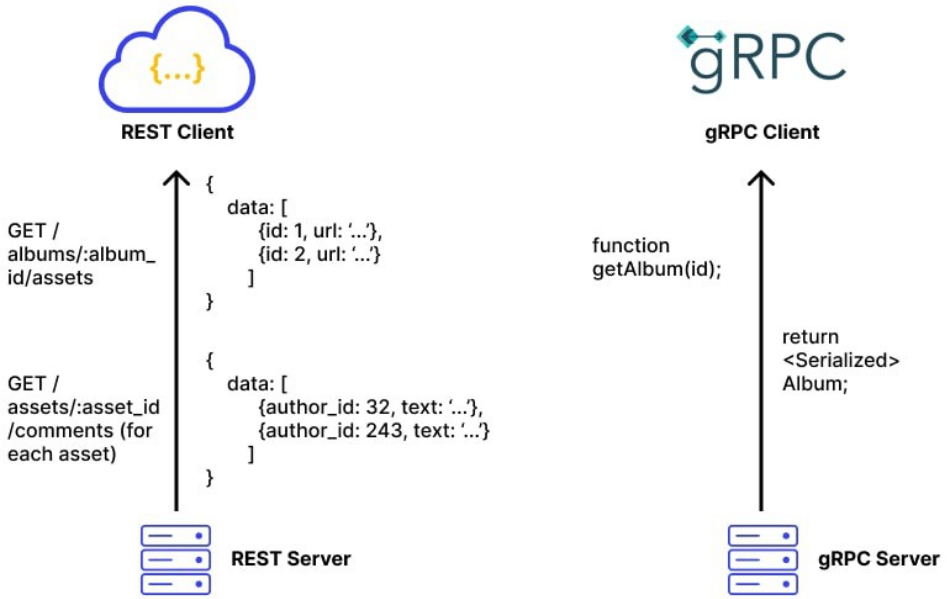
Комунікація

Комунікація у розподілених системах — це процес обміну даними між вузлами або сервісами через мережеві протоколи.

Оскільки компоненти фізично розташовані на різних машинах, вони спілкуються через протоколи зв’язку, найчастіше — HTTP, TCP/IP, WebSocket, RPC або gRPC. Комунікація може бути синхронною (очікування відповіді) або асинхронною (черги повідомлень, брокери подій).

Приклади:

- REST API-запити між фронтендом і бекендом;
- передача повідомлень через RabbitMQ або Kafka;
- взаємодія мікросервісів через gRPC.



