**Միկրոսերվիզային ճարտարապետությունում ծառայությունների միջև կապերի կազմակերպման մեթոդներ**

**Ներածություն**

Միկրոսերվիզային ճարտարապետությունը ներկայումս լայնորեն օգտագործվող մոտեցում է ծրագրային համակարգերի նախագծման մեջ։ Այն ներառում է ծրագրային համակարգի բաղադրիչների (ծառայությունների) փոքր միավորների բաժանում՝ յուրաքանչյուրը ինքնուրույն և անկախ գործող, ինչը ապահովում է ավելի լավ մասշտաբելիություն, կայունություն և հեշտ վերականգնողականություն։ Միկրոսերվիզները հաճախ աշխատում են որպես անկախ ծառայություններ, որոնք կարող են տեղակայվել տարբեր սերվերներում կամ ամպային միջավայրերում։ Այսպիսի ճարտարապետության հիմնական խնդիրներից է **ծառայությունների միջև կապերի արդյունավետ և անվտանգ կազմակերպումը**։ Այս փաստաթղթում ներկայացվում են միկրոսերվիզային ճարտարապետությունում ծառայությունների միջև կապերի հիմնական մեթոդները, օրինակներով և առավելություններով։

**Ծառայությունների միջև կապերի տեսակները**

Ծառայությունների միջև կապերը կարելի է բաժանել երկու հիմնական կատեգորիայով՝ **սինխրոն** և **ասինխրոն**։

**1. Սինխրոն կապ**

Սինխրոն կապի դեպքում ծառայությունները փոխանակում են տվյալները իրական ժամանակում։ Մի ծառայություն ուղարկում է հարցում և սպասում է պատասխան ստանալու, նախքան շարունակել իր գործողությունները։

**Օրինակներ**

* **HTTP/REST API**  
  Մեկ ծառայություն ուղարկում է HTTP GET կամ POST հարցում մեկ այլ ծառայությանը և սպասում է JSON կամ XML պատասխանի։  
  Օրինակ՝ **OrderService**-ը դիմում է **ProductService**-ին՝ ստանալու ապրանքի տվյալները։
* **gRPC**  
  Գործողություններ ավելի արդյունավետ են իրականացնում փոքրածավալ բինար տվյալների փոխանակումը, արագացնում է կապը և ապահովում է ուժեղ տիպերի ստուգում։

**Առավելություններ**

* Պարզ և ինտուիտիվ՝ հասկանալու համար։
* Լավ է դեպքերում, երբ պատասխան ստանալը անհրաժեշտ է անմիջապես։

**Նրբություններ**

* Ծառայություններ կարող են «փակվել» կամ կախված մնալ մեկը մյուսից, եթե սպասումը երկար է կամ ծառայությունը անհասանելի է։
* Ծրագրավորման և մասշտաբելիության դժվարություններ.

**2. Ասինխրոն կապ**

Ասինխրոն կապը հնարավորություն է տալիս ծառայություններին փոխանակել տվյալներ առանց անմիջական սպասման պատասխանին։ Այս մեթոդը հաճախ օգտագործվում է հաղորդագրությունների հերթերով (message queues)։

**Օրինակներ**

* **RabbitMQ**, **Azure Service Bus**, **Amazon SQS**
  + **Producer** ծառայությունը ուղարկում է հաղորդագրություն հերթի մեջ։
  + **Consumer** ծառայությունը վերցնում է հաղորդագրությունը և մշակում այն իր տեմպով։
* **Kafka**
  + Publisher/Subscriber մոդել, երբ բազմաթիվ սպառողներ կարող են ընդունել նույն հաղորդագրությունը։

**Առավելություններ**

* Մեծ մասշտաբելիություն՝ քանի որ ծառայությունները չեն կախված անմիջական պատասխանից։
* Հուսալիություն՝ հաղորդագրությունները չեն կորում, նույնիսկ եթե սպառողը զբաղված է կամ հանկարծակի փակվում է։

**Նրբություններ**

* Ավելացնում է հավելյալ ինֆրաստրուկտուրա (message broker):
* Պետք է մշակել հաղորդագրությունների կարգավորման մեխանիզմներ (ordering, retries, deduplication)։

**3. Event-Driven Architecture (EDA)**

**Միջոցառում-դրիվեն ճարտարապետություն**՝ ծառայությունները փոխանակում են **իրադարձություններ**։ Մի ծառայություն հրապարակում է իր կատարված գործողությունը որպես իրադարձություն, և մյուս ծառայությունները կարող են արձագանքել դրան։

**Օրինակ**

* **OrderCreatedEvent**
  + **OrderService**-ը ստեղծում է նոր պատվեր և հրապարակում OrderCreatedEvent։
  + **PaymentService**-ը կամ **NotificationService**-ը լսում է այդ իրադարձությունը և իրականացնում համապատասխան գործողություն (վճար ստացում, նամակ ուղարկում)։

**Առավելություններ**

* Մեծ մասշտաբելիություն և անջատվածություն (decoupling)
* Պարզեցնում է բազմապլատֆորմ զարգացումը
* Հարմար է իրական ժամանակի տվյալների հոսքերի համար

**Նրբություններ**

* Խնդիրներ կառավարման, հուսալիության և իրադարձությունների հերթագրման հետ
* Ապահովում է դժվար, եթե անհրաժեշտ է համոզվել, որ իրադարձությունը հաջողությամբ մշակվել է