

Микросервисни обрасци у изради

Проф. др Игор Дејановић (igord at uns.ac.rs)

Креирано 2023-03-05 Sun 15:15, притисни ESC за мапу, "м" за мени, Ctrl+Shift+F за претрагу

Садржај

1. Увод
2. Обрасци за рад са базама података
3. Обезбеђивање конзистенције
4. Постављање упита
5. Комуникација
6. Откривање сервиса
7. Литература

УВОД

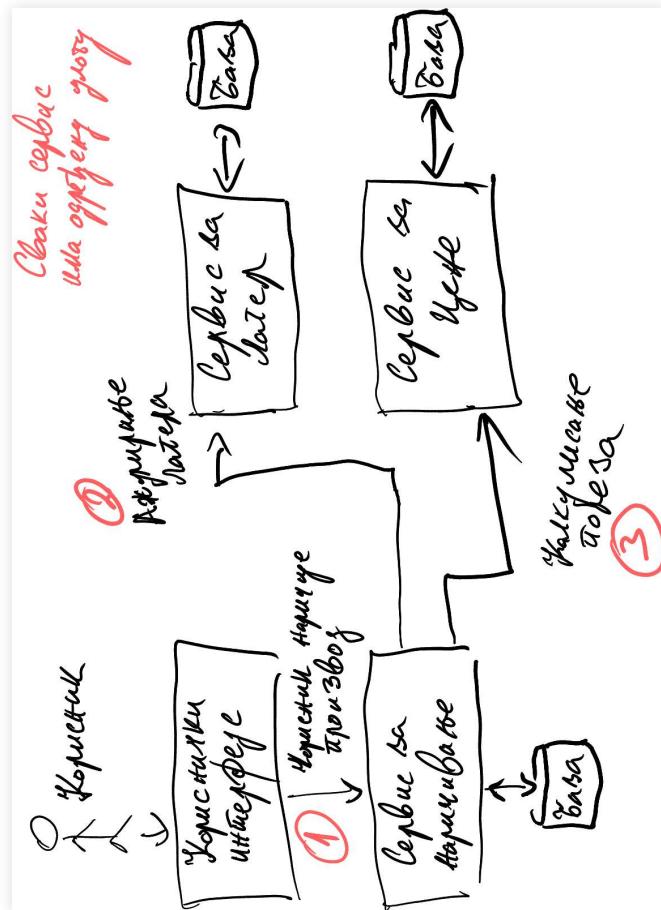
Микросрвјиси (*Microservices*)

- Софтверске компоненте.
- Висока кохезија, слаба спрега.
- Комуникација се обавља путем стандардних технолошки независних протокола (најчешће НТТР).
- Могу бити имплементирани у различитим програмских језицима и технологијама.
- Мали су, ограниченог контекста. Независно се развијају и уводе.
- Имају јасно дефинисане интерфејсе путем којих комуницирају.
- *Unix* филозофија:

Do one thing and do it well.

Архитектура базирана на микросервисима

- Архитектонски стил где се апликација гради као скуп слабо спрегнутих "малих" сервиса (микросервиса) који сарађују.
- Варијанта **Service-Oriented Architecture** (SOA) или сервиси су "мали" и протоколи за комуникацију су једноставни (*light-weight*).
- Микросервиси пружају услуге и/или користе друге микросервисе.
- У циљу независне миграције микросистема на нове верзије интерфејси се верзионирају и омогућава се клијентима да користе старе интерфејсе у прелазном периоду.



Предности у односу на монолитну архитектуру

- Тимови могу бити технолошки хетерогени.
- Увођење (*deploy*) се обавља у малим инкрементима (*fine-grained*).
- Модуларност, декомпозиција. Лакше разумевање, развој и тестирање. Отпорност на "ерозију архитектуре".
- Больја скалабилност. Больја еластичност. Лако додавање нових микросрвиса по потреби.
- Больја отпорност на отказе. Уколико један микросрвис "падне" остатак апликације наставља да ради.
- Лакша миграција на нове технологије. Могућа постепеним заменама микросрвиса.
- Интеграција хетерогених и "старих" система (*legacy*).
- Континуална интеграција и достава (*Continuous Integration/Delivery*)

Мане у односу на монолитну архитектуру

- Више "покретних делова". Захтева боље алате за увођење и надзор.
- Теже дебаговање. Дебаговање захтева праћење захтева кроз више микросервиса који се извршавају често на различитим физичким/виртуелним рачунарима.
- Додатни трошкови (*overheads*) услед комуникације.
- Додатни трошкови у случају потребе за додењем података.

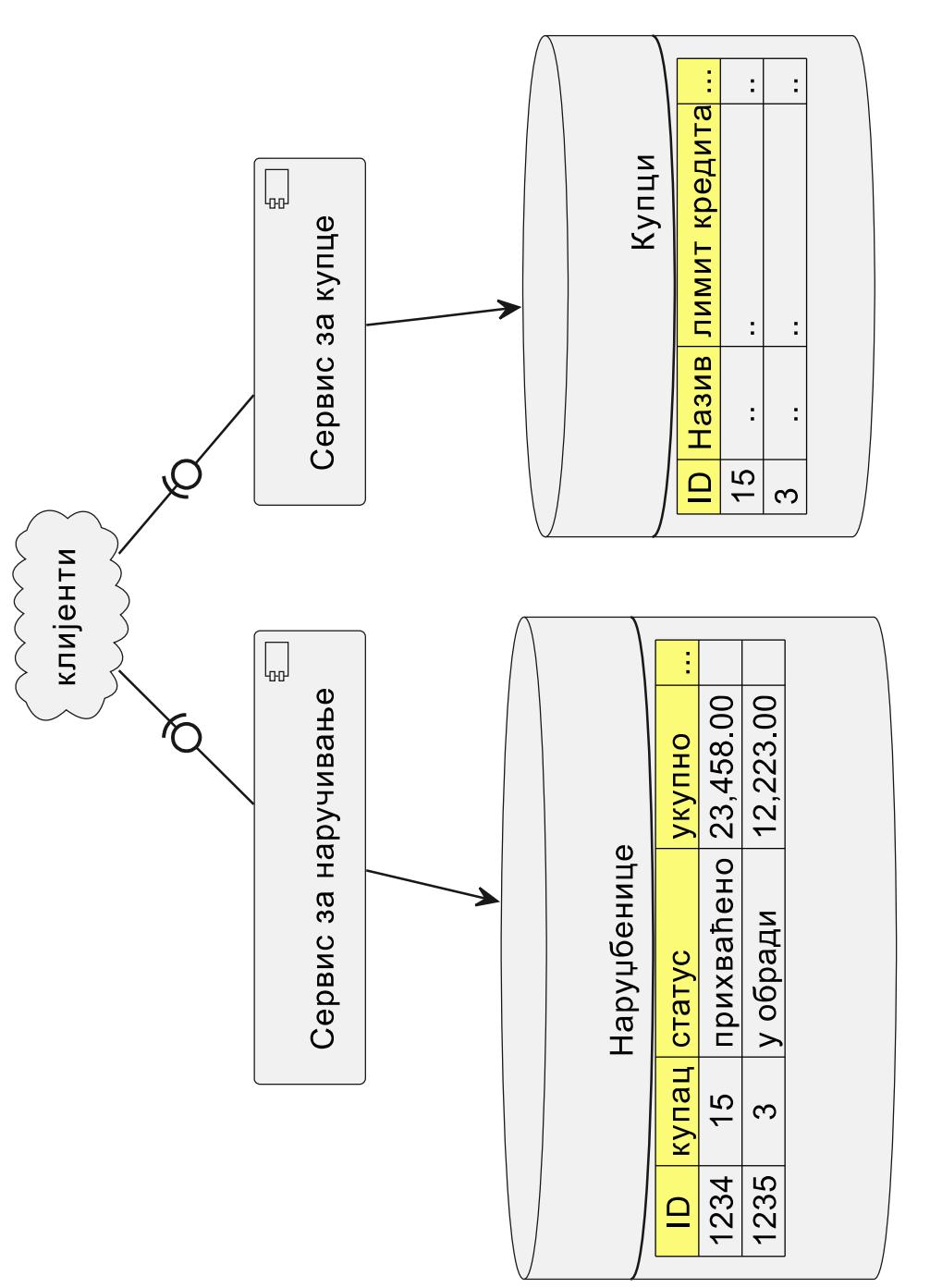
Обрасци за ради са базама података

База података по сервису (*Database per service*)

- У циљу слабог спрезанћа сервиса подаци над којима сервиси оперишу се имплементирају као приватни.
- Други сервиси не могу приступити подацима директно већ само кроз интерфејс сервиса.

<https://microservices.io/patterns/data/database-per-service.html>

Структура



Предности

- Подаци су део имплементације сервиса.
- Имплементација приватне базе се може менјати независно од остатка система.
- Могуће је користити хетерогене технологије.

Мане

- Отежано извођење трансакција које се обављају између података различитих сервиса. Видети образац [Saga](#).
- Отежани сложени упити који обухватају податке више сервиса. Видети образац [CQRS](#).

