Sieci przewodowe

Sieć przewodowa to **rodzaj sieci komputerowej**, w której **urządzenia są połączone za pomocą fizycznych kabli** — najczęściej **miedzianych (Ethernet)** lub **światłowodowych (fiber optic)**.

🧩 **Najprościej mówiąc:**

> Sieć przewodowa = połączenie komputerów, drukarek, routerów itp. **za pomocą kabli**, a nie przez Wi-Fi.

Główne cechy:

- **Przesył danych**: Dane są transmitowane przez przewodniki, co zapewnia stabilne i szybkie połączenie, zazwyczaj z mniejszą podatnością na zakłócenia niż w sieciach bezprzewodowych.

- **Przykłady zastosowań**: Sieci LAN w domach, biurach, centrach danych czy systemy monitoringu CCTV.

Przykłady sieci przewodowych

Ethernet (LAN) — najczęściej spotykana sieć przewodowa w domach i firmach.

- używa kabli **RJ-45 (skrętka)**,
- prędkości np. **100 Mb/s, 1 Gb/s, 10 Gb/s**,
- łączy komputery, routery, switche.

Światłowód (Fiber Optic) — sieć zbudowana z kabli światłowodowych.

- bardzo duża prędkość transmisji,
- stosowana w łączach między miastami, serwerowniach lub w nowoczesnych domach (FTTH – Fiber To The Home).

Połączenia szeregowe / przemysłowe — np. RS-232, RS-485 — używane w automatyce i systemach przemysłowych.

Jak działa sieć przewodowa

1. Każde urządzenie ma **kartę sieciową (NIC)**.
2. Kable łączą urządzenia poprzez **switch**, **router** lub **hub**.
3. Dane przesyłane są w postaci **sygnałów elektrycznych lub optycznych** przez przewody.
4. Komunikacja odbywa się według określonych **protokołów sieciowych** (np. TCP/IP, Ethernet).

Zalety sieci przewodowej

✅ **Szybka transmisja danych** – stabilniejsze połączenie niż Wi-Fi.

✅ **Niskie opóźnienia (ping)** – ważne np. w grach i serwerach.

✅ **Bezpieczeństwo** – trudniej podsłuchać połączenie fizyczne.

✅ **Odporność na zakłócenia radiowe** – nie zależy od zasięgu Wi-Fi.

Wady sieci przewodowej

❌ **Trudniejszy montaż** – trzeba prowadzić kable w ścianach lub po podłodze.

❌ **Mniejsza mobilność** – urządzenia muszą być fizycznie podłączone.

❌ **Koszty instalacji** – więcej sprzętu (kable, gniazda, przełączniki).

Sieć bezprzewodowa

Sieć komputerowa, w której urządzenia **łączą się ze sobą bez użycia kabli**, wykorzystując **fale radiowe**, **podczerwień** lub **mikrofale** do przesyłania danych.

🧩 **Najprościej mówiąc:**

> **Sieć bezprzewodowa = połączenie komputerów, telefonów, tabletów i innych urządzeń bez kabli**, np. przez **Wi-Fi** lub **Bluetooth**.

Przykłady sieci bezprzewodowych

- **Wi-Fi (WLAN – Wireless Local Area Network)**
  - najpopularniejsza sieć domowa lub biurowa,
  - wykorzystuje fale radiowe (częstotliwości 2,4 GHz i 5 GHz),
  - umożliwia łączenie komputerów, smartfonów i innych urządzeń z routerem bez kabli.
- **Bluetooth**
  - służy do połączenia urządzeń na krótką odległość (np. telefon ↔ słuchawki).

- **Sieć komórkowa (LTE, 5G)**
  - zapewnia dostęp do Internetu przez operatorów sieci komórkowych,
  - obejmuje duże obszary (miasta, regiony).

- **Hotspot Wi-Fi**
  - punkt dostępu umożliwiający innym urządzeniom korzystanie z Internetu bezprzewodowo.

Jak działa sieć bezprzewodowa

1. Urządzenie (np. laptop, telefon) ma **kartę sieciową Wi-Fi**.
2. Wysyła i odbiera dane przez **antenę**, która komunikuje się z **routerem (punktem dostępowym)**.
3. Router przekazuje dane dalej – np. do Internetu przez kabel (światłowód lub Ethernet).
4. Komunikacja odbywa się wg ustalonych **protokołów sieciowych** (np. 802.11).

Zalety sieci bezprzewodowej

• Mobilność	Można się łączyć z dowolnego miejsca w zasięgu sygnału
• Brak kabli	Nie trzeba prowadzić przewodów przez ściany
• Łatwość instalacji	Wystarczy router i urządzenia z Wi-Fi
• Wiele urządzeń	Może łączyć laptopy, smartfony, drukarki itd.

Wady sieci bezprzewodowej

- Mniejsza stabilność — Sygnał może zanikać przez ściany lub zakłócenia
- Bezpieczeństwo — Łatwiej podsłuchać lub włamać się niż w sieci przewodowej
- Niższa prędkość — Wolniejsza transmisja niż przez kabel Ethernet
- Zużycie energii — Urządzenia mobilne szybciej rozładowują baterię

Polityka bezpieczeństwa w sieci

Polityka bezpieczeństwa sieci (Network Security Policy) to **zbiór zasad, procedur i wytycznych**, które określają:

- jak chronić dane i zasoby sieciowe,
- kto i w jaki sposób może korzystać z sieci,
- jakie działania są dozwolone, a jakie zabronione,
- jak reagować w razie incydentów bezpieczeństwa.

Innymi słowy: to **plan działania**, który pomaga organizacji **utrzymać poufność, integralność i dostępność danych (zasada CIA: Confidentiality, Integrity, Availability)**.

Główne cele polityki bezpieczeństwa

1. **Poufność (Confidentiality)** – ochrona danych przed nieautoryzowanym dostępem.
   👉 np. szyfrowanie, uwierzytelnianie użytkowników.

2. **Integralność (Integrity)** – zapobieganie nieautoryzowanym zmianom danych.
   👉 np. sumy kontrolne, kopie zapasowe.

3. **Dostępność (Availability)** – zapewnienie, że dane i usługi są dostępne, gdy są potrzebne.
   👉 np. ochrona przed atakami DDoS, redundancja serwerów.

Rodzaje polityk bezpieczeństwa w sieci

1. **Polityka dostępu do sieci (Access Control Policy)**

   Określa, kto może korzystać z sieci i w jakim zakresie.
   - *Przykład:* pracownicy działu finansowego mają dostęp do serwera księgowego, ale nie do serwera HR.

2. **Polityka haseł (Password Policy)**

   Ustala reguły dotyczące złożoności, długości i zmiany haseł.
   - *Przykład:* hasło musi mieć min. 10 znaków, zawierać litery, cyfry i znaki specjalne, zmiana co 90 dni.

