Архитектурни стилове на софтуерни системи (част 2) -Резюме

Architectural Styles in Cloud (Архитектурни стилове в облака)

Circuit Breaker (Циркуит Брейкър)

Неуспех на услуга в разпределена среда - Класическо решение е да се имплементира timeout за други услуги, които я извикват. Това обаче може да доведе до ненужно използване на ресурси в облачна среда. Представете си стотици потребители, чакащи за отказала услуга, всеки с изчакване на timeout.

Circuit Breaker Pattern (Патърн „Циркуит Брейкър"):

- Предотвратява приложението да извършва операции, които е **малко вероятно да успеят**. Например, извикване на услуга, която вече е известна като неуспешна.
- Действа като **proxy** за операции, които могат да се провалят.
- Наблюдава **броя на последните неуспехи** и решава дали да позволи операцията или да върне изключение веднага.

Предимства:

- Повишава надеждността на системата.
- Приложенията **няма да чакат излишно за timeout**, когато услугата е известна като неработеща.
- Полезен при **временни прекъсвания на мрежата** или когато услугата е твърде **натоварена**.

Sharding Style (Стил „Шардинг") - Целта е увеличаване на мащабируемостта при работа с големи обеми от данни. Идеята е да се раздели хранилището на данни на хоризонтални партиции, наречени **shards**.

Мотивация за използване: Увеличаване на капацитета за съхранение; Повишаване на изчислителните ресурси; Оптимизация

на мрежовата пропускателност; Географско разпределение на данните.

Ключови фактори:

- Важно е как ще бъдат разделени данните между shards.
- **Shard key (ключ за разделяне)** определя как се разпределят данните. Ключът трябва да е статичен и да не зависи от данни, които може да се променят.

Имплементация: Логиката за sharding може да се реализира в приложението или да бъде част от самата база данни.

Стратегии за имплементация:

1. **Lookup strategy** (Стратегия за търсене)
2. **Range strategy** (Стратегия с диапазони)
3. **Hash strategy** (Хешираща стратегия)

Предимства: По-добро управление на данните; Повишена производителност на хранилището.

Недостатъци:

- Сложност при определяне на местоположението на данните.
- Повече режийни разходи, когато данни от една заявка са разпръснати в много shards.
- Неравномерно разпределение на данните.
- Трудности при проектиране на универсален shard key.

Queue (Опашка)

Проблем: В облака услуги могат да бъдат претоварени от множество едновременни заявки. **Решение**: Използване на queue за балансиране на натоварването и избягване на отказ на услуги.

Видове опашки:

- **Standard Queue (Стандартна опашка) -** Действа като буфер за съхранение на съобщенията, докато услугата ги обработи. Минимизира риска от недостъпност при голям брой заявки.
- **Priority Queue (Приоритетна опашка) -** Сортира заявките според приоритет. Приоритетът може да се зададе от приложението или автоматично от опашката.
- **Fixed Length Queue (Опашка с фиксирана дължина) -** Предотвратява Denial of Service (DoS) атаки чрез ограничаване на броя на съобщенията. Когато лимитът бъде достигнат, се връща изключение вместо изчакване на timeout.

Cache (Кеш) - Използва се за оптимизиране на многократния достъп до често използвани данни.

Процес:

- **Четене:**
  1. Проверка в кеша.
  2. Ако липсва, данните се извличат от хранилището и се съхраняват в кеша.
- **Запис:**

  1. Промяна на данните в хранилището.
  2. Инвалидиране на съответния елемент в кеша.

Service Oriented Architectures - Сервизно-ориентирани архитектури

Software Services (Софтуерни услуги) - софтуерен компонент, който може да бъде достъпен от отдалечени компютри през интернет. При подаден вход, услугата генерира съответен изход без странични ефекти.

-Услугата се достъпва чрез нейния публикуван **interface (интерфейс)** и всички детайли на имплементацията ѝ са скрити.

-Услугите не поддържат вътрешно състояние. **State information (информация за състоянието)** се съхранява в база данни или се управлява от заявителя на услугата.

-При заявка към услуга, информацията за състоянието може да бъде включена като част от заявката, а актуализираното състояние се връща като част от резултата.