Начини имплементације

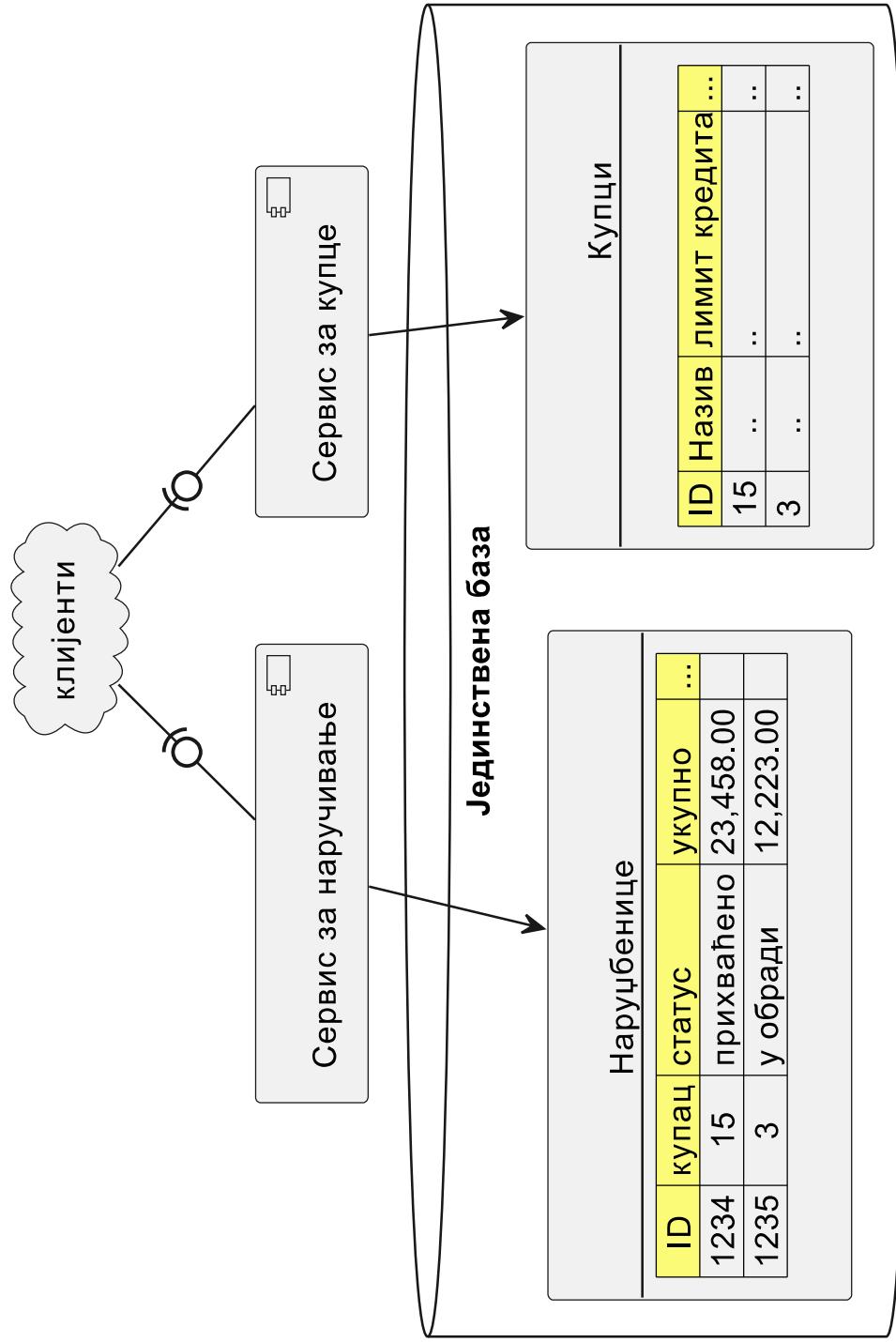
- Употреба једне инстанце базе за све сервисе:
 - приватне табеле по сервису,
 - приватна шема по сервису.
- Посебна инстанца базе по сервису.

Дељена база података (*Shared database*)

- У циљу подршке за ACID трансакције сервиси деле исту базу података и могу слободно да приступају подацима других сервиса.

<https://microservices.io/patterns/data/shared-database.html>

Структура



Предности

- Једноставније за имплементацију и операцију.
- Једноставније трансакције (ACID) и упити (нпр. могуће JOIN између табела различитих сервиса).

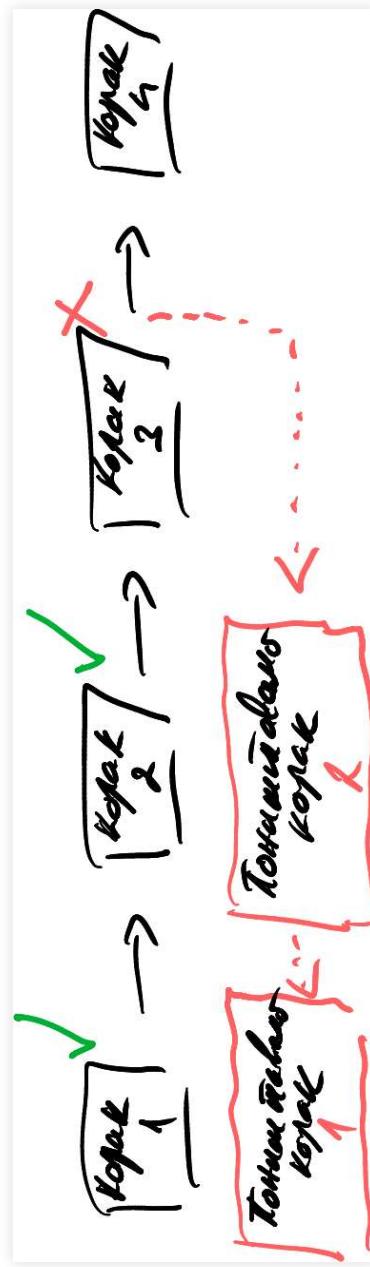
Мане

- Јача спрега између сервиса у време развоја (нпр. одржавање шеме базе мора бити кооринисано између тимова) и у време извршавања (нпр. један сервис може да закључча табелу и спречи друге сервисе да приступе).
- Сервиси могу да мењају податке других сервиса. Ово је могуће административно регулисати уколико база подржава.
- Иста база можда неће задовољити потребне функционалне и нефункционалне особине захтеване од стране неких сервиса.

Обезбеђивање конзистенције

Saga

- Вид дистрибуиране трансакције. Мање ригидна од *two-phase commit* (2PC).
- Очување конзистенције података и змеју сервиса.
- Користи се када је употреби Database per service образац за имплементацију трансакција.
- Низ локалних трансакција (*ACID*) које објавом поруке/догађаја иницирају следећу трансакцију у ланцу. Уколико нека од трансакција не успе, извршава се поништавање.

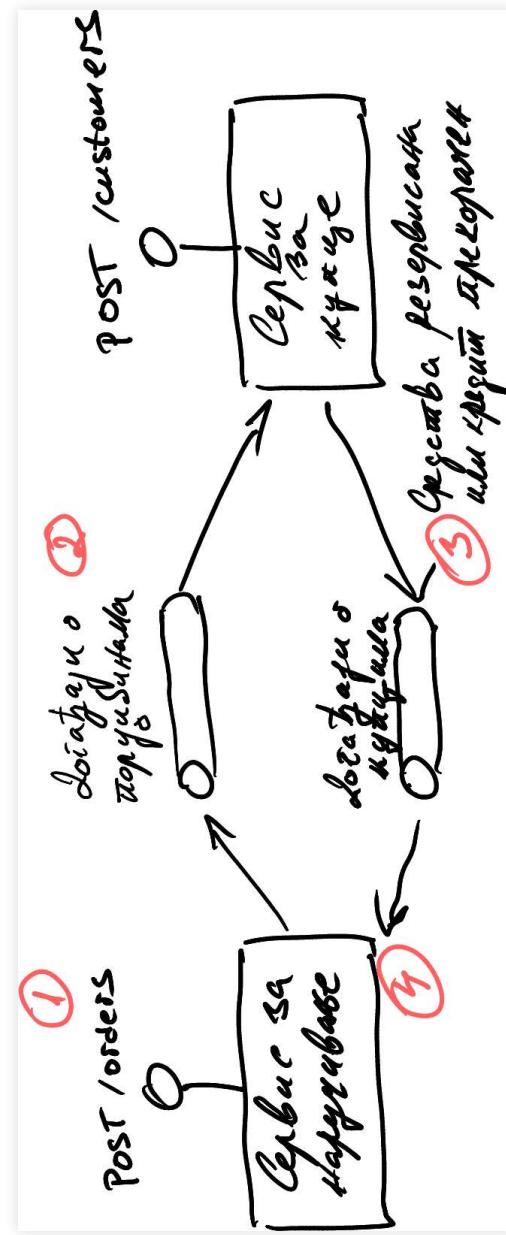


Приступи

- Два приступа у координацији трансакције:
 - Базиран на кореографији (*Choreography*) - после сваке локалне трансакције објављује се догађај који иницира извршавање следеће трансакције у низу.
 - Базиран на оркестрацији (*Orchestration*) - оркестратор (посебан објекат) је задужен да обавести учеснике да започну или да пониште трансакцију.