3. **Polityka korzystania z Internetu (Acceptable Use Policy)**

   Określa, w jaki sposób można korzystać z Internetu w pracy.
   - *Przykład:* zakaz wchodzenia na strony o treści niezgodnej z prawem lub niezwiązanej z pracą.

Rodzaje polityk bezpieczeństwa w sieci c.d.

4. **Polityka zapory sieciowej (Firewall Policy)**
   Definiuje, które połączenia są dozwolone, a które blokowane.
   - *Przykład:* ruch przychodzący na port 22 (SSH) dozwolony tylko z sieci firmowej.

5. **Polityka aktualizacji i łatania systemów (Patch Management Policy)**
   Określa zasady instalowania poprawek bezpieczeństwa.
   - *Przykład:* wszystkie serwery muszą być aktualizowane co najmniej raz w miesiącu.

6. **Polityka tworzenia kopii zapasowych (Backup Policy)**
   Wskazuje, jak i kiedy wykonywać backupy danych.
   - *Przykład:* codzienna kopia danych na serwer zewnętrzny, przechowywana przez 30 dni.

7. **Polityka reagowania na incydenty (Incident Response Policy)**
   Opisuje procedury w razie włamania, ataku lub awarii.
   - *Przykład:* po wykryciu ataku DDoS administrator odłącza serwer i powiadamia zespół SOC.

Praktyczne zastosowania

Pracownik loguje się do sieci VPN z domu
Zastosowana polityka: Polityka dostępu + polityka haseł (Efekt: Bezpieczny, uwierzytelniony dostęp)

Firma blokuje porty 21 (FTP) i 23 (Telnet)
Zastosowana polityka: Polityka zapory sieciowej (Efekt: Ochrona przed niebezpiecznymi protokołami)

Użytkownicy mają ograniczony dostęp do pendrive'ów
Zastosowana polityka: Polityka dostępu do urządzeń (Efekt: Zapobieganie wyciekom danych)

Codzienne kopie zapasowe serwerów
Zastosowana polityka: Polityka backupu (Efekt: Ochrona przed utratą danych)

Pracownicy nie mogą instalować oprogramowania
Zastosowana polityka: Polityka uprawnień użytkownika (Efekt: Redukcja ryzyka złośliwego oprogramowania)

Polityka haseł

Polityka haseł określa, jak **trudne powinny być hasła** i jak często trzeba je **zmieniać**.

Ma to chronić konta użytkowników przed włamaniami.

Hasło **powinno mieć duże i małe litery, cyfry i znaki specjalne**.

Nie powinno się używać tego samego hasła w kilku miejscach.

W firmach często **wymaga się zmiany hasła co kilka miesięcy**.

To **pomaga zwiększyć bezpieczeństwo** danych i systemów.

Polityka korzystania z Internetu

Ta polityka mówi, jak **pracownicy mogą używać Internetu w pracy.**

Na przykład **zabrania wchodzenia na strony niezwiązane z obowiązkami służbowymi.**

Chroni to firmę przed wirusami i stratą czasu w pracy.

Dzięki niej **administratorzy mogą łatwiej monitorować ruch sieciowy.**

W szkole taka **zasada też może obowiązywać** – np. nie wolno grać online podczas lekcji.

To **pomaga utrzymać porządek i bezpieczeństwo w sieci.**

Polityka zapory sieciowej (Firewall)

Zapora sieciowa **to program lub urządzenie, które blokuje niebezpieczne połączenia z Internetu.**

Polityka określa, które **połączenia są dozwolone, a które zablokowane**.

Na przykład może **pozwalać tylko na strony firmowe, a blokować porty używane przez hakerów**.

Dzięki temu **wirusy i nieautoryzowani użytkownicy nie dostają się do sieci.**

Firewall działa jak **strażnik** pilnujący bramy do firmy.

To jedna z podstawowych metod ochrony sieci komputerowych.

Polityka tworzenia kopii zapasowych (Backup)

Ta polityka mówi, **jak często trzeba robić kopie danych**, żeby ich nie stracić.

Backupy mogą być zapisywane na dysku zewnętrznym lub w **chmurze**.

W razie awarii, **ataku lub błędu można łatwo odzyskać dane**.

Firmy często robią kopie codziennie lub co tydzień.

W domu też warto robić kopie zdjęć czy **dokumentów**.

Dzięki temu nawet po awarii komputera dane nie przepadają.

Polityka reagowania na incydenty

Ta polityka określa, co robić, gdy **dojdzie do ataku lub problemu z bezpieczeństwem**.

Na przykład: kto ma zostać powiadomiony, jak zabezpieczyć dane i jak naprawić system.

Pomaga **szybko zareagować, zanim szkody będą duże.**

W firmach często **tworzy się specjalne zespoły reagowania (tzw. CERT lub SOC)**.

Dzięki tej **polityce każdy wie, jakie są jego obowiązki.**

To pozwala **utrzymać spokój i porządek w sytuacjach kryzysowych**.

Programowanie rozproszone

to sposób tworzenia oprogramowania, w którym **program (system)** nie działa na jednym komputerze, tylko na **wielu połączonych ze sobą komputerach (w sieci)**.

Każda z tych maszyn wykonuje część zadań, a razem tworzą jeden wspólny system.

Rodzaje (modele) progr. rozproszonego c.d.

Rodzaj:
Klient–serwer (Client–Server) - Klient wysyła żądania, serwer odpowiada.
Przykład: Strona WWW (przeglądarka ↔ serwer)

Wielowarstwowe (n-warstwowe) - System podzielony na warstwy: prezentacji, logiki i danych.
Przykład: Aplikacja webowa: frontend – backend – baza danych

Peer-to-Peer (P2P) - Każdy komputer (węzeł) może być klientem i serwerem jednocześnie.
Przykład: Torrent, komunikatory (np. Skype, BitTorrent)

Rodzaje (modele) programow. rozproszonego

Zdalne wywołania procedur (RPC) - Program na jednym komputerze wywołuje funkcję na innym.
Przykład: Java RMI, gRPC

Systemy oparte o komunikaty (Message-Oriented) - Komputery komunikują się przez kolejki wiadomości.
Przykład: RabbitMQ, Apache Kafka

Mikroserwisowe (Microservices) - System składa się z wielu małych usług, które komunikują się przez sieć.
Przykład: Aplikacje w chmurze (np. Netflix, Amazon)

Grid Computing / Cloud Computing - Wiele serwerów współdzieli zasoby obliczeniowe lub pamięć.
Przykład: AWS, Azure, Google Cloud

Technologie używane w progr. rozproszonym

Obszar:	Przykłady
Backend	Java (Spring Cloud), .NET, Node.js
Komunikacja	REST API, gRPC, WebSocket, message queues (RabbitMQ, Kafka)
Bazy danych	MongoDB, Cassandra, PostgreSQL Cluster
Architektura	mikroserwisy, chmura (AWS, Azure, GCP)

Cechy systemów rozproszonych

Cecha	Znaczenie
Współbieżność	wiele komputerów działa jednocześnie
Komunikacja przez sieć	wymiana danych np. przez HTTP, TCP/IP
Niezależność sprzętowa	różne komputery i systemy operacyjne mogą współpracować
Odporność na awarie	gdy jeden element padnie, reszta może działać dalej
Skalowalność	łatwo dodać więcej maszyn, żeby system działał szybciej

Przykład z życia

Wyobraź sobie aplikację bankową:

- Serwer A obsługuje logowanie,
- Serwer B zapisuje dane klientów w bazie,
- Serwer C przetwarza płatności.