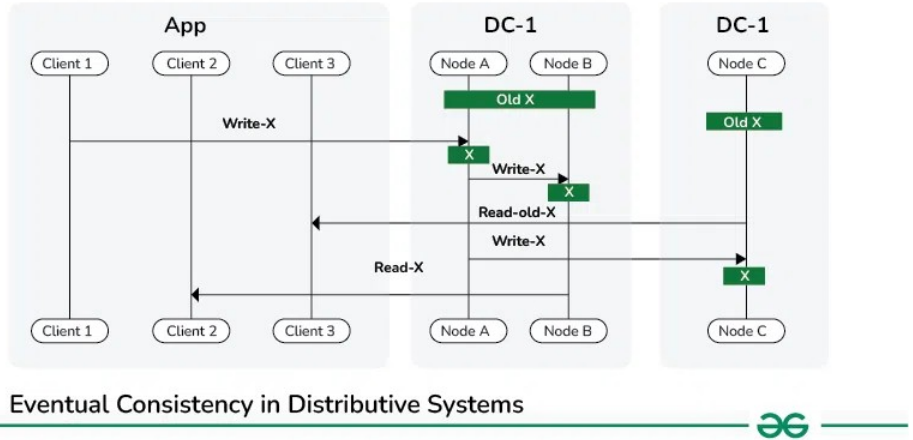
Синхронізація

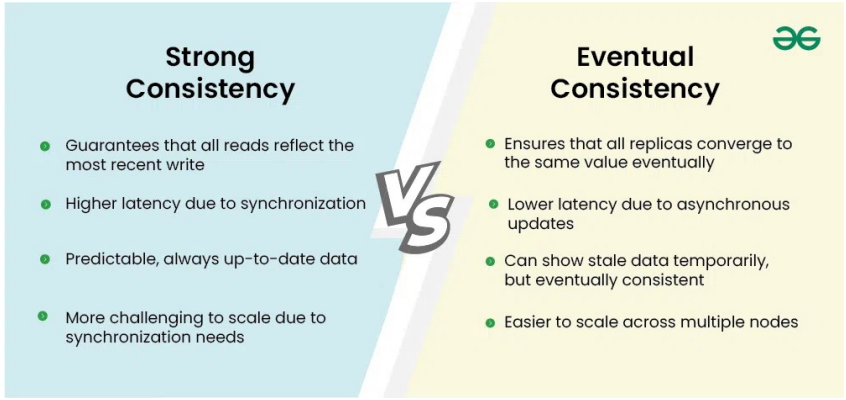
Синхронізація — це процес узгодження станів між сервісами або вузлами розподіленої системи.

Оскільки дані можуть оновлюватися одночасно у кількох вузлах, потрібні механізми, які забезпечують узгодженість (consistency).

Синхронізація може бути:

- жорсткою (strict) — усі вузли мають однакові дані у будь-який момент (наприклад, Paxos, Raft);
- узгодженість в кінцевому підсумку (eventual consistency) — вузли стають узгодженими через певний час (як у DynamoDB або Cassandra).





Реплікація

Реплікація — це створення і підтримання копій даних на кількох вузлах системи для підвищення доступності та надійності.

У разі відмови одного вузла інші можуть обслуговувати запити без втрати даних. Реплікація буває:

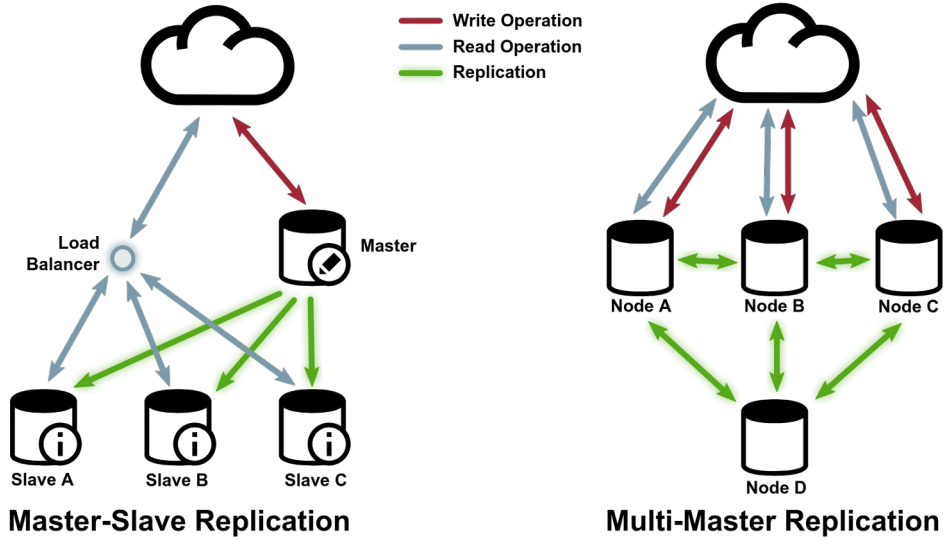
синхронна — зміни застосовуються одночасно у всіх копіях;

асинхронна — основна копія оновлюється, а решта наздоганяють з певною затримкою.

Приклади:

- реплікація бази даних у MySQL

  Master-Slave або Master-Master режимі;
- клонування сховищ у Cassandra чи MongoDB;
- реплікація об’єктів у S3 та CDN.