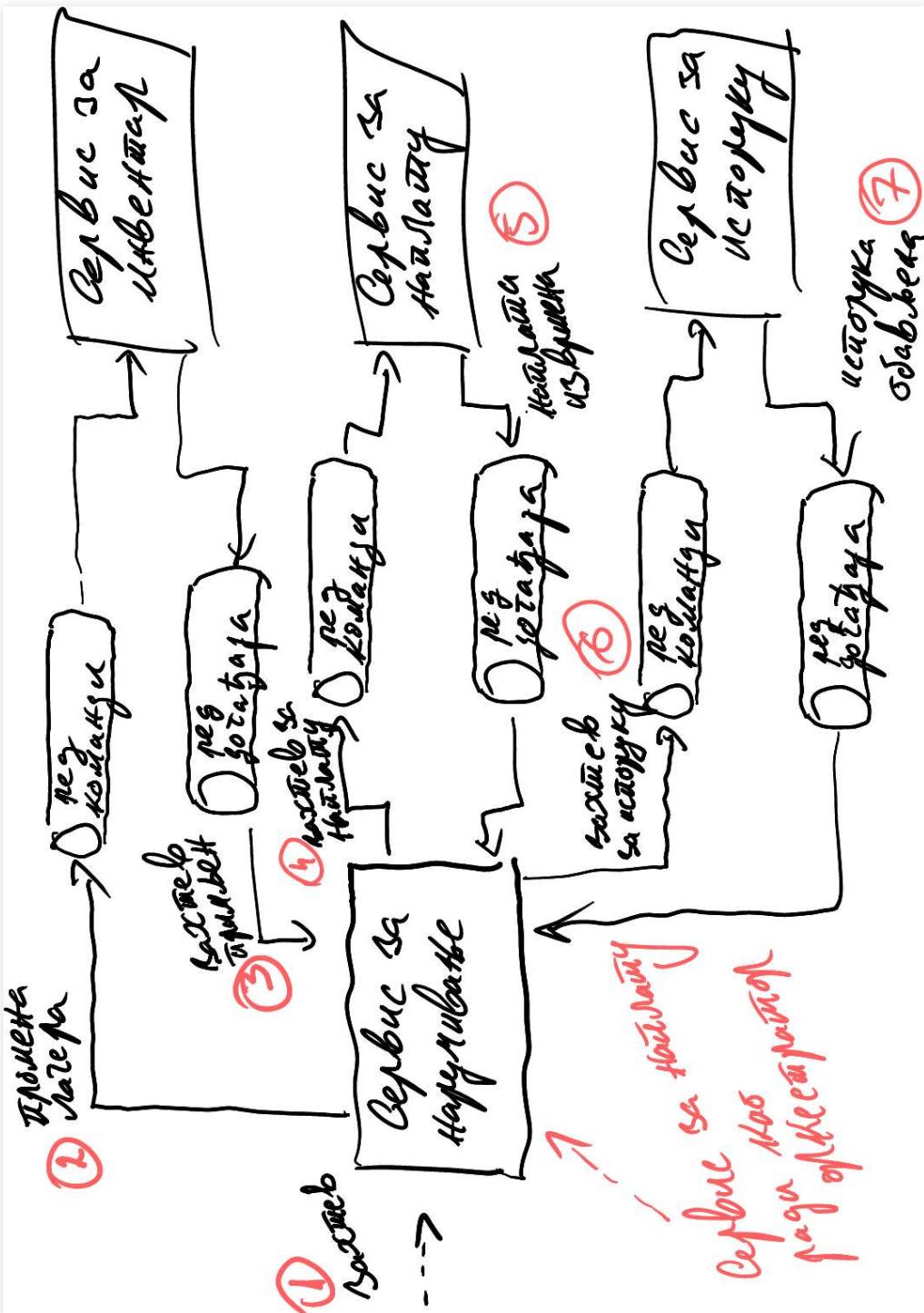
Структура - Кореографија

1. Сервис за наручињање прима захтев **POST /orders** Креира наруџбину у стању **у обради**
2. Прослеђује догађај **Наруџбина кремрана**
3. Сервис за купце покушава да резервише средства и објављује догађај који представља резултат операције: средства су резервисана или је кредит прекорачен.
4. Сервис за наручињање приhvата или одбија наруџбину.



Структура - оркестрација

- Сервис за наручивање је оркестратор тј. задужен је за координацију целокупног процеса наручивања.



Предности

- Слабија спрега. Мање ригидан систем.
- Больја скалабилност.

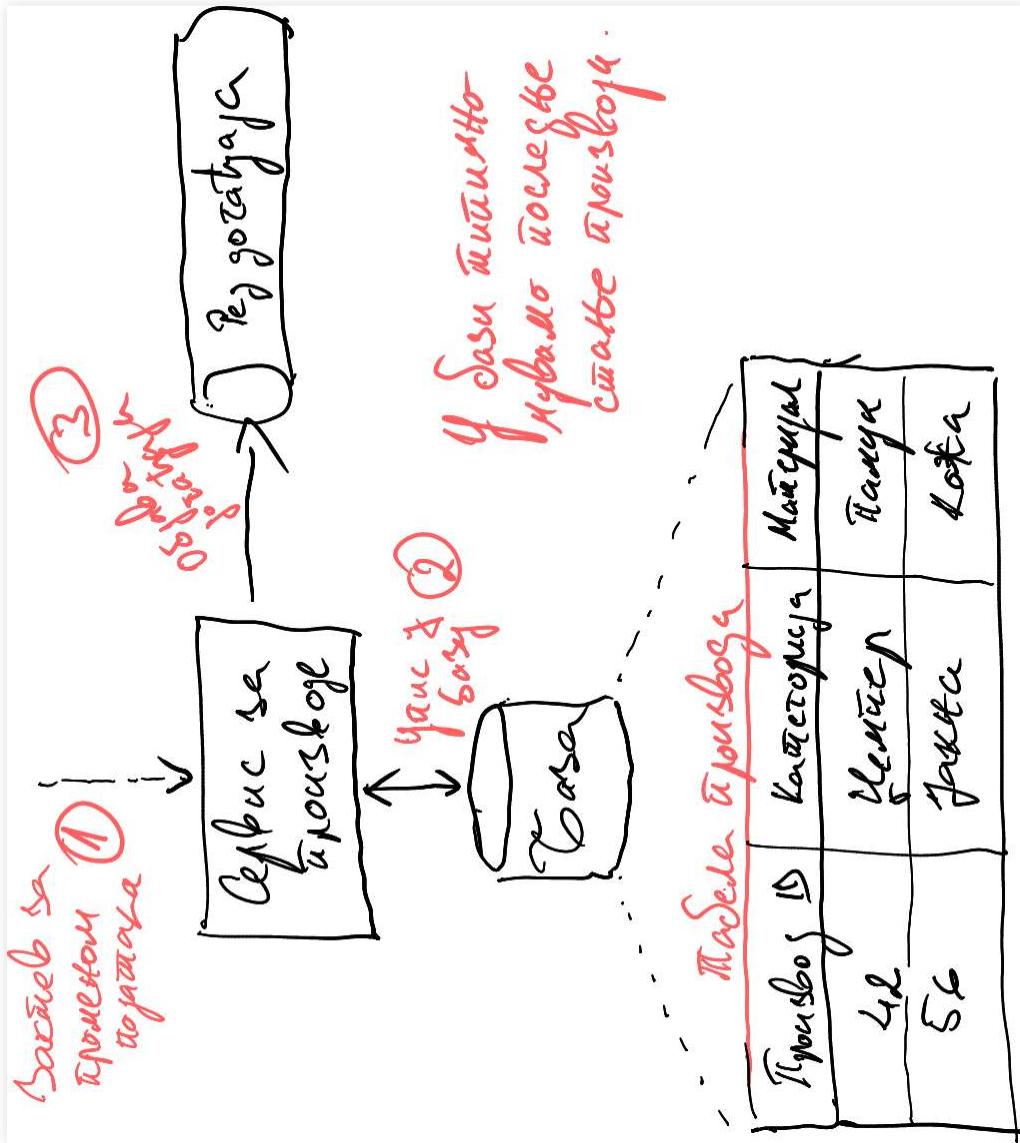
Мане

- Мора се пажљиво моделовати редослед операција због потенцијалног поништавања.
- Шта радити у ситуацији када је операција неуспешна због техничког проблема (нпр. сервис није тренутно доступан)?
- Коначна конзистентност (*Eventual Consistency*).

