Architektura Cassandry: pierścień, węzeł, replikacja

Poziom: podstawy **Dotyczy: Apache Cassandra 3.x / 4.x** **Czas: ~25 min**

Cassandra nie ma węzła głównego — każdy węzeł w klastrze jest sobie równy, a to, jak dane trafiają na dysk i jak wracają do klienta, wynika z jednej struktury: pierścienia tokenów.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
CAP	Client Access Point: punkt dostępu klienta, czyli nazwa i adres IP roli klastra, pod którymi użytkownicy łączą się z usługą niezależnie od tego, który węzeł ją prowadzi
DC	Domain Controller: serwer, który trzyma Active Directory i loguje użytkowników
RF	Replication Factor: liczba kopii tego samego wiersza trzymanych naraz w klastrze Cassandry
WMS	Warehouse Management System: system zarządzania magazynem

Po co to istnieje

> **Pojęcie: węzeł.** Pojedynczy serwer biorący udział w klastrze bazy danych, przechowujący fragment danych i odpowiadający na zapytania.

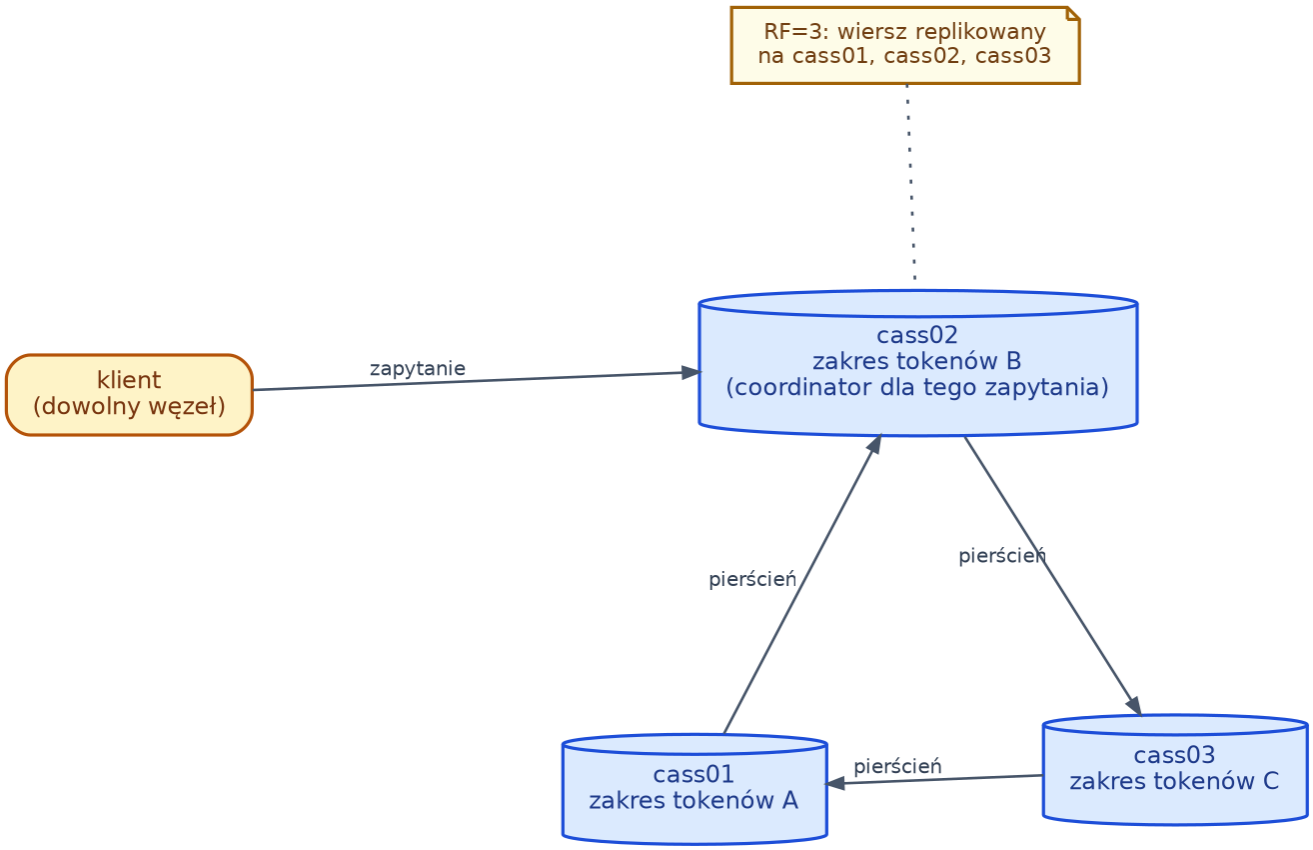
> **Pojęcie: replika.** Kolejna kopia tego samego wiersza danych, trzymana na innym węźle, żeby awaria jednego z nich nie oznaczała utraty dostępu do danych.

