Systemy rozproszone

Apache Zookeeper

Problem

- Zapewnienie najlepszego poziomu usług w środowisku rozproszonym gdzie usługi są zlokalizowane na wielu węzłach.
- W środowisku w którym pracują usługi zachodzą cały czas zmiany takie jak migracja na nowsze serwery lub do innych lokalizacji oraz rekonfiguracja oprogramowania.
- Problem – jak zmieniać konfiguracje kiedy aplikacje (usługi) działają w wielu (10, 100, 1000, …) instancjach na różnych węzłach.

Problem

- Można dokonać zmian i wymusić restart poszczególnych aplikacji po kolei na wszystkich serwerach -> dużo czasu, aplikacje (usługi) w tym czasie nie działają
- Najlepszym rozwiązaniem byłaby możliwość wprowadzenia zmian w konfiguracji we wszystkich działających instancjach aplikacji (usług) na wielu węzłach z bardzo małym opóźnieniem.

- Rozwiązanie - **Apache Zookeeper**.

Zookeeper

- Scentralizowany system do koordynowania rozproszonych serwisów.
  - utrzymuje wspólną konfigurację współdzieloną przez wszystkie węzły w systemie,
  - udostępnia wspólną przestrzeń nazw,
  - udostępnia rozproszoną synchronizację (blokady, kolejki, …)
  - umożliwia tworzenie grup serwisów,
- Umożliwia oprócz zarządzania konfiguracją także synchronizowanie stanu aplikacji, zarządzanie aplikacjami oraz obsługę kolejek.

Zookeeper

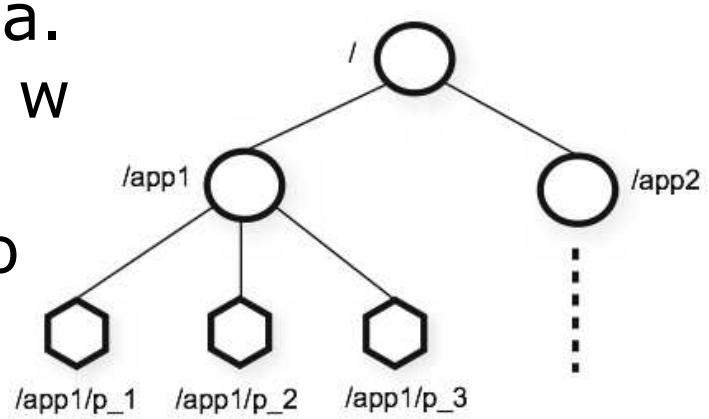
- Scentralizowany system do koordynowania rozproszonych serwisów.
  - projekt utrzymywany jest przez ASF (Apache Software Foundation) jako „Top Level Project”, wcześniej był rozwijany w ramach Hadoopa,
  - serwer ZooKeepera został zaimplementowany w języku Java, udostępniono również API w języku C,
  - Istnieją także implementacje między innymi w Javascript (node-zookeeper-client), .NET (ZooKeeperNetEx) i Python (Kazoo)

Zookeeper – zastosowanie

- Facebook
  - koordynacja i zarządzanie konfiguracją w różnych usługach.
- Yahoo!
  - zarządzanie rozproszonymi zasobami i utrzymania stabilności klastra.
- Netflix
  - koordynacja pomiędzy różnymi komponentami, utrzymanie wysokiej dostępności i odporności na awarie.
- Airbnb
  - koordynacja różnych systemów rozproszonych i usług.
- X (Twitter)
  - utrzymanie spójności i niezawodności, obsługa interakcji w czasie rzeczywistym.
- Apache Hadoop
- Apache Kafka

Znode

- Hierarchiczna struktura danych – składająca się z tzw. znode'ów (węzłów) powiązanych ze sobą w formie drzewa. Każdy z węzłów może przechowywać w sobie dane oraz posiadać potomków (inne znode'y). Struktura podobna do systemu plików ale znode może być naraz plikiem i katalogiem.