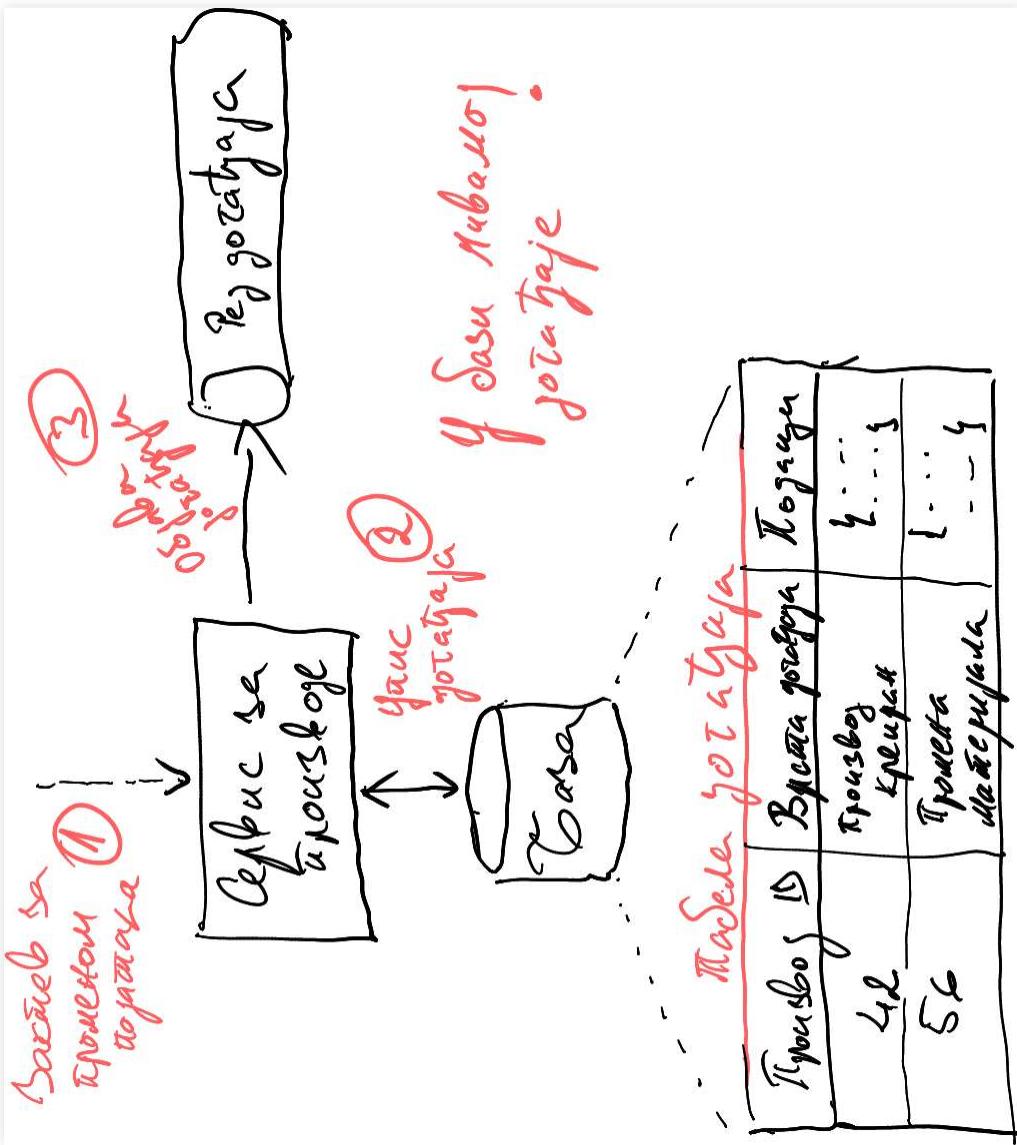
Event sourcing (ES)

- Користи се код архитектуре вођене догађајима (*Event-Driven*).
- Уместо чувања тренутног стања ентитета чува се низ догађаја који су менјали ентитет.
- Текуће стање се може добити применом свих догађаја до садашњег тренутка.
- Природно се користи са обрасцем CQRS.

Структура код классичног приступа



Структура код ES приступа



Напомене

- Ток догађаја треба да буде једини извор текућег стања.
- Предности:
 - различити модели се могу изградити применом тока догађаја у будућности.
 - природно садржи пуну историју измена што омогућава ревизију и контролу.
- Мана: немогућност постављања упита над током догађаја - због тога се користи у синергији са CQRS.
 - Додатно можемо користити брокере порука (*message brokers*) уместо базе података.

Постављање упита

API композиција (*API Composition*)

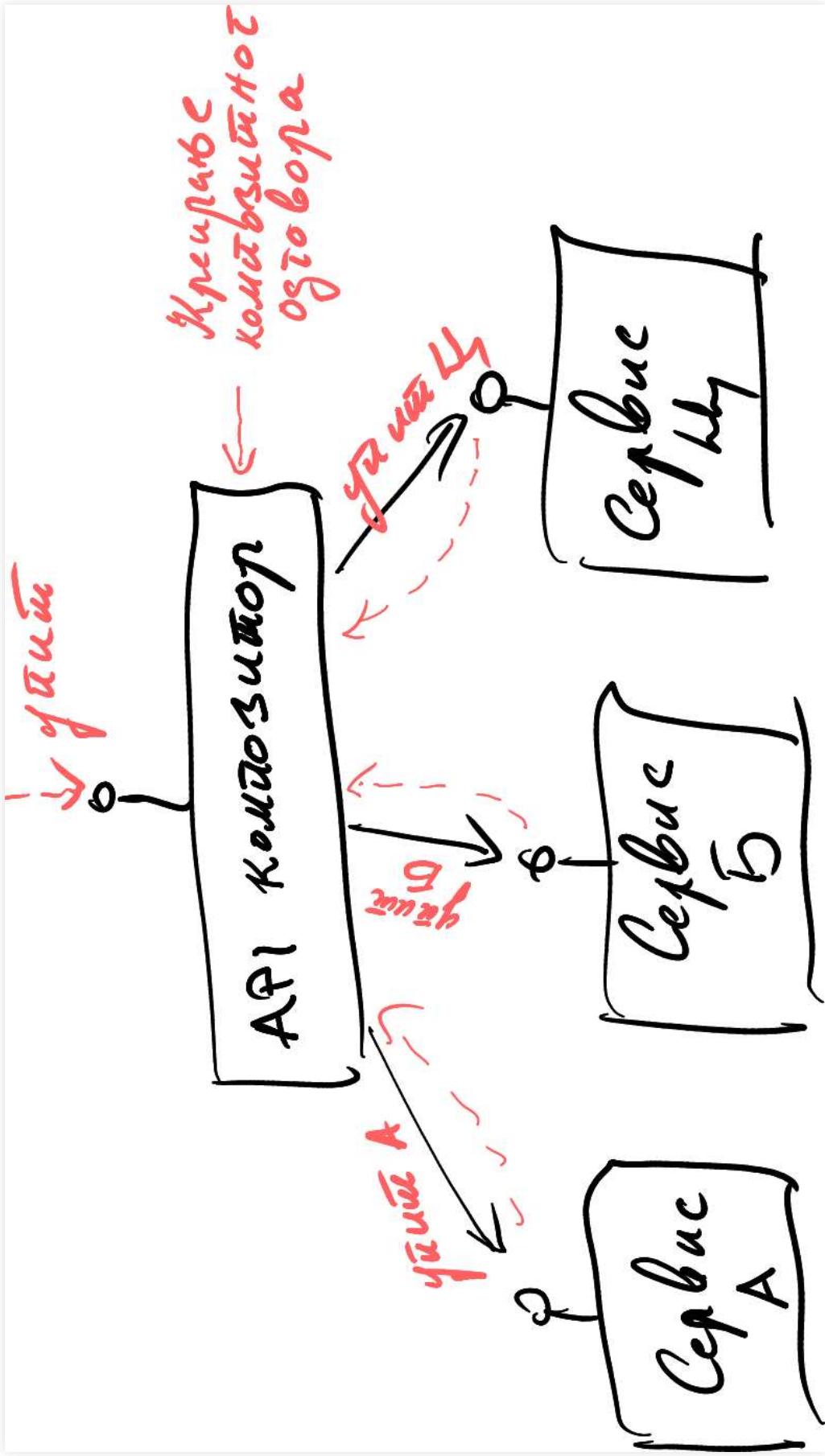
- У случају употребе обрасца *Database per service* поставља се питање како имплементирати упите који удржкују податке из различитих микросрвиса.

<https://microservices.io/patterns/data/api-composition.html>

Решење

- Креирати сервис који поставља појединачне упите и затим комбинује податке у меморији и враћа јединствени одговор са удруженим подацима.

Структура



Предности и Мане

- Предности:
 - Поједностављење сложених упита.
 - Једно место за ажурирање сложених упита.
- Мане:
 - Поједини упити могу бити неефикасни јер се велика количина података преноси преко мреже и удржује у меморији.