Толерантність до збоїв (Fault Tolerance)

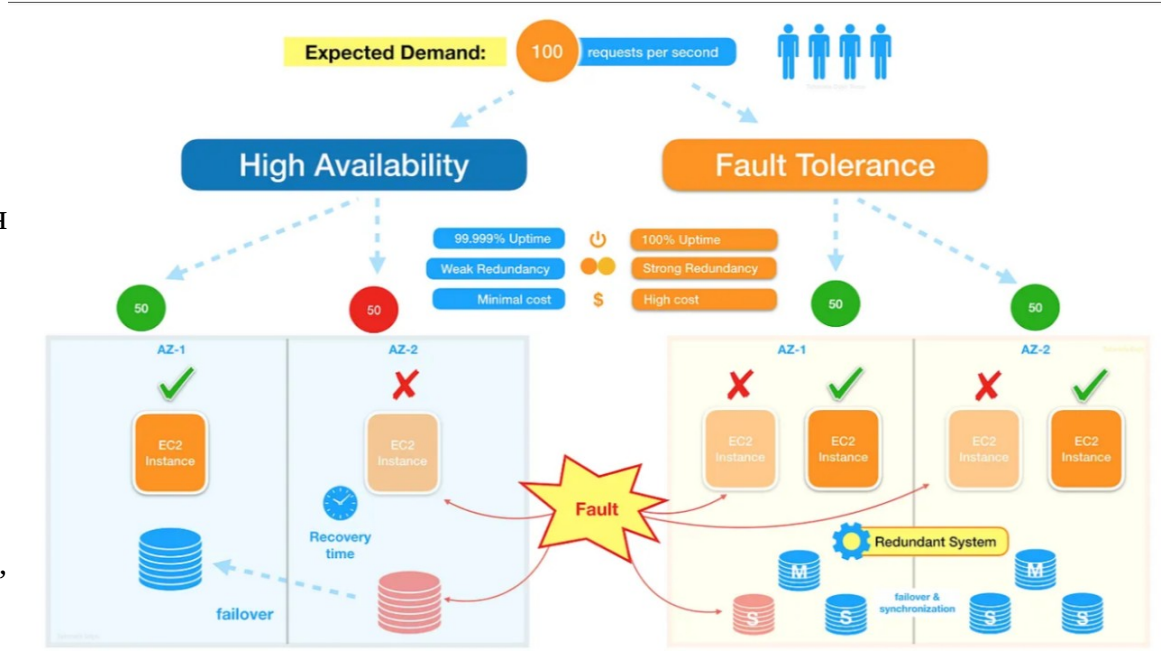
Толерантність до збоїв — це здатність системи продовжувати роботу навіть при відмові окремих компонентів.

Розподілені системи проектуються так, щоб збої були очікуваними, а не винятковими ситуаціями. Для цього застосовують:

- **кластеризацію** — дублювання вузлів;
- **балансування навантаження** — перенаправлення запитів на здорові вузли;
- **автоматичне відновлення (self-healing);**
- **моніторинг і heartbeat-перевірки.**

Приклади:

- Kubernetes автоматично перезапускає контейнер після збою;
- Netflix Chaos Monkey тестує стійкість, навмисно “ламаючи” вузли;
- Load balancer у AWS або Nginx забезпечує безперервність доступу.