Współpracują razem w sieci — **to właśnie programowanie rozproszone**.

GIT

Git to **system kontroli wersji** (ang. *Version Control System*), używany do **śledzenia zmian w plikach**, głównie w projektach programistycznych. Dzięki niemu możesz współpracować z innymi, wracać do wcześniejszych wersji plików i zarządzać historią projektu.

Najważniejsze cechy Gita

Śledzenie zmian

- Git zapisuje historię zmian plików.
- Możesz zobaczyć, kto i kiedy zmienił dany fragment kodu.

Gałęzie (*branches*)

- Pozwalają na tworzenie równoległych wersji projektu.
- Możesz testować nowe funkcje bez psucia głównej wersji (*main* lub *master*).

Scalanie zmian (*merge*)

- Gałęzie można łączyć.
- Git automatycznie łączy zmiany w jednym projekcie.

Najważniejsze cechy Gita

Rozproszony system

- Każdy użytkownik ma pełną kopię repozytorium.
- Możesz pracować lokalnie bez połączenia z internetem.

Współpraca

- Git ułatwia pracę zespołową.
- Popularne platformy: **GitHub**, **GitLab**, **Bitbucket**.

Podstawowe komendy Git

git init	- Tworzy nowe repozytorium w folderze
git clone <url>	- Pobiera repozytorium z internetu
git add <plik>	- Dodaje plik do "staging area" (przygotowanie do commit)
git commit -m "komentarz"	- Zapisuje zmiany z opisem
git status	- Pokazuje stan repozytorium i zmienione pliki

```
git
push
```
```
git
push
```

Podstawowe komendy Git

git push - Wysyła zmiany do zdalnego repozytorium

git pull - Pobiera zmiany z repozytorium zdalnego

git branch - Wyświetla dostępne gałęzie

git checkout <branch> - Przełącza się na inną gałąź

git commit

git commit - służy do zapisania zmian, które zostały wcześniej dodane do obszaru index (za pomocą **git add**)
Pomocne polecenia do wykonania commit

- **a** - przejście w tryb –INSERT , modyfikowania
- **ESCAPE :wq** - ESC (przełącza w tryb poleceń), w -zapisz, q-quit, wyjście

Ważne opcje:
-m "tytuł commit": Dodaje komunikat bezpośrednio w poleceniu.
--amend: Pozwala edytować ostatni commit (np. poprawić wiadomość lub dodać zapomniane pliki).
-a: Automatycznie dodaje wszystkie śledzone pliki do staging przed commitem (skrót od git add dla zmodyfikowanych plików).

git log

git log - służy do wyświetlania historii commitów w repozytorium. W kolejności chronologicznej – od najnowszego do najstarszego.
Wyświetlane informacje: Dla każdego commita pokazuje:

- ❏ Hash commita (unikalny identyfikator).
- ❏ Autora i datę.
- ❏ Wiadomość commita.

Ważne opcje:
--oneline: Wyświetla każdy commit w jednej linii (skrótowo: hash + wiadomość).
--graph: Pokazuje historię w formie grafu (drzewa branchy).
-n 5: Ogranicza do ostatnich n commitów (np. 5).
--author="Imię": Filtruje po autorze.
--since="2023-01-01": Commit'y od danej daty.
-p lub --patch: Pokazuje pełne zmiany (diff) w każdym commicie.

index i git checkout

Index w Git (nazywany też **staging area** albo **strefą przejściową**) to **miejsce, w którym Git przechowuje informacje o zmianach przygotowanych do zatwierdzenia** (**commit**).

Przykład 1:
Tworzymy katalog w nim są plik **$ mkdir folder-test** i **$ touch folder-test/plik2.txt**. **Dodajemy je do index** za pomocą polecenie **$ git add folder-test**. Następnie usuwamy jeden z plików w katalogu folder-test **$ rm folder-test/plik2.txt.** Aby go odzyskać mimo tego, że nie ma go w katalogu roboczym należy wykonać polecenie **$ git checkout folder-test/plik2.txt**
Przykład 2:
Teraz jak dodamy tekst do pliku **folder-test/plik2.txt** to plik będzie wyświetlany 2 razy w index (gdzie będzie pusty) i w katalogu roboczym z nową zawartością tekstu.

index i git reset HEAD , touch .gitignore

git reset HEAD <plik> usuwa wskazany plik z **indexu**, ale **nie usuwa go z katalogu roboczego**.

Plik **.gitignore** służy do tego, aby powiedzieć Gitowi, **których plików i katalogów NIE ma śledzić** (czyli nie dodawać ich do indexu, commitów i repozytorium).
Git ignoruje pliki wymienione w **.gitignore, o ile nie były wcześniej dodane do repozytorium.**
Zwykle ignorujemy katalogi node_module, dist, .env oraz pliki *.log, *.temp, DS_Store

Plik **.gitignore** jest niewidoczny dopiero polecenie **$ ls –all** go wyświetli

git revert SHA-1

git revert SHA-1 tworzy NOWY commit, który wykonuje **odwrotność zmian z tamtego commita.**

Przykład 1:
W commit **git show SHA-1 dodaliśmy w 2 linii tekst, a usunęliśmy pierwszą linie tekstu, to możemy to odwrócić.** Czyli po wykonaniu polecenia **git revert SHA-1** odwrócimy to co było w pliku. **Druga linia zostanie usunięta, a pierwsza linia zostanie dodana z zawartością tekstu przed usunięciem jej**. Pamiętajmy, że ta **zmiana dodaje nowy commit**

Działanie git reset z historią commit

git reset - **resetuje index do danego commit** lub **zresetowanie indexu do wybranych plików danego commit**. Czyli **git reset HEAD nazwa.txt**, **pobierze z ostatniego commit plik nazwa.txt przywróci do index**. Natomiast **to co jest obecnie w index umieści w katalogu roboczym**

Sprawdzenie wykonuje się poleceniem **git restore nazwa-pliku**

git rebase -interactive HEAD~

git rebase - **służy do przeniesienia commit z jednego miejsca do drugiego.** W szczególności do poprawy commit, łączenia ich, układania w odpowiedniej kolejności, usuniecia zbędnych commit