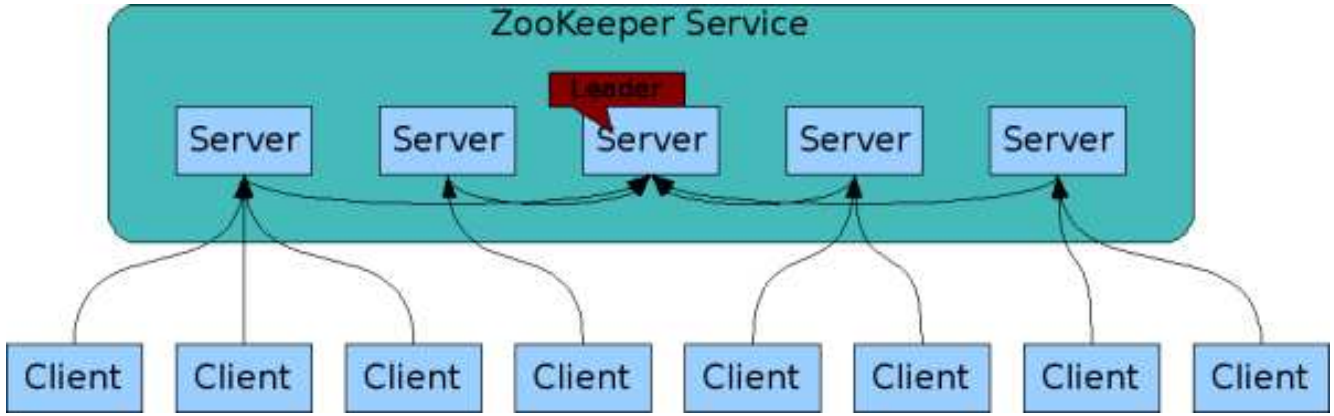
- Efemeryczne znode'y – znode'y które istnieją tylko na czas sesji klienta. W momencie rozłączenia klienta taki znode'y zostaje automatycznie usunięty.
- Powiadomienia (watchery) – pozwalają uzyskać informacje (PUSH) o zmianie danych w znodzie lub o dodaniu znode'a (potomka) do znode'a (rodzica).

Lider

- Lider – w klastrze w tym samym czasie tylko jeden serwer może pełnić role lidera, pozostałe serwery działają jako tzw. followers. Jeśli nie ma lidera lub aktualny lider przestaje odpowiadać, pozostałe serwery próbują wybrać nowego lidera. Do wyboru nowego serwera konieczne jest kworum (quorum). Lider odpowiada za wszystkie zapisy w obrębie klastra (followers przekierowują do niego zapisy) oraz o zachowanie kolejności operacji.

Quorum

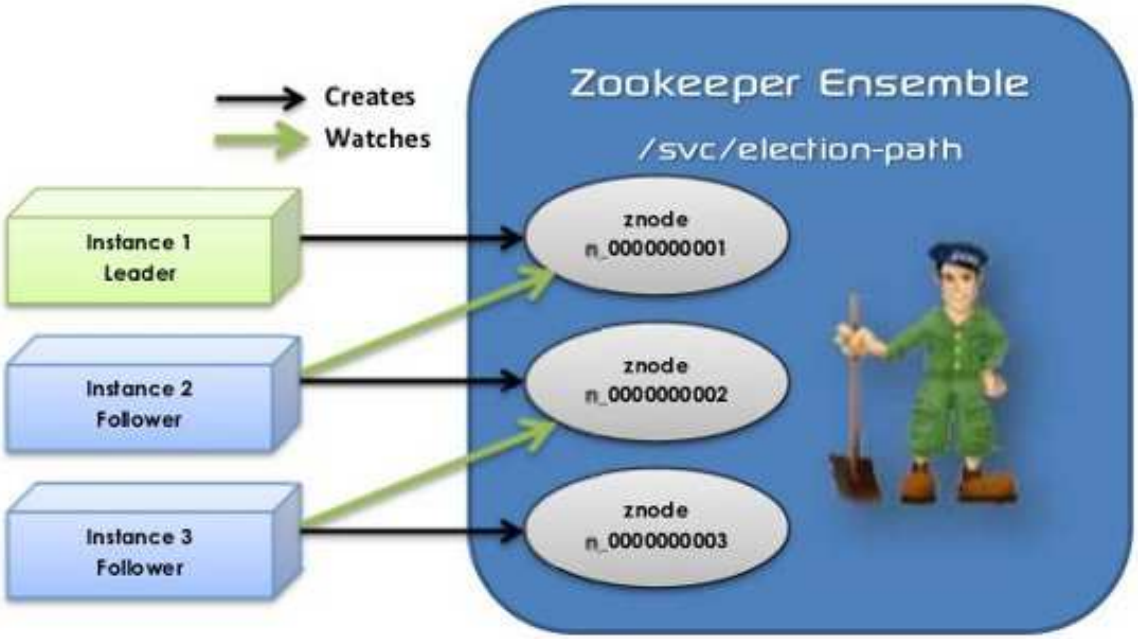
- Quorum – wybór lidera w klastrze odbywa się poprzez Quorum gdzie n/2 + 1 serwerów musi wspólnie wybrać nowego lidera. Nie jest zalecane żeby klaster składał się z parzystej liczby serwerów (nie zwiększa to odporności systemu na awarię).