Пример

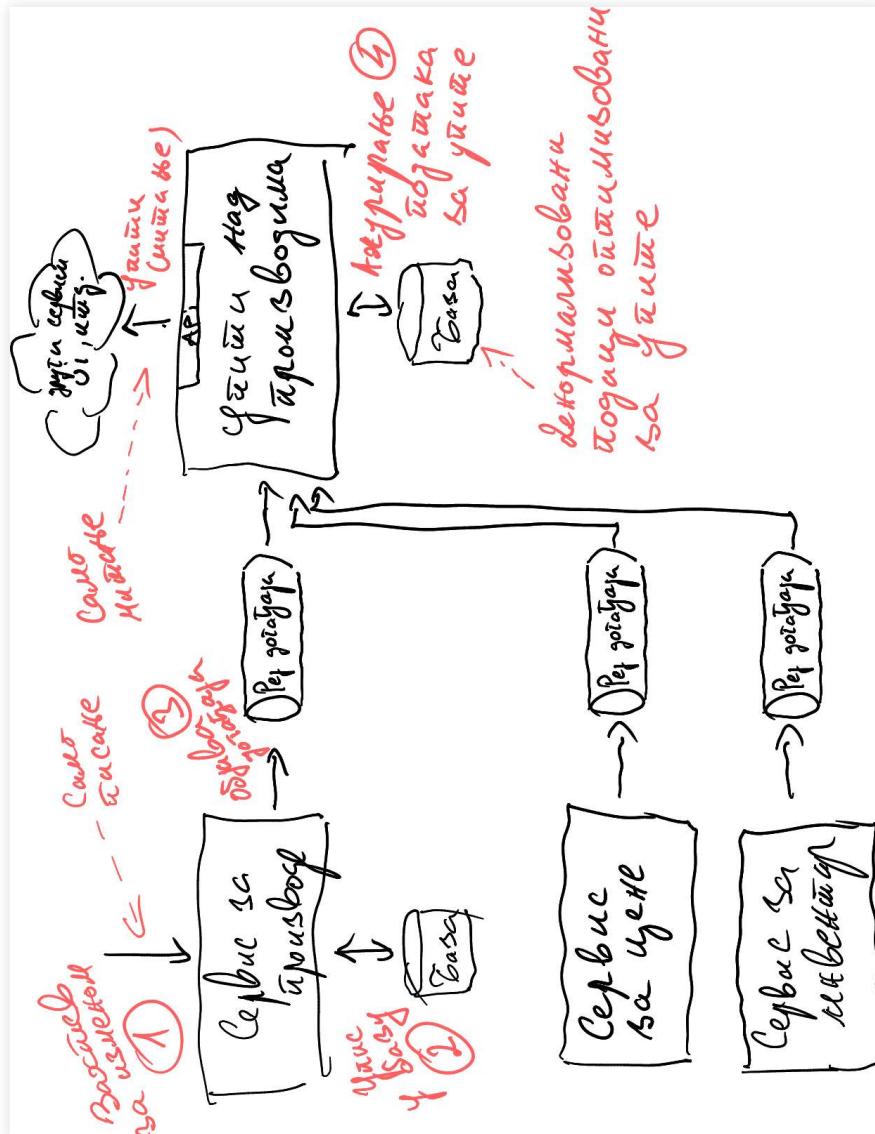
- Образац *API Gateway* често ради API композицију.

CQRS

- *Command Query Responsibility Segregation (CQRS)* се користи у ситуацији када имамо обрасце *Database per service* и *Event sourcing* имплементиране и желимо да подржимо упите који удрежују податке из више микросервиса.
- Базиран на идеји поделе захтева на оне који мењају стања и оне који само читају тј. немају бочне ефекте. Еквивалентно са *REST* методама за читање (`GET`, `HEAD`) и измену стања (`POST`, `PUT`, `PATCH`...).
- Креирање базе која је само за читање и која се континуално ажурира обрадом догађаја који се емитују при промени података.

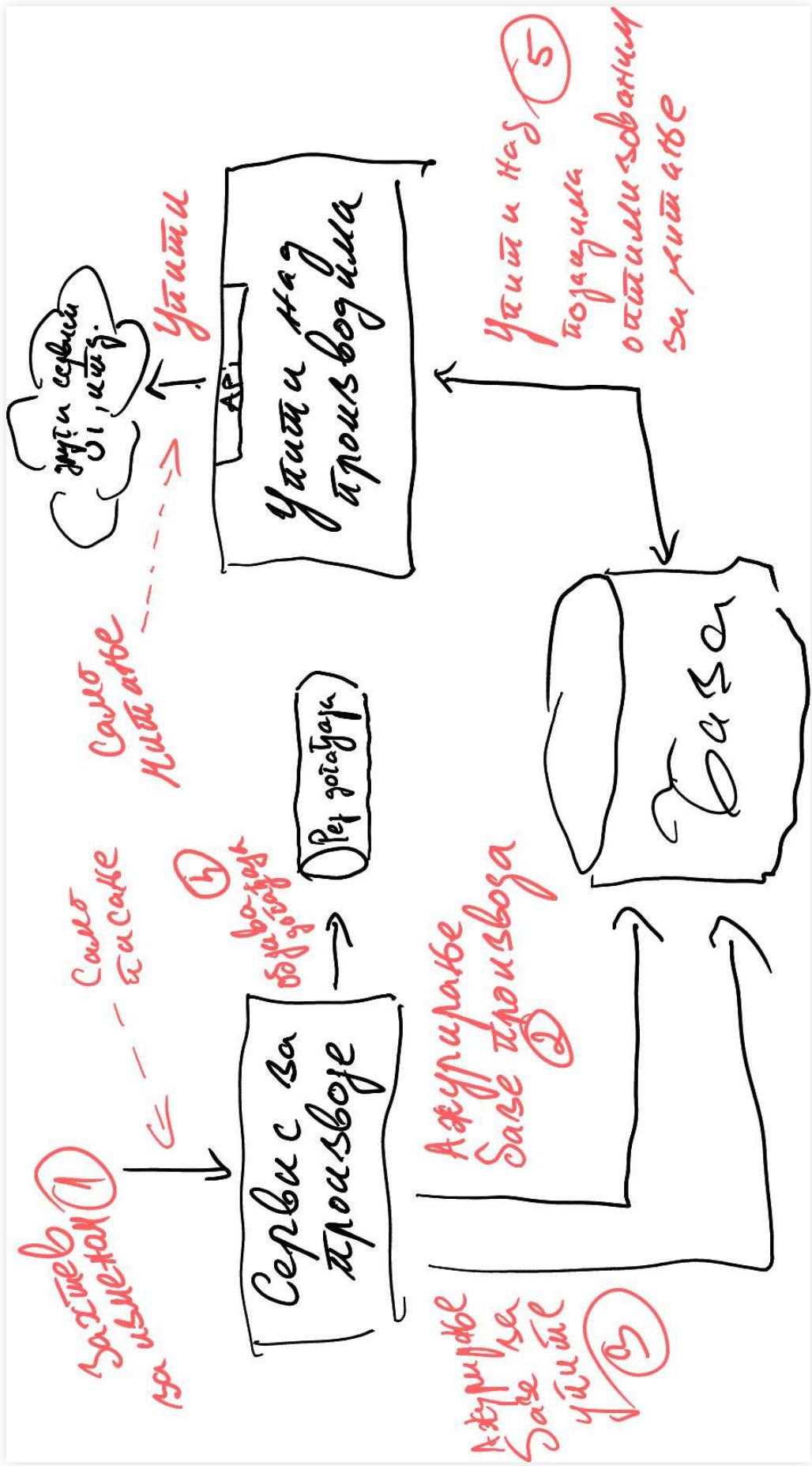
Структура - Посебне базе

- Захтев за изменом производа (1) уписује текуће стање у локалну базу (2) и објављује догађај (3).
- Сервис за упите, на основу догађаја, ажурира (4) свој интерни модел оптимизован за упите који чува у локалној бази.



Структура - јединствена база

- Захтев за изменом производа (1) уписује тренутно стање у локалну базу (2) и ажурира модел за упите (3) и затим објављује догађај (4).
- Срвис за упите чита ажуран модел за упите (5).



Предности

- Неминован код употребе обрасца *Event sourcing*.
- Больа подела надлежности.
- Једноставнији упитни модел. Больје перформансе упита. Подаци су најчешће денормализовани у циљу постизања оптималних перформанси.

Мане

- Увећана сложеност.
- Кашњење у репликацији. Коначна конзистентност (*Eventual Consistency*).
- Дуплирање података. Постоји могућност неконзистенције.