Commands:
p, pick <commit> = use commit
r, reword <commit> = use commit, but edit the commit message
e, edit <commit> = use commit, but stop for amending
s, squash <commit> = use commit, but meld into previous commit
f, fixup <commit> = like "squash", but discard this commit's log message
x, exec <command> = run command (the rest of the line) using shell
b, break = stop here (continue rebase later with 'git rebase --continue')
d, drop <commit> = remove commit
l, label <label> = label current HEAD with a name
t, reset <label> = reset HEAD to a label
m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
. create a merge commit using the original merge commit's
. message (or the oneline, if no original merge commit was
. specified). Use -c <commit> to reword the commit message.

git branch -verbose, git branch nowy-brache

git branch -v - wyświetla listę wszystkich gałęzi razem z ostatnim commitem na każdej z nich (skrót commit hash + pierwsza linia wiadomości, gdzie * oznacza branche na którym jesteśmy)

git branch –list - zwraca listę branch (tylko nazwę)

git branch -m nowy-branch nowa-nazwa-branch - zmienia nazwę branch

git branch -d nazwa-branch - usuwa branch

git checkout nazwa-branch

git checkout nazwa-branch - przełączenie się na brancha (gałąź) o podanej nazwie

git checkout -b nazwa-branch - tworzy nowy branch i od razu przełącza się na niego

git stash

git stash - **tymczasowo odkłada Twoje zmiany na bok**, tak żeby repo wyglądało, jakby nie było żadnych zmian. Zmiany muszą znajdować się w index aby można było je odłożyć na bok za pomocą polecenia git stash

git stash list - wyświetla listę stash

git stash pop - zdejmuje z listy stash ostatnią zmianę oraz wyświetla szczegółowe informacje o statusie

git merge nazwa_branch

git merge nazwa_branch (lub nazwe commit, SHA-1) - łączy (scalą) **historię wskazanej gałęzi z aktualną gałęzią na której jesteś.**

Mogą się pojawić konflikty. Czyli jeśli te same linie w tych samych plikach zostały zmienione inaczej w obu branchach – powstaje konflikt.
Rozwiązanie konfliktów wykonuje się przez otworzenie pliku w którym są konflikty. Usunięciu linii **<<<<<<<, =======, >>>>>>>** oraz **wybraniem poprawnej wersji.** Po czym należy dodać pliki **$ git add nazwa_pliku_z_konfliktem** i **wykonać commit**

git clone

git clone - **kopiuje całe zdalne repozytorium** (np. z GitHuba) na Twój komputer.

Oznacza to:

- ❑ **pobiera wszystkie pliki**
- ❑ **pobiera całą historię commitów**
- ❑ **ustawia domyślne połączenie z repozytorium (origin)**
- ❑ **tworzy katalog z projektem**

Przykład:
git clone https://github.com/user/projekt.git

git push

git push - wysyła Twoje **lokalne commity** do **zdalnego repozytorium** (np. GitHub, GitLab, Bitbucket).

git push origin nazwa_brancha
gdzie:
origin – nazwa zdalnego repozytorium
nazwa_brancha – gałąź, którą chcesz wysłać (np. main, feature/login)

Nie wysyła stash, plików roboczych, nic co jest nie commit

git pull

git pull - **pobierania i integracji zmian z zdalnego repozytorium do lokalnej kopii projektu. Łączy** ono **operacje git fetch** (pobieranie zmian) **i git merge** (scalanie ich z bieżącą gałęzią), co **pozwala na aktualizację lokalnego kodu** o najnowsze wersje od współpracowników.

git pull wykonuje wewnętrznie:

- **Fetch**: Git pobiera wszystkie nowe commity i branchy z zdalnego repozytorium, ale nie zmienia lokalnych plików.
- **Merge**: Scalane są zmiany z pobranej gałęzi do Twojej lokalnej. Jeśli są konflikty (np. edytowałeś te same linie), Git poprosi o ich ręczne rozwiązanie.

git fetch

git fetch - służy do **pobierania zmian z zdalnego repozytorium** (np. z GitHuba) **do lokalnego repozytorium, bez automatycznego scalania ich z Twoimi lokalnymi gałęziam**i. Aktualizuje ono informacje o zdalnych branchach (np. "origin/main"), co pozwala sprawdzić, co się zmieniło, zanim zdecydujesz się na integrację (np. via merge lub rebase)

git fetch vs git pull

	Co robi?	Czy zmienia pliki lokalne?	Czy łączy zmiany?
git fetch	pobiera zmiany z serwera	❌ NIE	❌ NIE
git pull	pobiera i łączy zmiany	✅ TAK	✅ TAK

Szyfrowanie sieci internetowych HTTPS

HTTPS (Hypertext Transfer Protocol Secure) to bezpieczna wersja protokołu HTTP używana w internecie do przesyłania danych między przeglądarką a serwerem. Dzięki HTTPS komunikacja jest

szyfrowana,

uwierzytelniana

integralna,

co zapewnia ochronę danych użytkownika.

Co daje HTTPS?

Szyfrowanie (encryption)
Dane przesyłane między przeglądarką a serwerem są zaszyfrowane przy pomocy protokołu **TLS (Transport Layer Security)**. Nawet jeśli ktoś przechwyci ruch sieciowy – zobaczy tylko zaszyfrowane, bezużyteczne dane.

Uwierzytelnienie (authentication)
Certyfikat SSL/TLS potwierdza, że strona, z którą się łączysz, jest prawdziwa i należy do właściwej firmy lub osoby

Integralność danych
Zapewnia, że dane nie zostały zmienione po drodze – nikt nie może ich potajemnie edytować, podmienić ani wstrzyknąć złośliwy kod.

Certyfikaty internetowe

Certyfikaty internetowe (zwane także **certyfikatami cyfrowymi**, **SSL/TLS certyfikatami**) to specjalne elektroniczne dokumenty, które służą do **potwierdzania tożsamości stron w internecie** oraz **umożliwiają bezpieczne, szyfrowane połączenia** między użytkownikiem a serwerem.

Jest oparty na standardzie X.509 i zawiera zestaw kluczowych informacji, które służą do weryfikacji tożsamości serwera oraz umożliwienia szyfrowania danych.

certyfikaty internetowe

1️⃣ Potwierdzają tożsamość strony

Certyfikat zapewnia, że strona, którą odwiedzasz, jest rzeczywiście tym, za co się podaje.

Przykład:
Jeśli wpisujesz *https://bank.pl*, certyfikat gwarantuje, że łączysz się z prawdziwym bankiem, a nie z fałszywą stroną.

2️⃣ Umożliwiają szyfrowanie danych

Certyfikat zawiera **klucz publiczny**, używany do bezpiecznego przesyłania danych.

Dzięki temu:

- hasła
- numery kart płatniczych
- dane osobowe

są zaszyfrowane i niemożliwe do odczytania przez osoby trzecie.

3️⃣ Są wystawiane przez zaufane instytucje

Nazywa się je **CA – Certificate Authority**, np.:

- Let's Encrypt
- DigiCert
- GlobalSign
- Sectigo

Przeglądarki ufają tym instytucjom.