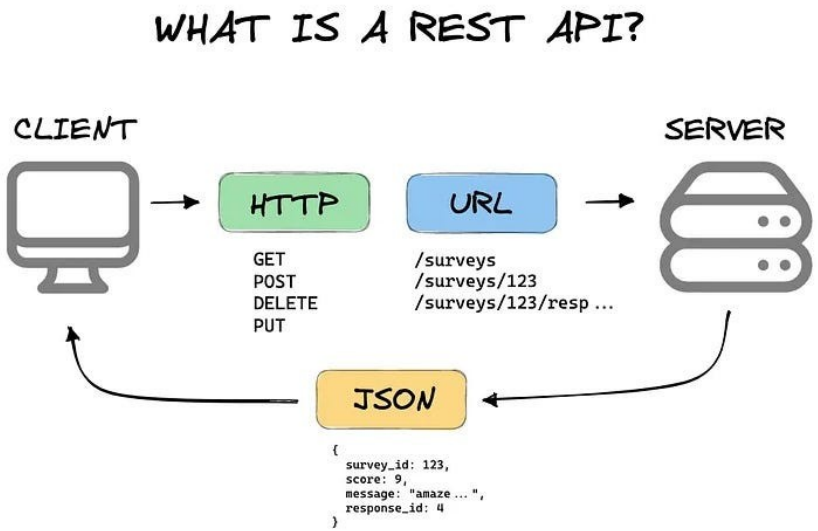


REST

REST (Representational State Transfer) — це архітектурний стиль, який визначає принципи взаємодії клієнта й сервера через стандартний протокол HTTP.

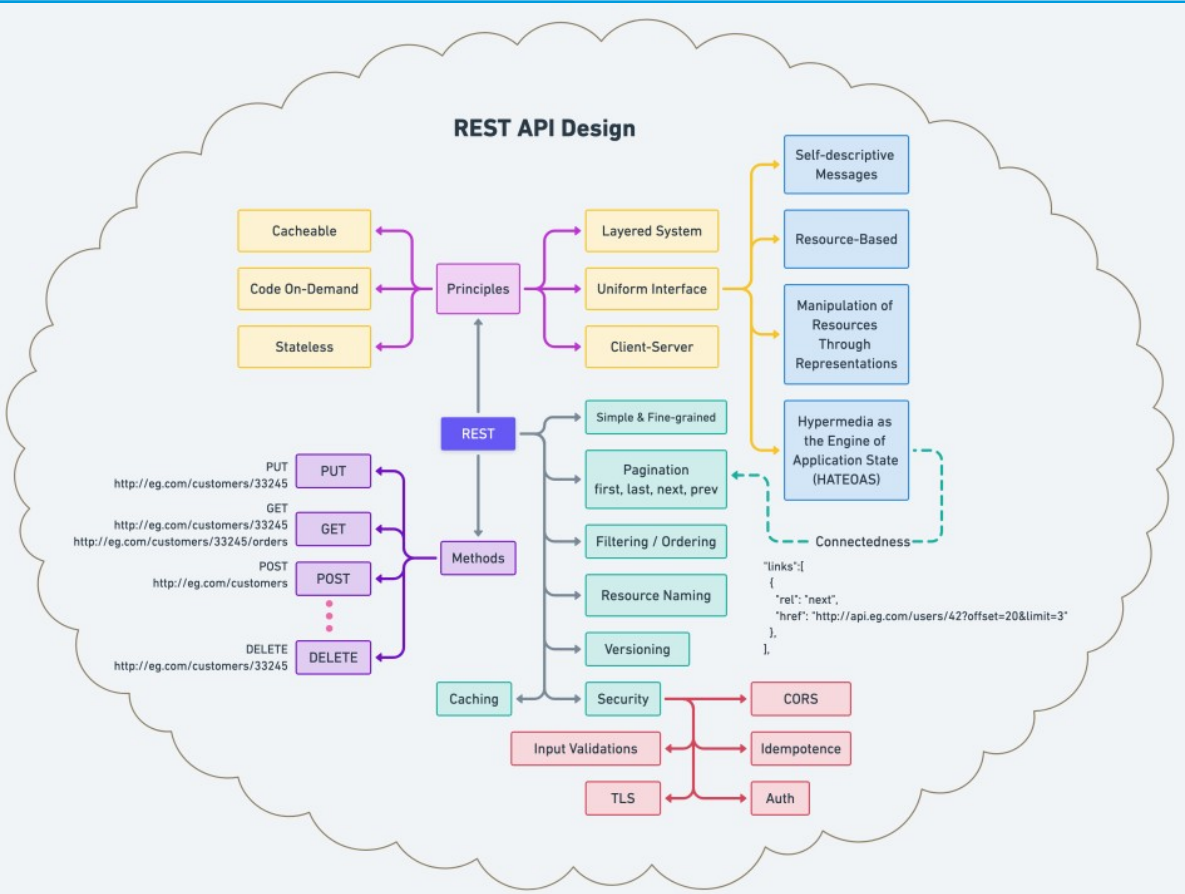
REST не є технологією чи бібліотекою — це набір принципів, які забезпечують простоту, масштабованість і незалежність компонентів.