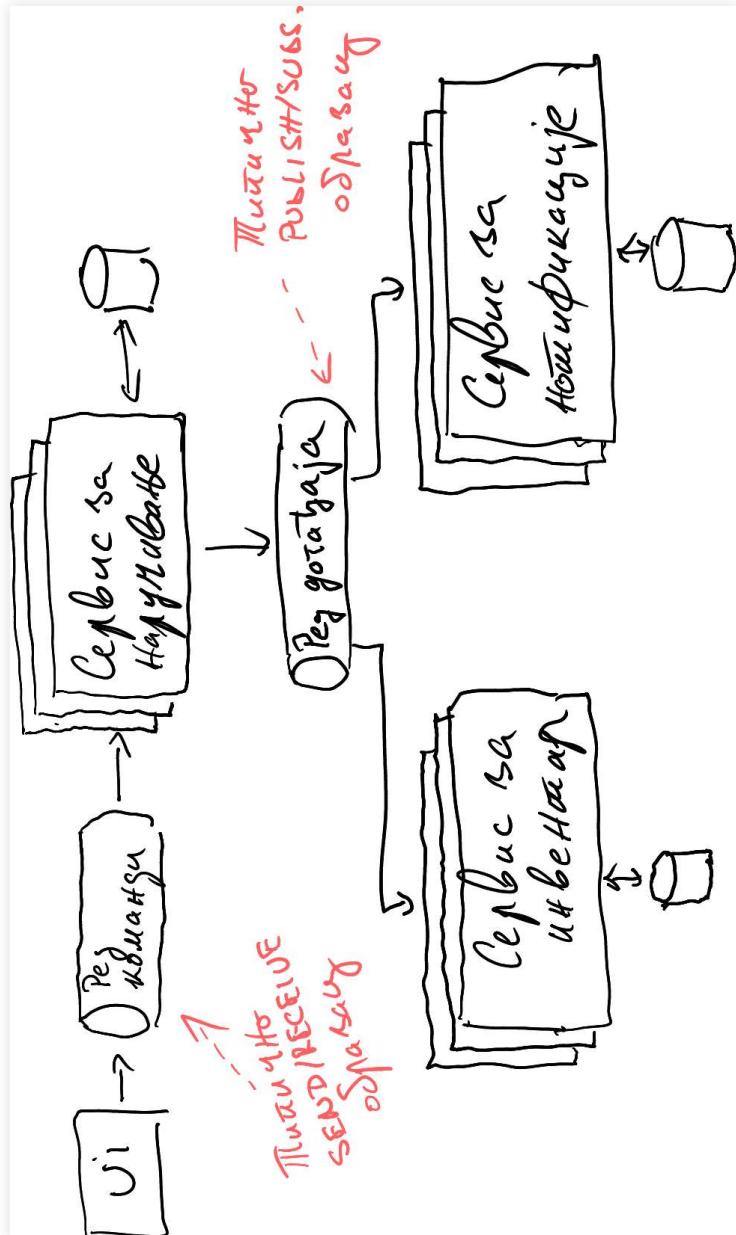
Комуникација

Messaging

- Слање порука је у основи архитектуре вођене догађајима.
- Асинхронна комуникација, слабо спрезање микросервиса.
- Посредник (*message broker*), који мора бити високо доступан, омогућава бафериовање и перзиstenцију порука.

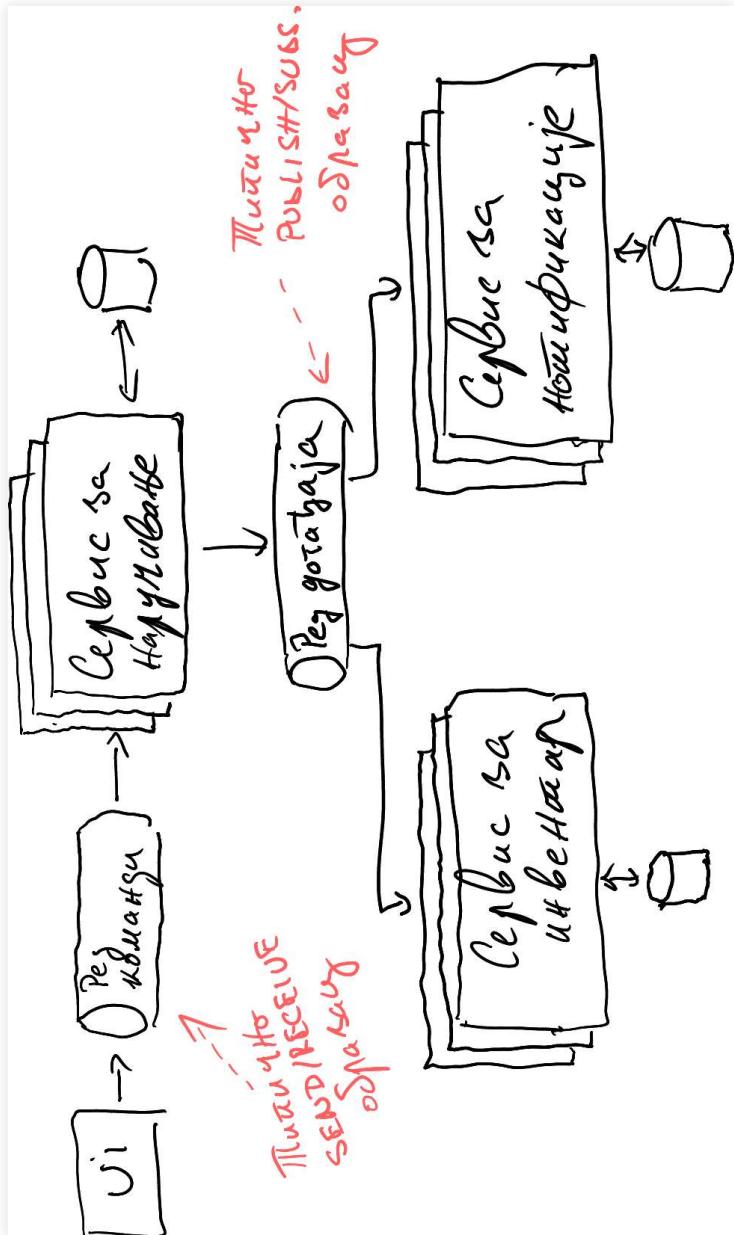
Send/Receive обраќац

- Обично представља комуникацију између два учесника (**point-to-point**) са специфичном наменом, најчешће извршење акције над циљним сервисом.
- Овај облик је коришћен типично од стране команди.
- Мора се обезбедити да само циљани сервис реагује на поруку.



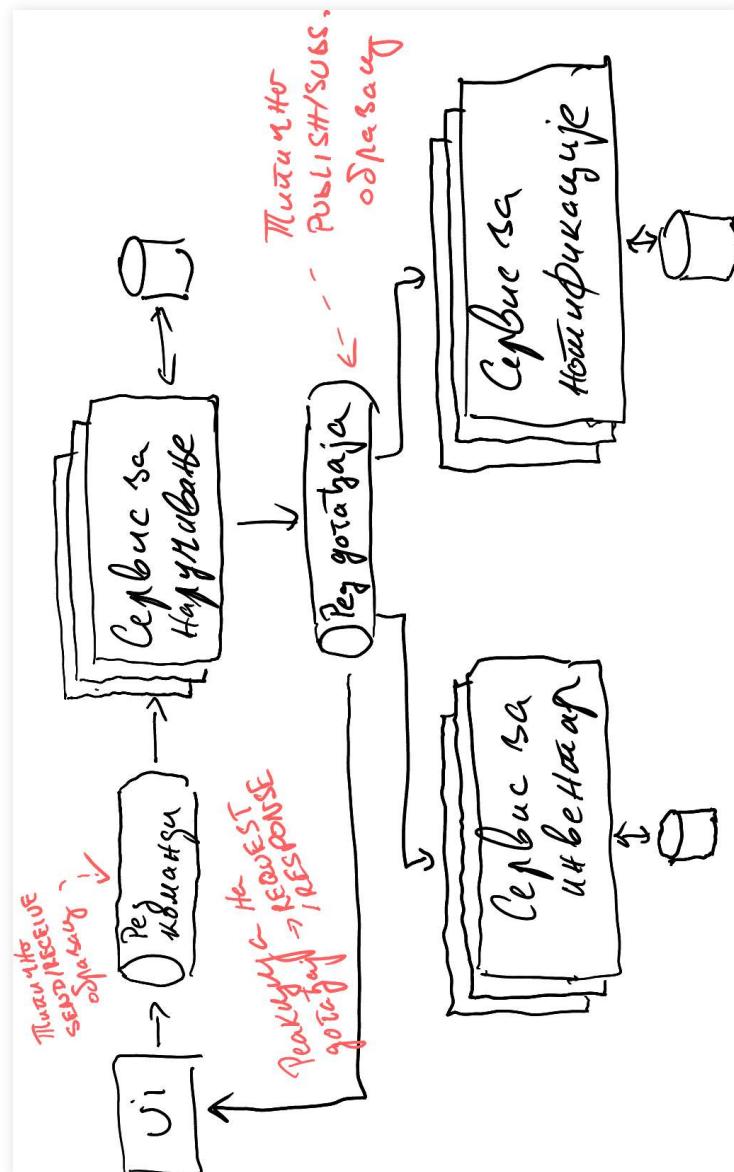
Publish/Subscribe обраザц

- Комуникација "један на више".
- Сервиси заинтересовани за одређене поруке се региструју (**subscribe**).
- Сервиси примају поруке и обраћају их у паралели различитом брзином.
- Основа хоризонталног скалирања.