Model relacyjny, opisany na stronach fundamentów, zakłada zwykle jeden serwer albo klaster z jasno wydzieloną rolą — węzeł podstawowy przyjmuje zapisy, repliki tylko czytają. Cassandra powstała z przeciwnego założenia: żaden pojedynczy węzeł nie może być punktem awarii ani wąskim gardłem, więc architektura eliminuje pojęcie węzła głównego całkowicie. Każdy węzeł potrafi przyjąć zapis, każdy potrafi odpowiedzieć na odczyt, a decyzję, który węzeł faktycznie przechowuje dany wiersz, podejmuje deterministyczna funkcja skrótu, nie konfiguracja administratora.

Konsekwencją tego wyboru jest to, że pytania, które w bazie relacyjnej są oczywiste — „gdzie jest ten wiersz?", „czy odczytam najświeższą wersję?" — w Cassandrze wymagają zrozumienia mechaniki pierścienia, replikacji i poziomów spójności. To, czym różni się model kolumnowy od relacyjnego na poziomie koncepcyjnym, opisuje strona Relacyjne, dokumentowe, kolumnowe, klucz-wartość: mapa rodzin — tutaj zajmujemy się wyłącznie tym, jak Cassandra jest zbudowana w środku.

Pierścień i zakresy tokenów

> **Pojęcie: pierścień.** Logiczna struktura, w której wszystkie węzły klastra są ułożone w okrąg według przypisanych im wartości liczbowych — tokenów. Przestrzeń możliwych tokenów jest skończona i cykliczna (od wartości minimalnej do maksymalnej, a potem z powrotem do minimalnej), więc każdy węzeł odpowiada za pewien wycinek tego okręgu — zakres tokenów między swoim tokenem a tokenem poprzedniego węzła w kierunku wskazówek zegara.



Trzy wezly ulozone w pierscien, kazdy z wlasnym zakresem tokenow, i wiersz replikowany na wszystkie trzy (kliknięcie otwiera pełny rozmiar)

Pojęcie: token. Liczba przypisana węzłowi (albo, przy vnodes, wielu liczb przypisanych temu samemu węzłowi), która wyznacza jego pozycję na pierścieniu. Każdy klucz partycji, zanim trafi do konkretnego węzła, przechodzi przez funkcję haszującą partycjonera (domyślnie `Murmur3Partitioner`), która zamienia go na liczbę z tej samej przestrzeni tokenów. Węzeł odpowiedzialny za dany wiersz to ten, którego zakres tokenów obejmuje wynik tego hasza — nie ten, którego administrator „przypisał" do danej tabeli. Sposób, w jaki klucz partycji trafia do funkcji haszującej, opisuje strona Model danych: keyspace, tabela, klucz partycji, klucz klastrujący.

Pojęcie: vnodes. Zamiast przypisać każdemu fizycznemu węzłowi jeden token (jeden ciągły zakres na pierścieniu), nowoczesna Cassandra przypisuje mu domyślnie 256 mniejszych, rozproszonych po całym pierścieniu tokenów wirtualnych. Efekt jest taki, że dane, za które odpowiadał węzeł, który właśnie odpadł z klastra, są rozłożone na wielu pozostałych węzłach zamiast skupione na jednym sąsiedzie — dołączenie albo usunięcie węzła rozkłada obciążenie równomiernie zamiast przenosić cały ciężar na najbliższego sąsiada w pierścieniu. Liczbę tokenów widzimy wprost w wyniku `nodetool info`, opisanym na stronie nodetool: stan klastra i operacje administracyjne.