-Поради липсата на локално състояние, услугите могат динамично да се преразпределят между виртуални сървъри и да се репликират на няколко сървъра.

Web Services (Уеб услуги)

След различни експерименти през 90-те години със **service-oriented computing (сервисно-ориентирано изчисление)**, концепцията за **"big" Web Services ("големи" уеб услуги)** се появява в началото на 2000-те.Те се базират на протоколи и стандарти, основаващи се на **XML**, като: **SOAP (Simple Object Access Protocol)** за взаимодействие между услуги; **WSDL (Web Services Description Language)** за описание на интерфейса

Повечето софтуерни услуги не изискват сложността, присъща на уеб протоколите, затова съвременните системи използват по-прости и ефективни методи с по-малък **overhead (режийни разходи)**. Съвременните сервизно-ориентирани системи използват по-прости, „леко тежки“ (lightweight) протоколи за взаимодействие, които имат по-ниски разходи и по-бързо изпълнение.

Описание на уеб услуга (Web Service Description Language - WSDL) - Интерфейсът на услугата се дефинира в описание на услугата, изразено в WSDL.

WSDL спецификацията определя:

- Какви операции поддържа услугата и формата на съобщенията, които се изпращат и получават.

- Как се достъпва услугата - свързването (binding) картографира абстрактния интерфейс към конкретен набор от протоколи.
- Къде се намира услугата, обикновено изразено като URI (Universal Resource Identifier).

Микроуслуги (Microservices) -малки, безсъстояние (stateless) услуги с една отговорност. Те се комбинират за създаване на приложения.

• Те са напълно независими със собствена база данни и код за управление на потребителския интерфейс (UI).

• Софтуерните продукти, използващи микроуслуги, имат архитектура от тип микроуслуги (microservices architecture).

• Ако е необходимо създаване на софтуерни продукти за облака, които са адаптивни, мащабируеми и устойчиви, препоръчително е да бъдат проектирани около архитектура от микроуслуги.

RESTful уеб услуги (RESTful Web Services) - Уеб услугите, базирани на XML стандарти, са критикувани като „тежки“- “over-general” и неефективни. **REST (REpresentational State Transfer)** е архитектурен стил, базиран на трансфер на представяния на ресурси от сървър към клиент. Този стил е в основата на уеб като цяло и е по-прост за реализиране на уеб услуги в сравнение със SOAP/WSDL.

• RESTful услугите изискват по-ниски разходи и се използват от много организации, които изграждат системи, базирани на услуги.

Ресурси (Resources) - Основният елемент в RESTful архитектурата е **ресурсът**- елемент от данни като каталог, медицински запис или документ. Ресурсите могат да имат множество представяния: MS Word, PDF, Quark Xpress. Операции върху ресурси: **Create, Read, Update, Delete**

HTTP REST URL: https://api.example.com/users?id=123

Протокол (Protocol) - http:// или https://

- Определя транспортната схема, която ще се използва за изпращане и получаване на заявката.
- HTTP (HyperText Transfer Protocol) е най-често използваният.
- HTTPS е защитена версия, която криптира данните.

Име на хост (Host Name) - api.example.com

- Идентифицира сървъра, на който се намира ресурсът.
- Това е домейн или IP адрес, който посочва точното местоположение на ресурса в интернет.

Път (Path) /users

- Определя конкретния ресурс на сървъра, който искаме да достъпим.
- Структурата на пътя може да показва йерархия от ресурси.

Стринг на заявка (Query String) ?id=123

- Използва се за филтриране или персонализиране на заявката.
- Започва със символа ? и съдържа двойки ключ-стойност (**ключ=стойност**), разделени със **&** при повече параметри.

Протокол: https – методът за комуникация; Хост: api.example.com – сървърът, където се намира ресурсът; Път: /users – кой ресурс искаме; Стринг на заявка: ?id=123 – параметър, който уточнява кой точно потребител искаме (с ID = 123)

Недостатъци на RESTful подхода: При сложен интерфейс може да е трудно да се проектира набор от RESTful услуги; Липсват стандарти за описание на RESTful интерфейси, което налага използването на неофициална документация за разбиране на интерфейса.