http://zookeeper.apache.org/doc

Wybór lidera

- Wszystkie serwery Zookeepera tworzą efemerycznego-sekwencyjnego znode w tej samej ścieżce (np. /svc/election-path). Znode z najmniejszym numerem sekwencyjnym zostaje liderem.



https://www.outbrain.com/techblog/2011/07/leader-election-with-zookeeper/

Gwarancja spójności

- Spójność sekwencyjna – wszystkie aktualizacje zapisów klientów są dokonywane sekwencyjnie,
- Atomowość – aktualizacja albo się powiedzie, albo nie (brak stanów pośrednich),
- Pojedynczy widok na system, klient widzi ten sam serwis Apache Zookepera niezależnie do którego serwera się podłączy,
- Trwałość – jeżeli uaktualnienie się powiedzie, jest ono trwale zapisane,
- Apache Zookeeper nie gwarantuje że w każdym momencie wszyscy klienci mają ten sam widok systemu.

Zalety

- Proste użycie:
  - proste API,
  - hierarchiczna struktura danych.
- Replikacja danych,
- Zachowanie kolejności operacji,
- Szybkość działania,
- Powiadomienia.

Przykład

- Aplikacje powinny być projektowane w odpowiedni sposób:
  - muszą działać asynchronicznie (asynchroniczne powiadomienia),
  - muszą umożliwiać przeładowanie konfiguracji w locie, bez konieczności restartu.
- Przykładowa aplikacja (usługa) działająca na wielu serwerach potrzebuje połączyć się z **bazą danych**.
- Aplikacja **podczas startu łączy się z ZooKeeperem** i pobiera z ustalonego **znode** swoją konfigurację.

Przykład

- W pobranej konfiguracji znajdują się informacje potrzebne do połączenia z bazą danych (adres serwera gdzie jest baza, użytkownik, hasło itp.)
- **Aplikacja zaczyna także nasłuchiwać na dalsze zmiany tego znode**.
- W sytuacji kiedy przykładowo używana baza danych jest przeniesiona na inny serwer wystarczy uaktualnić znode gdzie jest przechowywana konfiguracja bazy.
- Do przykładowej usługi (zainstalowanej na wielu serwerach) **Apache ZooKeeper wyśle powiadomienie** o zmianie danych w znode gdzie jest przechowywana konfiguracja.

Przykład

- Aplikacja (usługa) obserwuje zmianę konfiguracji bazy, sprawdza co się zmieniło w konfiguracji i odpowiednio reaguje.
- W sytuacji kiedy konieczne jest połączenie się z bazą na innym hoście, aplikacja kończy przetwarzać obecne żądania. Następnie, kolejkuje przychodzące żądania, łączy się do nowej bazy danych oraz wznawia działanie.
- Jeśli nie jest możliwe połączenie z nową bazą to łączy się do starej bazy danych.
- **W podanym przykładzie zapewnione zostało ciągłe działanie aplikacji (usługi) w momencie migracji bazy danych.**