Węzeł, rack, datacenter

Pojęcie: węzeł. Pojedynczy proces `cassandra` (jedna instancja bazy) uczestniczący w klastrze, przechowujący swój wycinek danych i biorący udział w gossipie opisanym niżej. W tej książce węzły nazywamy `cass01`, `cass02`, `cass03`.

Pojęcie: rack. Grupa węzłów deklarowana administracyjnie jako fizycznie bliska sobie — ten sam szereg szaf, ta sama strefa dostępności w chmurze. Cassandra unika umieszczania więcej niż jednej repliki tego samego wiersza w tym samym racku, o ile jest to możliwe przy zadanym `replication_factor` — dzięki temu awaria całej szafy nie zabiera jednocześnie dwóch kopii tych samych danych.

Pojęcie: datacenter (DC) w Cassandrze. Największa jednostka topologii — grupa racków, zwykle odpowiadająca fizycznej lokalizacji albo logicznemu podziałowi ruchu (na przykład osobny DC pod obciążenie analityczne, żeby ciężkie zapytania nie odbijały się na latencji ruchu produkcyjnego). Przynależność węzła do racku i DC deklarujemy w pliku `cassandra-rackdc.properties`

, opisanym na stronie Konfiguracja i diagnostyka.

Replication factor i strategia replikacji

Pojęcie: replication factor (RF). Liczba określająca, na ile węzłów będzie replikowany dany keyspace — czyli ile kopii każdego wiersza istnieje jednocześnie w klastrze. RF ustawiany jest per keyspace, nie globalnie dla całego klastra, więc różne keyspace w tym samym klastrze mogą mieć różny poziom replikacji.

To, na jakie konkretnie węzły dany wiersz trafi, decyduje już strategia replikacji, nie samo RF — RF mówi tylko „ile", strategia mówi „gdzie".

Pojęcie: SimpleStrategy. Najprostsza strategia replikacji — rozkłada kopie danych po kolejnych węzłach na pierścieniu, idąc zgodnie z ruchem wskazówek zegara od węzła wskazanego przez token, zupełnie ignorując podział na racki i datacenter. Dla klastra w jednym DC działa poprawnie, ale w klastrze wielo-DC może umieścić wszystkie kopie w tym samym centrum danych, robiąc replikację bezużyteczną w razie awarii całego DC.

Pojęcie: NetworkTopologyStrategy. Strategia świadoma topologii — RF deklarujemy osobno dla każdego datacenter, a strategia w obrębie każdego z nich stara się dodatkowo rozłożyć repliki po różnych rackach. To jedyna strategia sensowna w środowisku produkcyjnym i jedyna, która ma sens przy więcej niż jednym DC. Pełną składnię `CREATE KEYSPACE` i `ALTER KEYSPACE` dla obu strategii pokazujemy na stronie CQL: zapytania i operacje na danych.

Strategia	Świadoma racków i DC	Sensowna w produkcji wielo-DC
SimpleStrategy	nie	nie
NetworkTopologyStrategy	tak	tak

Spójność ostateczna i poziomy spójności

> **Pojęcie: spójność ostateczna (eventual consistency).** Gwarancja słabsza od spójności silnej znanej z transakcji relacyjnych — mówi, że jeśli zapis do repliki się powiedzie, to wszystkie repliki tego wiersza *w końcu* będą miały tę samą wartość, ale nie gwarantuje, że stanie się to natychmiast ani że każdy kolejny odczyt zobaczy już najnowszą wersję. W klastrze z RF = 3 zapis może zostać potwierdzony klientowi, zanim wszystkie trzy repliki faktycznie zapiszą dane na dysk — pozostałe dwie dogonią pierwszą z opóźnieniem rzędu milisekund, w normalnych warunkach sieciowych.

> **Pojęcie: poziom spójności (consistency level).** Parametr ustawiany per zapytanie (nie globalnie dla całej bazy), który mówi, ile replik musi potwierdzić operację, zanim Cassandra uzna ją za udaną i odpowie klientowi. To jest strojenie kompromisu między spójnością a dostępnością i latencją osobno dla każdego zapytania, nie decyzja podejmowana raz na cały klaster.