Certyfikat internetowy zawiera m.in.:

- **Wersja (Version)**: Określa wersję standardu certyfikatu (np. wersja 3 dla X.509 v3).
- **Numer seryjny (Serial Number)**: Unikalny identyfikator nadany przez urząd certyfikacji (CA).
- **Algorytm podpisu (Signature Algorithm)**: Określa algorytm użyty do podpisania certyfikatu (np. SHA-256 with RSA).
- **Wydawca (Issuer)**: Dane urzędu certyfikacji, który wystawił certyfikat, w tym nazwa, kraj, organizacja itp.
- **Okres ważności (Validity)**: Daty "od" (Not Before) i "do" (Not After), definiujące, kiedy certyfikat jest aktywny.

Certyfikat internetowy zawiera m.in.:

- **Podmiot (Subject)**: Dane właściciela certyfikatu, np. nazwa domeny (Common Name), organizacja, kraj, miasto.
- **Informacje o kluczu publicznym podmiotu (Subject Public Key Info)**: Klucz publiczny serwera oraz algorytm klucza (np. RSA lub ECDSA).
- **Rozszerzenia (Extensions)**: Dodatkowe pola, takie jak Subject Alternative Names (SAN) dla alternatywnych domen, użycie klucza (Key Usage), autorytet certyfikacji itp.
- **Podpis cyfrowy (Signature)**: Podpis urzędu CA, weryfikujący autentyczność certyfikatu.

Rodzaje certyfikatów

➡️ DV – Domain Validation

Najprostsze i najtańsze (np. Let's Encrypt).
Sprawdzana jest tylko domena.

➡️ OV – Organization Validation

Weryfikowana jest firma, która posiada domenę.

➡️ EV – Extended Validation

Najwyższy poziom weryfikacji – pełna weryfikacja organizacji.
Kiedyś prezentowane jako zielony pasek w przeglądarce.

SSL czy TLS

SSL (Secure Sockets Layer): To starszy protokół szyfrowania, opracowany w latach 90. przez Netscape. Jego ostatnie wersje (SSL 2.0 i 3.0) miały poważne luki bezpieczeństwa, dlatego od dawna nie są zalecane i nie są używane w praktyce.

TLS (Transport Layer Security): To ulepszona i bezpieczniejsza wersja SSL, rozwijana od 1999 roku. Aktualne wersje to TLS 1.2 i TLS 1.3 (TLS 1.0 i 1.1 też są przestarzałe i wycofywane). TLS jest standardem de facto w dzisiejszym internecie.

Szyfrowanie end-to-end (E2EE)

to **metoda zabezpieczania komunikacji**, dzięki której wiadomość jest **dostępna tylko dla nadawcy i odbiorcy**. **Dane są szyfrowane na urządzeniu nadawcy i odszyfrowywane dopiero na urządzeniu odbiorcy**, co oznacza, że żaden pośrednik, w tym dostawca usługi, nie ma dostępu do ich treści.

✔️ Przykład:

- WhatsApp,
- Signal

Jak działa?

- **Klucze szyfrujące:** Szyfrowanie end-to-end opiera się na parze kluczy: publicznego (do szyfrowania) i prywatnego (do deszyfrowania).
- **Szyfrowanie u nadawcy**: Wiadomość jest szyfrowana kluczem publicznym nadawcy i wysyłana.
- **Odszyfrowanie u odbiorcy**: Tylko klucz prywatny odbiorcy jest w stanie odszyfrować wiadomość, która dociera do niego w nieczytelnej formie.
- **Brak dostępu dla pośredników:** Żaden serwer pośredniczący ani dostawca usługi nie ma możliwości odszyfrowania wiadomości, ponieważ nie posiada klucza prywatnego.

Zalety

- Zapewnia wysoki poziom prywatności – tylko nadawca i odbiorca mogą odczytać dane, chroniąc przed podsłuchem, hakerami czy szpiegostwem rządowym.
- Dane są bezpieczne nawet w przypadku naruszenia serwera dostawcy, ponieważ są zaszyfrowane.
- Utrudnia masową inwigilację, co jest korzystne dla wolności słowa i ochrony danych osobowych.
- Chroni przed nadużyciami ze strony firm lub rządów, promując zaufanie do usług komunikacyjnych.
- Minimalizuje ryzyko wycieków danych w tranzycie, co jest kluczowe w aplikacjach jak komunikatory (np. Signal, WhatsApp).

Wady

- Jeśli użytkownik zgubi klucz prywatny lub hasło, dane stają się nieodwracalnie niedostępne (brak możliwości odzyskania przez dostawcę).
- Zwiększa złożoność systemu – wymaga prawidłowego zarządzania kluczami, co może być trudne dla mniej technicznych użytkowników.
- Platformy nie mogą monitorować treści, co utrudnia walkę z nielegalnymi materiałami (np. dezinformacją, mową nienawiści czy treściami pedofilskimi).
- Może komplikować egzekwowanie prawa – organy ścigania nie mają dostępu do danych, co budzi kontrowersje w sprawach bezpieczeństwa narodowego.
- Wpływa na wydajność – szyfrowanie i deszyfrowanie wymaga więcej zasobów obliczeniowych, co może spowalniać urządzenia.

Kryptografia

Kryptografia to dziedzina nauki i techniki **zajmująca się ochroną informacji poprzez ich szyfrowanie, deszyfrowanie oraz zapewnianie bezpieczeństwa komunikacji i danych**. Pochodzi od greckich słów "kryptos" (ukryty) i "graphein" (pisać), co dosłownie oznacza "tajne pisanie".

W praktyce kryptografia pozwala na przekształcanie czytelnych danych (zwanych tekstem jawnym) w formę nieczytelną (tekst zaszyfrowany) za **pomocą algorytmów i kluczy,** a następnie na ich odzyskiwanie tylko przez upoważnione osoby.

Główne elementy kryptografii:

- **Szyfrowanie (encryption)**: Proces konwersji danych na formę zaszyfrowaną, aby uniemożliwić nieautoryzowany dostęp.
- **Deszyfrowanie (decryption)**: Odwrotny proces, wymagający klucza do przywrócenia oryginalnej formy danych.
- **Klucze kryptograficzne**: Sekretne wartości używane w algorytmach do szyfrowania i deszyfrowania. Mogą być symetryczne (ten sam klucz do obu procesów) lub asymetryczne (para kluczy: publiczny i prywatny).
- **Algorytmy kryptograficzne**: Matematyczne procedury, np. AES (symetryczny), RSA (asymetryczny) czy SHA (do haszowania).

Zastosowania kryptografii:

- **Bezpieczeństwo online**: W protokołach jak HTTPS (szyfrowanie stron internetowych), VPN czy end-to-end encryption (E2EE) w komunikatorach (np. Signal, WhatsApp).
- **Ochrona danych**: W bankowości (karty płatnicze, transakcje), medycynie (rekordy pacjentów) czy wojskowości (tajne komunikaty).
- **Blockchain i kryptowaluty**: Podstawa Bitcoin i innych, zapewniająca integralność transakcji.
- **Podpisy cyfrowe**: Weryfikacja autentyczności dokumentów lub oprogramowania.

