



انجمن انفورماتیک ایران

از سلسله وبینارهای

گروه تخصصی معماری سازمانی انجمن انفورماتیک ایران

جامعیت داده‌ها

در معماریهای مبتنی بر میکروسرویس

Data consistency in microservice based architecture

رضا گنجی

کارشناس ارشد فناوری اطلاعات

۳۰ فروردین ۱۴۰۱

جامعیت داده‌ها در معماریهای مبتنی بر میکروسرویس

Data consistency in microservice based architecture

آنچه خواهیم گفت...

معماری مونولوتیک

معماری میکروسرویس

مهاجرت از معماری مونولیتیک به میکروسرویس

تراکنش‌های توزیع شده

SAGA

مطالعه یک مورد کاربردی



به معماری از سیستم ها اطلاق می شود که تمام اجزای سیستم در قالب یک کل واحد بالا می آید به نحوی که هر تغییری در هر بخش ممکن است منجر به تغییراتی در بخش های دیگر بشود.



مشکلات معماری مونولیتیک

هزینه نوآوری بالا

وابستگی بیش از حد سیستم ها

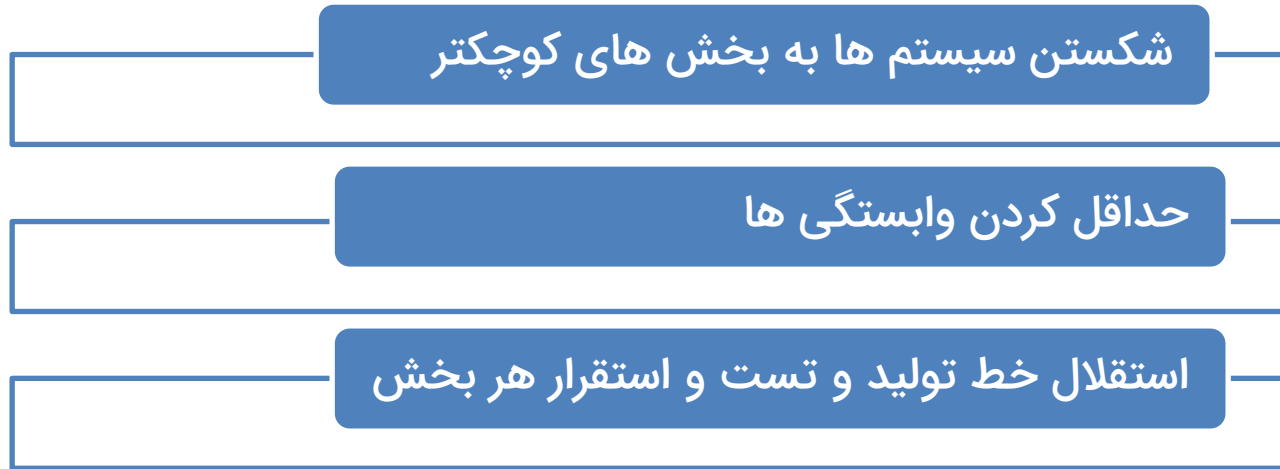