REST був описаний Роєм Філдінгом у 2000 році як альтернатива складним XML-вебсервісам (SOAP) у межах архітектури Інтернету.



Принципи REST-архітектури

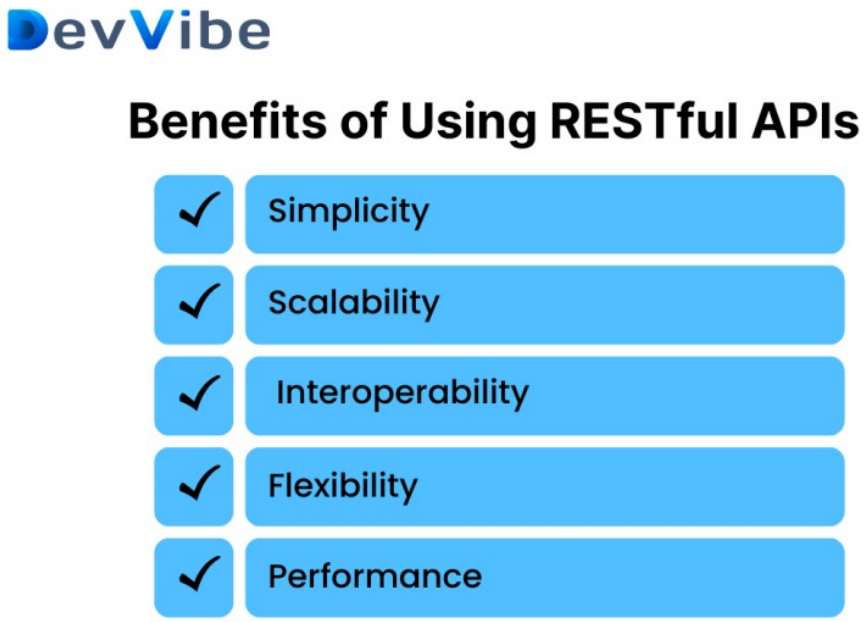
- **Клієнт-серверна модель** — розділення відповідальностей: клієнт відповідає за інтерфейс, сервер — за дані.
- **Відсутність стану (stateless)** — кожен запит самодостатній, не залежить від попередніх.
- **Кешування (cacheable)** — можливість зберігати відповіді для зменшення навантаження.
- **Єдиний інтерфейс (uniform interface)** — використання уніфікованих методів HTTP (GET, POST, PUT, DELETE).
- **Багаторівнева система (layered)** — клієнт не знає про проміжні проксі чи балансувальники.
- **Код за запитом (optional)** — можливість передавати виконуваний код (наприклад, JavaScript).



Переваги REST

- Простота реалізації та розуміння
- Підтримка стандартних протоколів (HTTP, HTTPS).
- Незалежність клієнта від сервера.
- Легка інтеграція з веб та мобільними додатками.
- Масштабованість і кешування відповідей.

REST став стандартом для веб-API у сучасних платформах – від соціальних мереж до IoT-пристроїв.