Klucz publiczny

- To klucz, który **możesz udostępniać każdemu**.
- Służy do **szyfrowania wiadomości**, które mają trafić do Ciebie.
- Każdy, kto zna Twój klucz publiczny, może wysłać Ci zaszyfrowaną wiadomość, ale **tylko Ty możesz ją odczytać**.

Przykład:

- Chcesz, żeby ktoś wysłał Ci bezpieczną wiadomość. Podajesz mu swój klucz publiczny, on szyfruje wiadomość, a następnie wysyła do Ciebie.

Klucz prywatny

- To klucz, który **zawsze musisz trzymać w tajemnicy**.
- Służy do **odszyfrowania wiadomości** zaszyfrowanych Twoim kluczem publicznym.
- Tylko osoba posiadająca klucz prywatny może odczytać wiadomość.

Przykład:

- Kiedy otrzymujesz zaszyfrowaną wiadomość, używasz swojego klucza prywatnego, aby odszyfrować jej treść i ją przeczytać.

Klucz symetryczny (Symmetric Key)

- W kryptografii symetrycznej do **szyfrowania i odszyfrowywania** danych używa się **tego samego klucza**.
- Klucz musi być **tajny** i znany tylko nadawcy i odbiorcy.

Przykłady algorytmów: AES, DES.

AES (Advanced Encryption Standard)

AES to **symetryczny algorytm szyfrowania**, co oznacza, że **ten sam klucz** służy do szyfrowania i odszyfrowywania danych. Jest bardzo szybki i wydajny, dlatego stosuje się go do szyfrowania dużych ilości danych.

Jak działa w praktyce:

1. Nadawca szyfruje wiadomość przy użyciu klucza AES.
2. Odbiorca używa **tego samego klucza**, aby odszyfrować wiadomość.

Jak działa w praktyce:

1. Nadawca szyfruje wiadomość za pomocą klucza symetrycznego.
2. Wiadomość trafia do odbiorcy.
3. Odbiorca używa tego samego klucza, aby odszyfrować wiadomość.

Zalety:

- Szybkie szyfrowanie i odszyfrowywanie (wydajne dla dużych danych).

Wady:

- Problem z bezpiecznym przesłaniem klucza – jeśli ktoś przechwyci klucz, może odczytać wszystkie wiadomości.

Klucz asymetryczny (Asymmetric Key)

W kryptografii asymetrycznej używa się **pary kluczy**:

1. **Klucz publiczny** – do szyfrowania wiadomości.
2. **Klucz prywatny** – do odszyfrowania wiadomości.

Klucz publiczny może być **udostępniany każdemu**, a klucz prywatny **pozostaje tajny**.

Przykłady algorytmów: RSA, ECC.

RSA (Rivest–Shamir–Adleman)

RSA to **asymetryczny algorytm szyfrowania**, który używa **pary kluczy**:

- **Klucz publiczny** → do szyfrowania wiadomości.
- **Klucz prywatny** → do odszyfrowania wiadomości.

Umożliwia bezpieczne przesyłanie danych nawet przez niezabezpieczone kanały.

Jak działa w praktyce:

1. Nadawca szyfruje wiadomość kluczem publicznym odbiorcy.
2. Odbiorca używa swojego klucza prywatnego, aby odszyfrować wiadomość.

Jak działa w praktyce:

1. Nadawca szyfruje wiadomość kluczem **publicznym odbiorcy**.
2. Wiadomość trafia w formie zaszyfrowanej.
3. Odbiorca używa swojego **klucza prywatnego**, aby odszyfrować wiadomość.

Zalety:

- Bezpieczne przesyłanie wiadomości, nawet jeśli klucz publiczny jest znany wszystkim.
- Idealne dla komunikacji w Internecie (np. E2EE).

Wady:

- Wolniejsze niż szyfrowanie symetryczne, szczególnie dla dużych plików.

Polityki bezpieczeństwa w sieci

To zestaw zasad, procedur i wytycznych, które mówią, **jak chronić systemy komputerowe, dane i użytkowników przed zagrożeniami w internecie**.
Są one stosowane w firmach, szkołach, organizacjach, a czasem nawet w domowych sieciach.

Można powiedzieć, że polityki bezpieczeństwa to **instrukcja, jak dbać o bezpieczeństwo sieci i danych**.

Po co istnieją polityki bezpieczeństwa?

Aby chronić:

- dane (przed utratą, wyciekiem, zniszczeniem) np.: obowiązek **tworzenia kopii zapasowych (backupów),**
- systemy (przed atakami i awariami) np.: stosowanie **antywirusa i zapory sieciowej (firewall),**
- użytkowników (przed zagrożeniami) np.: polityka **silnych haseł,** automatyczne **wylogowanie po czasie bezczynności.,**
- firmę/szkołę (przed stratami i odpowiedzialnością prawną) np.: zgodność z **RODO** – ochrona danych osobowych**.**

Co zawierają polityki bezpieczeństwa w sieci?

✔ 1. Zasady korzystania z sieci i komputerów

- kto ma dostęp do sieci,
- jakie urządzenia mogą się łączyć,
- jakie oprogramowanie jest dozwolone.

✔ 2. Zasady tworzenia i zarządzania hasłami

- minimalna długość hasła,
- konieczność zmiany co X dni,
- zakaz używania jednego hasła do wszystkiego.

✔ 3. Ochrona przed wirusami i malware

- obowiązkowe aktualizacje,
- używanie antywirusów,
- zakaz otwierania podejrzanych załączników.

Co zawierają polityki bezpieczeństwa w sieci?

✔ **4. Dostęp do danych**
- kto ma dostęp do jakich plików,
- jak przechowywać dane wrażliwe (szyfrowanie).

✔ **5. Zasady pracy zdalnej**
- VPN,
- zabezpieczone Wi-Fi,
- sprawdzone urządzenia.

✔ **6. Kopie zapasowe (backupy)**
- jak często robić,
- gdzie przechowywać,
- kto jest odpowiedzialny.

Co zawierają polityki bezpieczeństwa w sieci?

✔ 7. Reagowanie na incydenty bezpieczeństwa

- co zrobić, jeśli nastąpi:
  - włamanie,
  - infekcja systemu,
  - utrata danych,
  - próba phishingu.

✔ 8. Ochrona sieci

- firewalle,
- filtrowanie stron,
- kontrola ruchu (monitoring sieci),
- aktualizacja systemów.

✔ 9. Polityka prywatności użytkowników

- co można logować,
- jakie dane są zbierane,
- kto je może przeglądać.

Przykłady polityk bezpieczeństwa

- polityka haseł,
- polityka dostępu do sieci Wi-Fi,
- polityka aktualizacji systemów,
- polityka backupów,
- polityka korzystania z urządzeń USB,
- polityka pracy zdalnej,
- polityka zabezpieczenia serwerowni.