هزینه تولید بالا

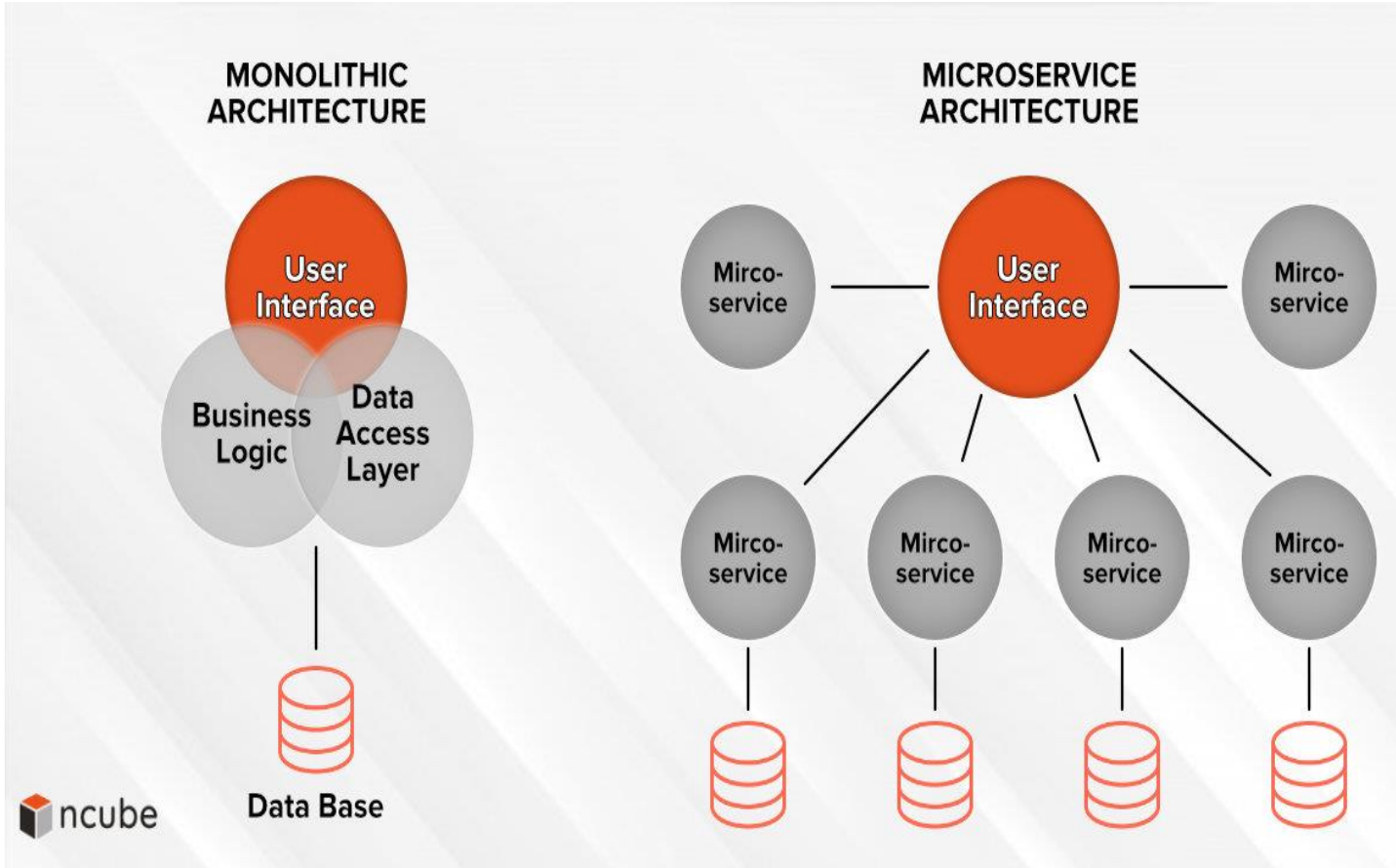
محدودیت تکنولوژی

عدم چابکی

تشابه خطرناک مشکلات در معماری مونولیتیک

مشکلات موجود در زیر ساخت های نرم افزاری بانکی به طرز خطرناکی شبیه به هم هستند, با مطالعه ای که در سال ۲۰۱۹ بر روی بانک های بریتانیا انجام شده , حداقل یک بانک در این کشور در روز یک مشکل جدی دارد , که منجر به بلاک شدن سپرده های حدود ۲ میلیون مشتری بانکی می شود و خسارتی بالغ بر ۳۳۰ میلیون پوند بر بانک وارد می کنند , با بررسی بیشتر مشخص شده که این موضوع ارتباط مستقیمی با بزرگی و پیچیدگی سیستم دارد , سیستم های نصب شده در این بانک ها به صورت کاملاً یکپارچه و در هم تنیده هستند و مشکلات در هر بخش منجر به مشکلات در بخش های دیگر می شود به عنوان مثال در بانک ها مشکلی در سیستم سپرده منجر به مشکل در سیستم صندوق هم می شود و نهایتاً منجر به از کار افتادن کل سیستم بانکی می شود.