Обмеження REST

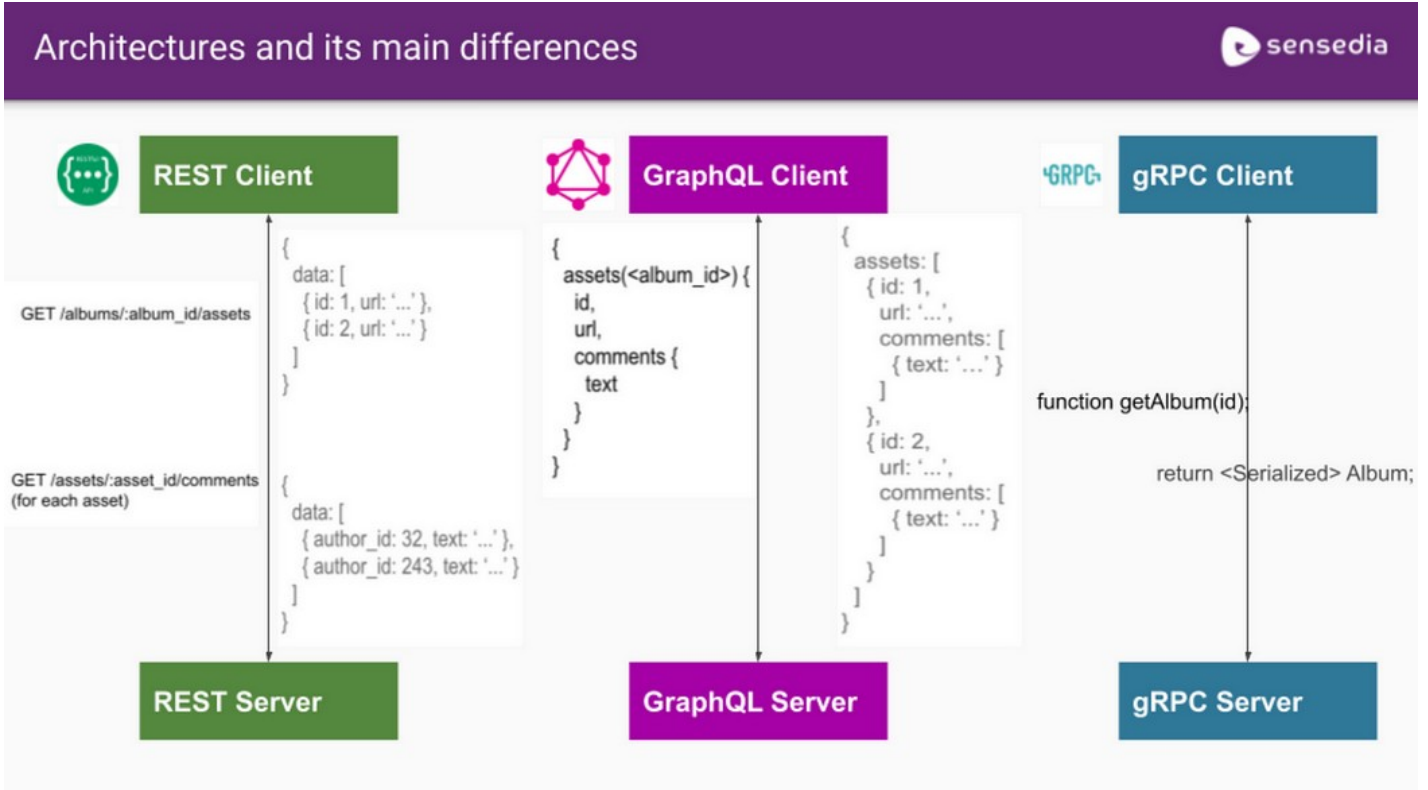
- Відсутність підтримки складних транзакцій.
- Неоптимальність при великих об’ємах даних
- Високі накладні витрати HTTP-запитів.
- Немає вбудованого механізму push-повідомлень (для цього використовують WebSocket або gRPC).

Еволюція REST: GraphQL i gRPC

Нові підходи намагаються подолати обмеження REST:

- GraphQL (Facebook, 2015) — дозволяє клієнту самостійно визначати, які дані потрібні.
- gRPC (Google, 2016) — високопродуктивна модель RPC на основі HTTP/2 і protobuf, орієнтована на взаємодію мікросервісів.

Однак REST залишається базовим архітектурним підходом для веб- і мобільних систем через універсальність та легкість інтеграції.



Висновки

REST-архітектура — це ключова концепція побудови розподілених систем у цифрову епоху.

Її простота, гнучкість і відповідність принципам Інтернету зробили REST універсальним стандартом комунікації між клієнтами й сервісами.

У поєднанні з мікросервісами й контейнеризацією REST формує основу сучасних масштабованих архітектур.

Дякую за увагу!