Przy RF = 3 (trzy kopie każdego wiersza) najczęściej spotykane poziomy zachowują się tak:

Poziom	Ile replik musi odpowiedzieć przy RF = 3	Skutek
`ONE`	1	najniższa latencja, najsłabsza gwarancja świeżości danych
`QUORUM`	2 (większość z 3)	kompromis — odporny na awarię jednej repliki
`LOCAL_QUORUM`	2, ale tylko replik w lokalnym DC	jak `QUORUM`, bez czekania na odpowiedź z odległego datacenter
`ALL`	3 (wszystkie)	najsilniejsza gwarancja, zero tolerancji na awarię pojedynczej repliki

> **Pojęcie: kworum.** Liczba obliczana jako `(RF / 2) + 1`, zaokrąglona w dół przy dzieleniu całkowitym. Dla RF = 3 kworum wynosi 2, dla RF = 5 wynosi 3. Poziom `QUORUM` wymaga odpowiedzi od tylu replik, ile wynosi kworum — nie od większości w sensie potocznym, tylko od tej konkretnej, wyliczonej liczby.

Wzór R + W > RF. Jeśli poziom spójności odczytu (R, liczba replik odpytywanych przy `SELECT`) i poziom spójności zapisu (W, liczba replik, które muszą potwierdzić `INSERT`/`UPDATE`) spełniają

nierówność R + W > RF, to każdy odczyt musi trafić na przynajmniej jedną replikę, która widziała najnowszy zapis — bo zbiór replik odpytanych przy odczycie i zbiór replik, które potwierdziły zapis, matematycznie muszą się nachodzić. Na liczbach: przy RF = 3, W = QUORUM (2) i R = QUORUM (2), mamy 2 + 2 = 4 > 3 — silna spójność jest zagwarantowana, bo nie da się wybrać dwóch replik do zapisu i dwóch do odczytu spośród trzech bez choć jednej wspólnej. Przy R = ONE (1) i W = ONE (1) mamy 1 + 1 = 2, co nie przekracza RF = 3 — odczyt może trafić na replikę, która jeszcze nie dostała zapisu, i zwrócić dane nieaktualne.

Gossip, snitch i coordinator node

Pojęcie: gossip. Protokół, którym węzły Cassandry wymieniają się informacją o stanie klastra — który węzeł żyje, jaki ma load, jaką wersję schematu — bez centralnego rejestru. Co sekundę każdy węzeł wybiera losowo kilku innych węzłów i wymienia z nimi swoją wiedzę o reszcie klastra, więc informacja o zmianie stanu jednego węzła dociera do wszystkich w kilka sekund, nawet w dużym klastrze. Stan gossipu dla każdego węzła widzimy komendą `nodetool gossipinfo`, opisaną na stronie nodetool: stan klastra i operacje administracyjne. Właściwość `Gossip active` w wyniku `nodetool info` mówi, czy dany węzeł w ogóle bierze udział w tej wymianie — jeśli nie, węzeł widoczny jest w klastrze jako `DN`, mimo że sam proces `cassandra` może działać poprawnie.

Pojęcie: snitch. Komponent, który mówi Cassandrze, do jakiego racku i datacenter należy dany węzeł, i na tej podstawie liczy „odległość" między węzłami przy podejmowaniu decyzji o replikacji i o tym, którą replikę odpytać przy odczycie. `GossipingPropertyFileSnitch` (czytający przynależność z lokalnego pliku `cassandra-rackdc.properties` i propagujący ją dalej przez gossip) jest wyborem właściwym w większości wdrożeń produkcyjnych.

Pojęcie: coordinator node. Węzeł, do którego klient faktycznie się połączył i wysłał zapytanie — niekoniecznie ten, który przechowuje dane. Coordinator na podstawie tokenu wylicza, które węzły są odpowiedzialne za dany wiersz, przekazuje im zapytanie, czeka na tyle odpowiedzi, ile wymaga poziom spójności, i dopiero wtedy odpowiada klientowi. Każdy węzeł w klastrze może pełnić rolę coordinatora dla dowolnego zapytania — to kolejny przejaw braku węzła głównego.