Request/Response образац

- Имплементира се када је потребна повратна информација, обично при имплементацији *Send/Receive* образца.
- Команда и догађај као реакција на команду могу бити корелирани одређеним идентификатором.



Remote Procedure Invocation (RPI)

- Сервиси често морају тесније сарађивати да би обрадили одређени захтев.
- Понекад је синхрони начин комуникације бољи. Тада користимо *RPI*.

Предности

- Једноставан вид комуникације. Синхронна варијанта *Request/Response* образца.

Мане

- Јако темпорално спрезање сервиса. Морају бити доступни истовремено.

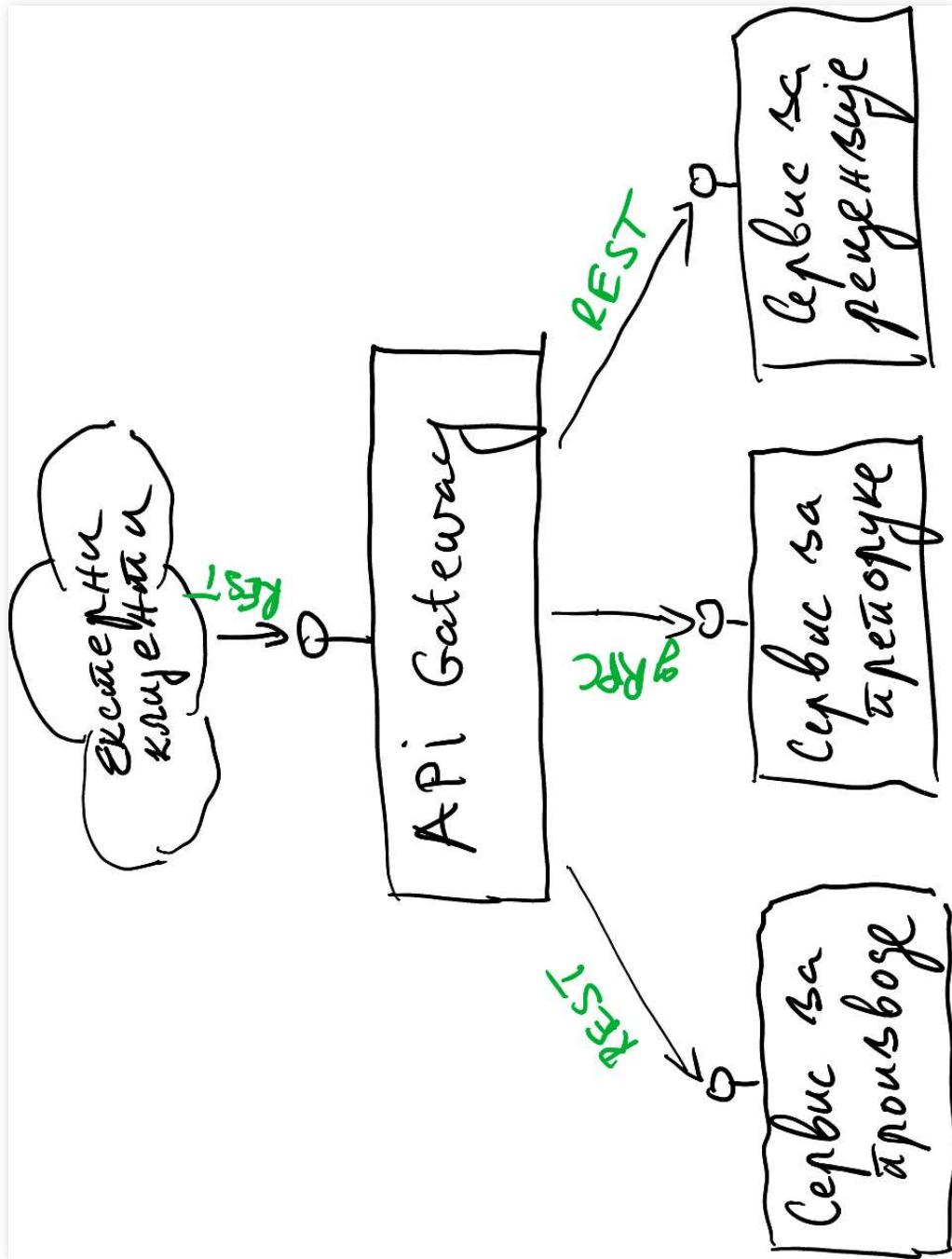
Приступи

- REST
 - Добро познат приступ. Основа комуникације на вебу.
 - Најчешће се користи у комбинацији са текстуалним порукама, нпр. JSON.
- gRPC
 - Развијен у Гуглу.
 - Акценат на перформансама.
 - Бинарне поруке базиране на технологији Protocol Buffers.
- Apache Thrift
 - Развијен у Фејсбуку.
 - Различити формати порука и транспортни протоколи.

API Gateway

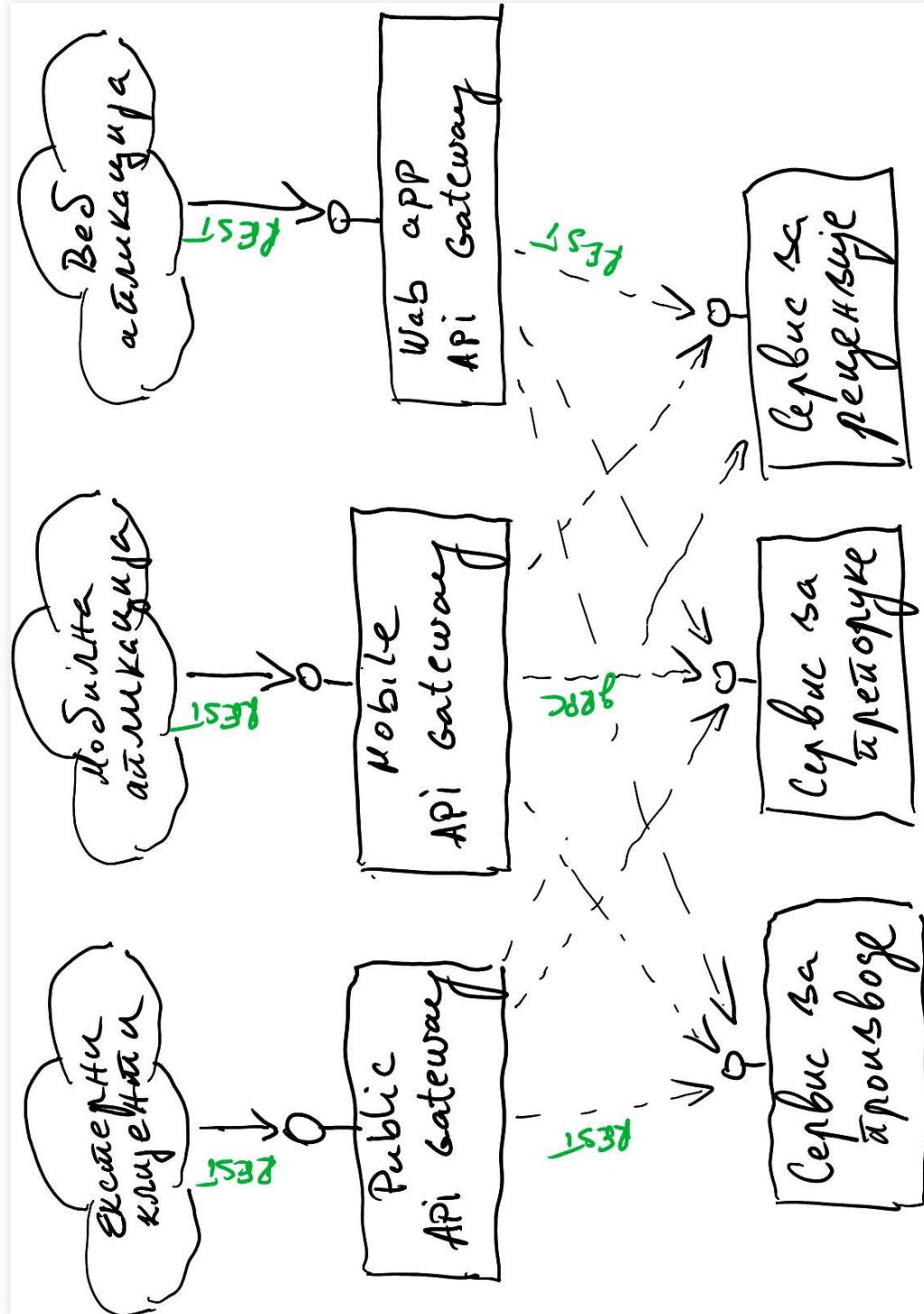
- Инстанца класичног ОО обрасца *Fascade*. Скривање интерне сложености.
- Посредник за спољне клијенте.
- Инстанца обрасца API композиције за екстерне клијенте.
Интеграција података са више микросервиса.
- Може имплементирати додатне функционалности, нпр. ауторизацију.