Dlaczego polityki bezpieczeństwa są ważne?

Zapobiegają:

- atakom hakerów,
- kradzieży danych,
- infekcjom malware,
- wyciekom danych osobowych (RODO),
- przestojom firmy/szkoły,
- awariom systemów.

To podstawowy element cyberbezpieczeństwa

Cyberbezpieczeństwo

To zbiór:

- zasad,
- technologii,
- procedur i działań,

których celem jest **ochrona systemów komputerowych, sieci, urządzeń i danych** przed:

- atakami,
- kradzieżą,
- zniszczeniem,
- nieuprawnionym dostępem.

Dotyczy komputerów, smartfonów, Internetu, serwerów, chmur, sieci Wi-Fi.

Cyberbezpieczeństwo służy do:

- ochrony **danych osobowych** (np. PESEL, hasła, oceny),
- zabezpieczenia **systemów informatycznych**,
- zapewnienia **ciągłości działania** firmy, szkoły lub instytucji,
- ochrony **użytkowników** przed oszustwami i zagrożeniami w sieci,
- zapobiegania **atakom hakerskim i wirusom**.

Zalety cyberbezpieczeństwa

Najważniejsze zalety:

- **ochrona danych** przed wyciekiem i kradzieżą,
- **bezpieczne działanie systemów** i sieci,
- **ochrona użytkowników** przed phishingiem i malware,
- **mniejsze straty finansowe**,
- **zgodność z prawem** (np. RODO),
- **ochrona reputacji** szkoły lub firmy.

Przykłady zagrożeń, przed którymi chroni cyberbezpieczeństwo:

- phishing (fałszywe maile),
- wirusy i ransomware,
- kradzież danych,
- ataki DDoS,
- podsłuchiwanie sieci Wi-Fi.

Cyberbezpieczeństwo zagrożenia i zabezpieczenia

Zagrożenie	Zabezpieczenie
Phishing (fałszywe e-maile, SMS-y)	szkolenia użytkowników, filtry antyspamowe
Wirusy i malware	program antywirusowy, aktualizacje systemu
Ransomware (blokada plików)	kopie zapasowe (backup), antywirus
Kradzież haseł	silne hasła, uwierzytelnianie dwuetapowe (2FA)
Nieautoryzowany dostęp	kontrola dostępu, loginy i role

Cyberbezpieczeństwo zagrożenia i zabezpieczenia

Wycieki danych	szyfrowanie danych, ograniczenie uprawnień
Ataki hakerskie	firewall, monitoring sieci
Podsłuch w sieci Wi-Fi	szyfrowanie Wi-Fi (WPA2/WPA3)
Utrata danych (awaria sprzętu)	regularne kopie zapasowe
Błędy użytkownika	szkolenia, procedury bezpieczeństwa

Przykłady systemów infor. opartych na chmurach

Platformy chmurowe (IaaS / PaaS)

- **Amazon Web Services (AWS)** – EC2, S3, Lambda
- **Microsoft Azure** – App Services, Azure SQL, Functions
- **Google Cloud Platform (GCP)** – Compute Engine, BigQuery
- **Oracle Cloud Infrastructure**

Systemy biznesowe (SaaS)

- **Salesforce** – CRM
- **SAP S/4HANA Cloud** – ERP
- **Microsoft Dynamics 365**
- **Workday** – HR i finanse
- **HubSpot** – marketing i CRM

Komunikacja i praca zespołowa

- **Microsoft 365 (Office 365)** – Outlook, Teams, OneDrive
- **Google Workspace** – Gmail, Drive, Docs
- **Slack**
- **Zoom**

Budowa systemów infor. opartych na chmurach

System informatyczny oparty na chmurze to zbiór **współpracujących warstw**, które razem umożliwiają przechowywanie danych, przetwarzanie informacji oraz dostęp do aplikacji przez Internet.

Warstwa infrastruktury (IaaS – Infrastructure as a Service)

To **podstawa całego systemu**. Składa się z:

- **serwerów fizycznych** (data center dostawcy chmury)
- **maszyn wirtualnych**
- **pamięci masowych** (dyski, obiekty – np. S3)
- **sieci komputerowych** (load balancery, firewalle)

Zapewnia:

- moc obliczeniową
- przestrzeń dyskową
- połączenia sieciowe

Przykład: AWS EC2, Azure Virtual Machines

Warstwa platformy (PaaS – Platform as a Service)

Ułatwia tworzenie i uruchamianie aplikacji.

Zawiera:

- systemy operacyjne
- środowiska uruchomieniowe (Java, .NET, Node.js)
- serwery aplikacyjne
- bazy danych
- narzędzia developerskie

Programista **nie zarządza serwerem**, tylko kodem.

Przykład: Google App Engine, Azure App Service

Warstwa aplikacji (SaaS – Software as a Service)

To **gotowe aplikacje dostępne przez przeglądarkę**.
Charakterystyka:

- brak instalacji lokalnej
- dostęp z dowolnego miejsca
- aktualizacje po stronie dostawcy

Przykłady:

- Gmail
- Microsoft 365
- Salesforce
- Dropbox

Warstwa danych

Odpowiada za:

- przechowywanie danych
- kopie zapasowe (backup)
- replikację danych
- bezpieczeństwo informacji

Rodzaje baz danych:

- relacyjne (np. Azure SQL, Amazon RDS)
- nierelacyjne (NoSQL – MongoDB, DynamoDB)

Warstwa bezpieczeństwa

Chroni system i dane użytkowników.

Obejmuje:

- uwierzytelnianie (logowanie)
- autoryzację (role, uprawnienia)
- szyfrowanie danych
- zapory sieciowe
- monitorowanie zagrożeń

Bardzo ważna w chmurze publicznej.

Warstwa zarządzania i monitoringu

Umożliwia:

- kontrolę zasobów
- skalowanie systemu
- monitorowanie wydajności
- rozliczanie zużycia (pay-as-you-go)

Przykłady narzędzi:

- AWS CloudWatch
- Azure Monitor

Warstwa dostępu użytkownika

To interfejs, przez który użytkownik korzysta z systemu:

- przeglądarka internetowa
- aplikacja mobilna
- API

Schemat uproszczony

Pliki, bazy danych , aplikacje w Chmurze

Chmura obliczeniowa (cloud computing) to model dostarczania zasobów IT do których masz dostęp przez internet, gdzie dane i aplikacje są przechowywane na zdalnych serwerach zarządzanych przez dostawców takich jak:

- Amazon Web Services (AWS),
- Google Cloud
- Microsoft Azure

Zamiast trzymać wszystko na lokalnym komputerze lub serwerze, dane są umieszczane w centrach danych rozproszonych po świecie. Czyli dane i aplikacje nie znajdują się na Twoim komputerze ani w serwerowni firmy, tylko na fizycznych serwerach należących do dostawcy chmury

Chmura i jej usługi

Przechowywanie odbywa się za pomocą specjalistycznych usług, które zapewniają skalowalność, redundancję (kopie zapasowe) i dostępność.
Do tych usług należą między innymi

1. **Object Storage** (najczęściej spotykane). **Object Storage** (przechowywanie obiektowe) to **sposób przechowywania danych w chmurze**, w którym pliki są trzymane jako **niezależne obiekty**, a nie jak na dysku w folderach. Przykłady:
   - AWS S3
   - Azure Blob Storage
   - Google Cloud Storage

Przechowuje:

- pliki
- zdjęcia
- PDF-y
- backupy
- logi

Chmura i jej usługi, c.d.