Trzy mechanizmy uzgadniania stanu

Rozproszenie danych po wielu węzłach i dopuszczenie potwierdzania zapisu przed dotarciem do wszystkich replik oznacza, że repliki będą się od siebie chwilowo różnić. Cassandra utrzymuje je w zgodzie trzema niezależnymi mechanizmami, każdym reagującym na inny rodzaj rozjazdu.

Pojęcie: hinted handoff. Mechanizm uruchamiany w chwili zapisu, gdy jedna z docelowych replik jest chwilowo niedostępna. Coordinator zapisuje u siebie „podpowiedź" (hint) — informację, jaki zapis należał się nieobecnej replice — i odtwarza go automatycznie, gdy tamten węzeł wróci do klastra. Hinty mają ograniczony czas życia (domyślnie 3 godziny w `max_hint_window_in_ms`); po jego przekroczeniu odzyskanie spójności zostaje w całości w gestii pozostałych dwóch mechanizmów.

Pojęcie: read repair. Mechanizm uruchamiany przy okazji zwykłego odczytu. Gdy coordinator pyta więcej niż jedną replikę (co dzieje się zawsze przy poziomach silniejszych niż `ONE`, a sporadycznie nawet przy `ONE`, zgodnie z parametrem `read_repair_chance`), porównuje otrzymane wersje wiersza i jeśli się różnią, wysyła najnowszą wersję do repliki, która miała starszą — naprawa dzieje się jako efekt uboczny odczytu, bez dodatkowego zadania administracyjnego.

Pojęcie: anti-entropy repair (naprawa pełna). Jedyny z trzech mechanizmów wymagający ręcznego albo zaplanowanego uruchomienia — komendą `nodetool repair`. Węzły porównują ze sobą drzewa Merkle (skrócone, hierarchiczne sumy kontrolne danych w danym zakresie tokenów) i wymieniają tylko te fragmenty danych, w których drzewa się różnią, zamiast przesyłać wszystko. To jedyny mechanizm, który dociera też do wierszy, których nikt nigdy nie odczytał (więc read repair ich nie dotknął) i do zapisów starszych niż okno hintów. Dlaczego repair musi być wykonywany cyklicznie i co się stanie, jeśli tego zaniedbamy, rozwija strona Utrzymanie klastra: repair, compaction, czyszczenie.

Mechanizm	Kiedy działa	Wymaga ręcznego uruchomienia
Hinted handoff	w chwili zapisu, gdy replika jest chwilowo niedostępna	nie
Read repair	przy okazji odczytu, gdy odpytane repliki się różnią	nie
Anti-entropy repair	na żądanie albo wg harmonogramu	tak

Twierdzenie CAP i miejsce Cassandry

Ogólne brzmienie twierdzenia CAP i najczęstsze błędne jego odczytanie opisuje strona Relacyjne, dokumentowe, kolumnowe, klucz-wartość: mapa rodzin — tu tylko lokujemy na nim Cassandrę, bez mitologii, która wokół tego narosła.

Cassandra w chwili partycji sieci domyślnie wybiera dostępność — węzeł odcięty od reszty klastra dalej odpowiada na zapytania, potwierdzając zapisy przy poziomie `ONE` nawet bez łączności z pozostałymi replikami, i dogania resztę klastra przez hinted handoff oraz repair, gdy partycja się skończy. To bywa streszczane jako „Cassandra to baza AP", i w konfiguracji domyślnej to trafny skrót. Ale poziomy spójności opisane wyżej pozwalają przesunąć tę wagę w stronę spójności per zapytanie — `ALL` w połączeniu z partycją sieci oznacza, że część replik jest nieosiągalna i zapytanie zwróci błąd zamiast nieaktualnych danych, czyli zachowanie bliższe stronie C. Twierdzenie CAP nie jest więc etykietą przyklejoną raz na zawsze do silnika — jest opisem tego, co silnik *domyślnie* robi w chwili awarii sieci, i Cassandra jest jednym z niewielu silników, które to zachowanie oddają administratorowi jako parametr zapytania, a nie ukrywają za jedną globalną decyzją.