Структура



Варијанта - *Backends for Frontends*

- По један гејтвеј за сваки фронтенд.
- Специјализација API-ја.



Напомена

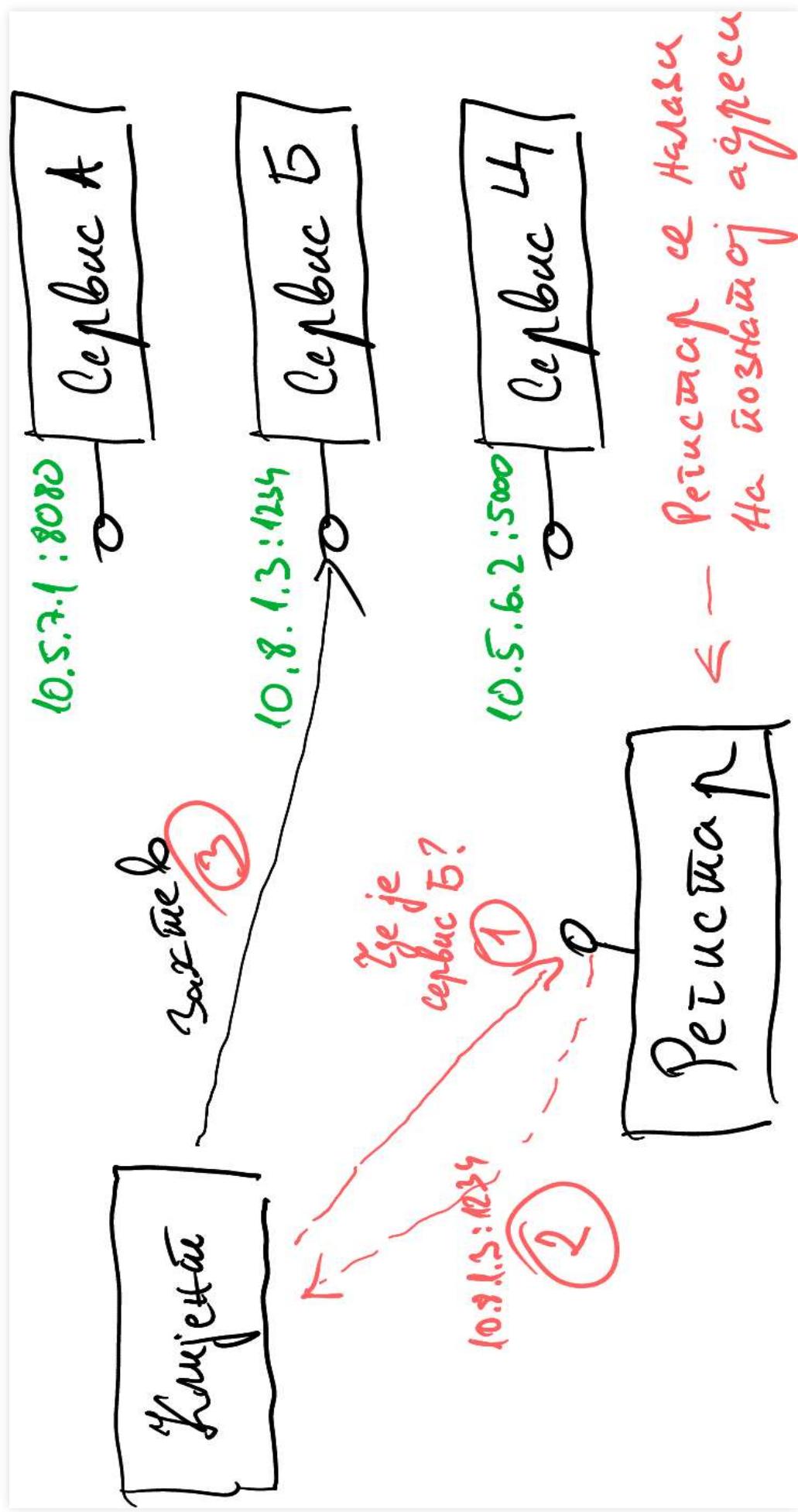
- Једна тачка отказа. Обезбедити високу доступност.

Откриване на сервиса

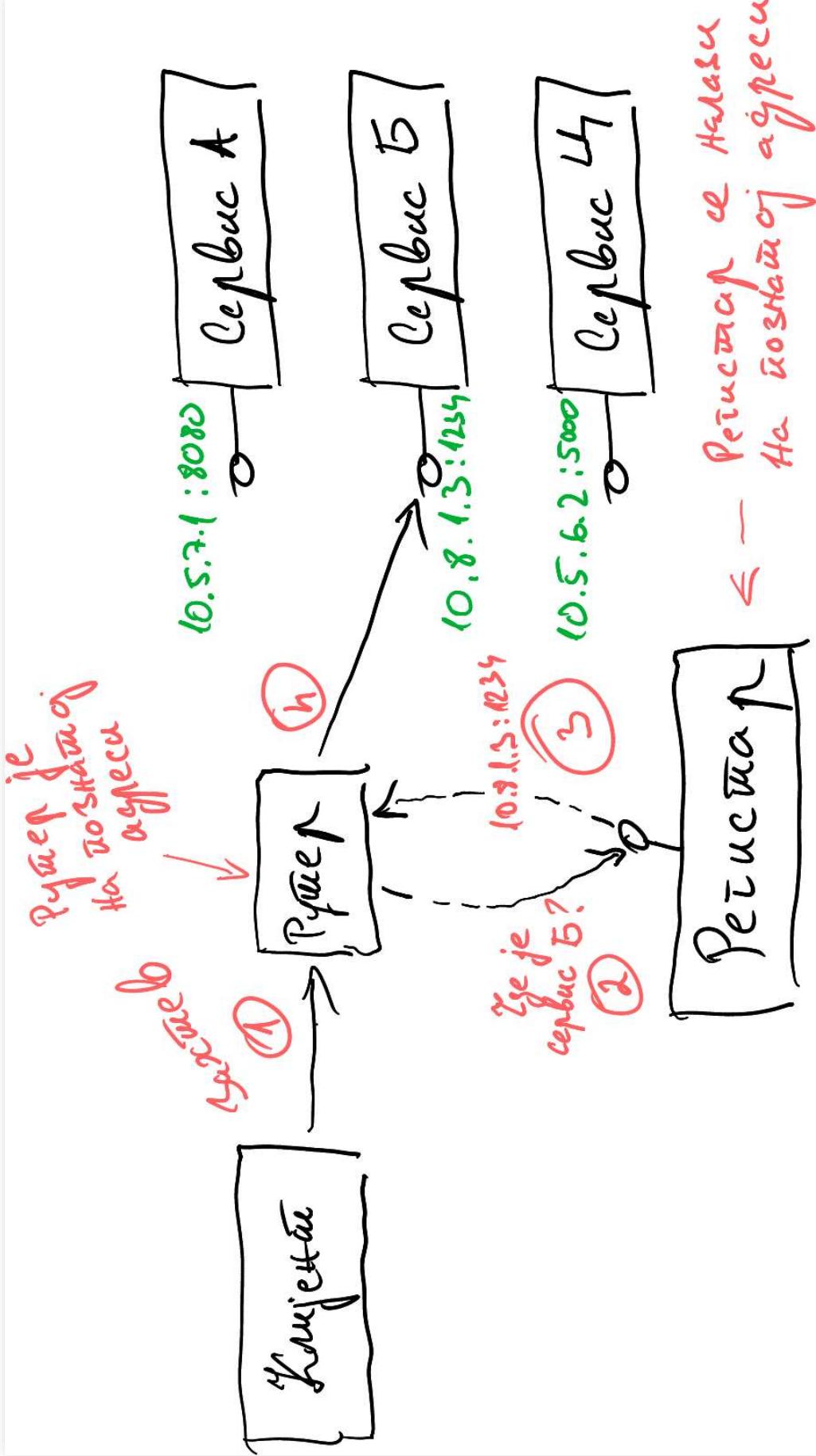
Service Registry

- За разлику од класичних дистрибуираних система код микросервисних архитектура сервиси нису увек на истој мрежној адреси.
 - Како клијент сазнаје где се сервис налази?
 - Специјални сервис који је увек на истој локацији и који има информације о локацијама свих других сервиса.
- Два приступа:
 - *Клијентски* при којем клијент сервис сам пита регистар,
 - *Серверски* при којем имамо посредника (*рутер*) који поставља питање регистру.

Структура - клијентски



Структура - серверски



Литература

- Hugo Filipe Oliveira Rocha, *Practical Event-Driven Microservices Architecture*, Apress, 2022.
- Microsoft, *Cloud Design Patterns*
- Chris Richardson, *Microservice Architecture*
- Wikipedia, *Microservices*