2. **Block Storage** (wirtualne dyski) to **rodzaj przechowywania danych w chmurze**, który działa **jak klasyczny dysk twardy** podłączony do komputera lub serwera.
Przykłady:

- AWS EBS
- Azure Managed Disks

Przechowuje:

- system plików
- dane baz danych
- aplikacje

Chmura i jej usługi, c.d.

3. **File Storage** (jak sieciowy dysk). to **współdzielone miejsce na pliki**, do którego **wiele komputerów lub serwerów ma jednoczesny dostęp**, tak jak do **dysku sieciowego w firmie**. Przykłady:

- AWS EFS
- Azure Files

Przechowuje:

- pliki współdzielone
- uploady użytkowników

4. **Bazy danych** jako usługa. Przykłady:

- AWS RDS
- Azure SQL Database
- Cloud SQL

Przechowuje:

- dane strukturalne

Chmura i jej usługi, c.d.

5. **Backup / Archive Storage** to **specjalny typ przechowywania danych w chmurze**, który służy **do kopii zapasowych i długoterminowego archiwizowania danych**, a nie do codziennej pracy z plikami.
Przykłady:

- AWS Glacier
- Azure Archive Storage

Przechowuje:

- archiwa
- dane rzadko używane

1. Przechowywanie plików w chmurze

Pliki (np. dokumenty, zdjęcia, wideo) są przechowywane w usługach typu **object storage** lub **block storage**.

To działa jak wirtualny dysk twardy, ale dane są dzielone na obiekty (pliki) i przechowywane w rozproszonych systemach. Dostawca chmury zarządza replikacją danych (kopiami w różnych lokalizacjach), aby zapobiec utracie.

Przechowywanie plików w chmurze

Jak to działa: Użytkownik uploaduje plik przez API, aplikację lub interfejs webowy. Plik jest dzielony na bloki, szyfrowany i rozproszony po serwerach. Dostęp odbywa się przez unikalny URL lub klucz dostępu.

Przykład: Amazon S3 (Simple Storage Service) – możesz wrzucić plik PDF, a S3 automatycznie stworzy kopie w różnych regionach geograficznych dla bezpieczeństwa.

Zastosowanie: W aplikacjach mobilnych, np. Instagram przechowuje zdjęcia użytkowników w chmurze, co pozwala na szybki dostęp z dowolnego urządzenia.

2. Przechowywanie baz danych w chmurze

Bazy danych są hostowane w usługach **managed database**, gdzie dostawca chmury zarządza serwerem, aktualizacjami i backupami.

Mogą to być relacyjne bazy (SQL, jak MySQL) lub nierelacyjne (NoSQL, jak MongoDB).

Przechowywanie baz danych

Jak to działa: Tworzysz instancję bazy, łączysz się z nią przez połączenie sieciowe (np. JDBC). Dane są przechowywane na dyskach chmurowych z automatyczną replikacją i skalowaniem (np. dodawanie mocy obliczeniowej w razie wzrostu obciążenia).

Przykład: Google Cloud SQL – firma e-commerce przechowuje dane klientów (imiona, zamówienia) w bazie MySQL hostowanej w chmurze, co pozwala na automatyczne skalowanie podczas Black Friday.

Zastosowanie: W aplikacjach webowych, np. Netflix używa chmurowych baz do przechowywania preferencji użytkowników, co umożliwia personalizację treści na skalę globalną.

3. Przechowywanie aplikacji

Aplikacje są wdrażane w chmurze za pomocą **kontenerów** (np. Docker), orkiestrantów (Kubernetes) lub modeli **serverless** (bez serwerowych).

Kod aplikacji jest pakowany w kontener i uruchamiany na wirtualnych maszynach lub funkcjach chmurowych.

Przechowywanie aplikacji

Jak to działa: Aplikacja jest budowana jako obraz (image), uploadowana do repozytorium chmurowego, a potem uruchamiana na żądanie. Chmura zarządza zasobami (CPU, RAM) dynamicznie.

Przykład: AWS Lambda – aplikacja do przetwarzania obrazów (np. automatyczne tagowanie zdjęć) jest przechowywana jako funkcja kodu, która uruchamia się tylko wtedy, gdy ktoś uploaduje plik, bez stałego serwera.

Zastosowanie: W DevOps, np. Uber deployuje aplikacje w chmurze, co pozwala na szybkie aktualizacje i skalowanie floty serwerów w zależności od liczby użytkowników.

Ogólne zastosowanie chmury

Chmura jest używana w biznesie, edukacji i codziennym życiu.

Na przykład, firma startupowa może przechowywać pliki w Dropbox (opartym na chmurze), bazy danych w Azure SQL i aplikacje w Heroku, co pozwala na szybki rozwój bez inwestycji w sprzęt. W medycynie chmura przechowuje dane pacjentów (zgodne z GDPR/HIPAA), enabling telemedycynę.

Zalety

Zalety

- **Skalowalność**: Łatwo zwiększyć zasoby (np. z 1 TB do 100 TB) bez kupowania nowego sprzętu.
- **Dostępność**: Dane dostępne z dowolnego miejsca z internetem, 24/7, z wysoką redundancją (np. 99.99% uptime).
- **Koszt**: Płacisz tylko za użyte zasoby (pay-as-you-go), co jest tańsze niż utrzymanie własnego serwera.
- **Bezpieczeństwo i backup**: Automatyczne szyfrowanie, firewalle i kopie zapasowe.
- **Łatwość zarządzania**: Dostawcy obsługują aktualizacje i maintenance.

Wady

Wady

- **Zależność od internetu**: Bez połączenia nie masz dostępu do danych.
- **Koszty ukryte**: Przy dużym zużyciu (np. transfer danych) rachunki mogą wzrosnąć nieoczekiwanie.
- **Bezpieczeństwo**: Ryzyko wycieków danych (np. ataki hakerskie), choć dostawcy mają zabezpieczenia – zależy od konfiguracji użytkownika.
- **Utrata kontroli**: Dane są u trzeciej strony, co może budzić obawy o prywatność lub zgodność z prawem (np. gdzie fizycznie są serwery?).
- **Opóźnienia (latency)**: Dla aplikacji wymagających szybkiego dostępu (np. gry online) chmura może być wolniejsza niż lokalny serwer.