Z życia wzięte

> **Z ŻYCIA WZIĘTE**
>
> O 6:00 w Swarzędzu czterdzieści skanerów loguje się naraz do WMS, a razem z nimi telemetria wózków widłowych zaczyna zapisywać się do klastra cass01, cass02 i cass03. Tomek sprawdza, który węzeł faktycznie odpowiada za dany wózek, i odkrywa, że to nie administrator decyduje o tym przypisaniu, tylko funkcja haszująca identyfikator wózka na pierścieniu tokenów. Kiedy cass02 na chwilę traci łączność sieciową podczas porannego szczytu, reszta klastra dalej przyjmuje zapisy z poziomem spójności ONE, a zaległe dane dogonią go automatycznie przez hinted handoff. Marek z magazynu nawet nie zauważa przerwy, bo system śledzenia wózków działa bez przestoju.

Częste błędy

Mylenie replication factor z liczbą replik biorących udział w konkretnej operacji. RF = 3 mówi, że istnieją trzy kopie wiersza — nie mówi, ile z nich musi odpowiedzieć na dany `SELECT` czy `INSERT`. To reguluje osobno poziom spójności tej konkretnej operacji.

SimpleStrategy w klastrze wielo-DC. Strategia nieświadoma topologii potrafi umieścić wszystkie kopie danych w jednym datacenter, mimo że fizycznie klaster rozciąga się na dwa — awaria tego jednego DC oznacza wtedy utratę dostępu do danych, których replikacja miała temu zapobiec.

Traktowanie `ONE` jako błędu konfiguracyjnego. Niski poziom spójności bywa świadomym wyborem — dla zapisu telemetrii, gdzie pojedynczy nieaktualny odczyt nic nie kosztuje, `ONE` daje realny zysk w latencji i przepustowości, którego `QUORUM` by nie dał.

□□ Ćwiczenia

> Poniższe pytania sprawdzają rozumienie mechanizmu, nie wymagają uruchomionego klastra.

Pytania kontrolne

1. Klaster ma trzy węzły w jednym racku i RF = 3. Dlaczego świadomość racków, którą oferuje NetworkTopologyStrategy, w tym konkretnym przypadku nic nie daje?

Odpowiedź

Bo wszystkie trzy węzły i tak są w jednym racku — strategia świadoma racków rozkłada repliki między różne racki tylko wtedy, gdy więcej niż jeden rack istnieje. Przy jednym racku efekt jest identyczny jak przy SimpleStrategy, mimo innej nazwy strategii w definicji keyspace.

2. Przy RF = 5 zapis wykonano z poziomem `QUORUM`, a odczyt zaraz po nim z poziomem `ONE`. Czy odczyt ma gwarancję zobaczenia tego zapisu? Ile wynosi kworum przy RF = 5?

Odpowiedź

Kworum przy RF = 5 wynosi 3 (`(5/2)+1` = 3, przy dzieleniu całkowitym). W = 3, R = 1, więc R + W = 4, co nie przekracza RF = 5 — nierówność R + W > RF nie jest spełniona, więc odczyt nie ma matematycznej gwarancji trafienia na replikę, która już widziała ten zapis.

3. Węzeł jest widoczny w `nodetool status` jako `DN`, ale proces `cassandra` na nim działa i przyjmuje połączenia lokalnie. Który z trzech mechanizmów uzgadniania — hinted handoff, read repair, anti-entropy repair — najprędzej ucierpi na tej sytuacji, i dlaczego akurat ten?

Odpowiedź

Hinted handoff — bo hinty mają ograniczony czas życia (domyślnie 3 godziny). Jeśli węzeł jest odcięty od gossipu dłużej niż to okno, nagromadzone dla niego podpowiedzi wygasają i nie zostaną odtworzone automatycznie po jego powrocie — odzyskanie spójności zostanie wtedy wyłącznie w gestii read repair i uruchomionego ręcznie `nodetool repair`.

Powiązane: Relacyjne, dokumentowe, kolumnowe, klucz-wartość: mapa rodzin · Model danych: keyspace, tabela, klucz partycji, klucz klastrujący · Utrzymanie klastra: repair, compaction, czyszczenie · nodetool: stan klastra i operacje administracyjne

Źródła: dokumentacja Apache Cassandra, rozdz. „Architecture" · notatki własne z eksploatacji klastrów produkcyjnych

---

Revision #43
Created 2026-09-24 23:34:48 UTC by Tomasz Konieczny
Updated 2026-10-26 14:13:03 UTC by Tomasz Konieczny