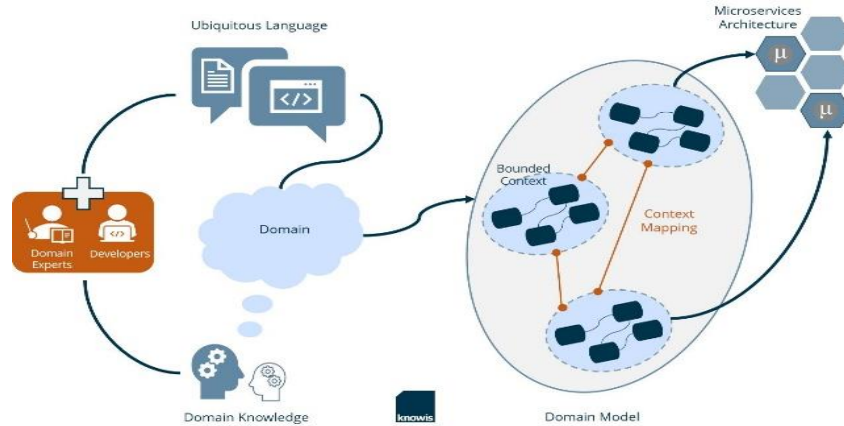


- کوچکتر شدن صورت مساله
- مقیاس پذیری
- توسعه سریع
- استقرار سریع
- رویکرد مشتری محور
- مهاجرت تدریجی به جای مهاجرت یکباره
- نوآوری پیوسته
- خط تولید و ادغام پیوسته

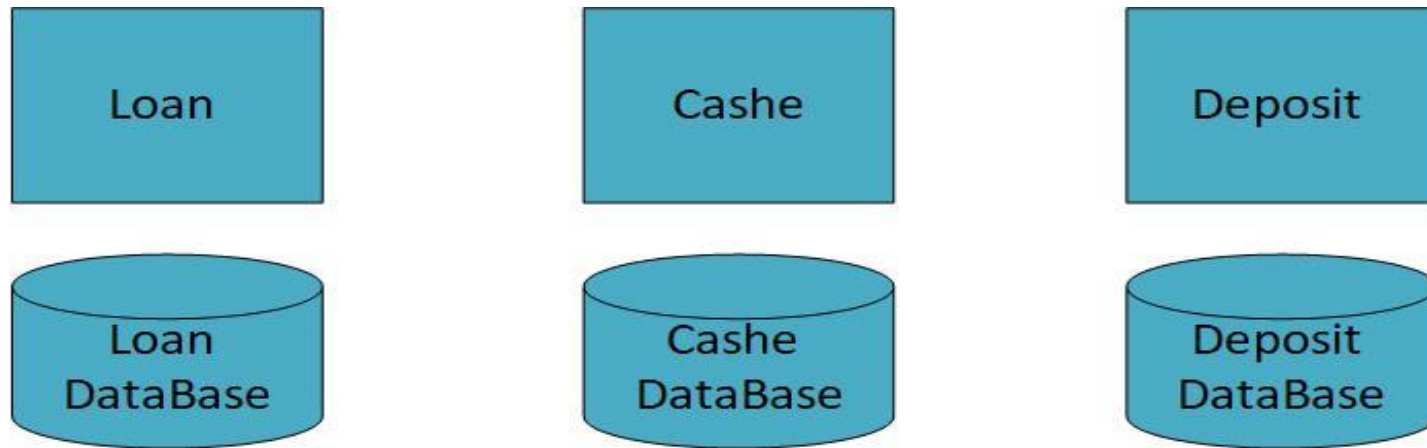
طراحی DDD پیشنهاد معماری میکروسرویس

این روش طراحی بر مبنای یک عنصر پایه با نام دامین استوار است تمام یا بخشی از بیزینس یک دامین در واقع یک محدوده دانشی است که بقیه اجزای معماری بر اساس آن چیده می شود.

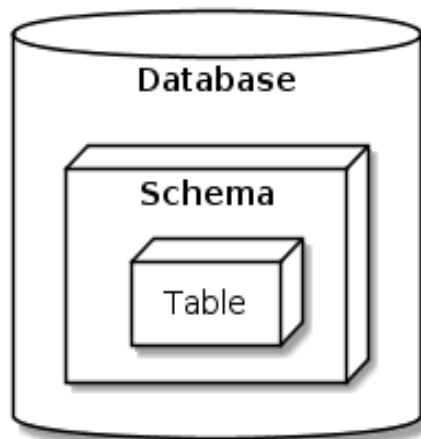
- Ubiquitous language
- Entity
- Value object
- Domain services
- Application service



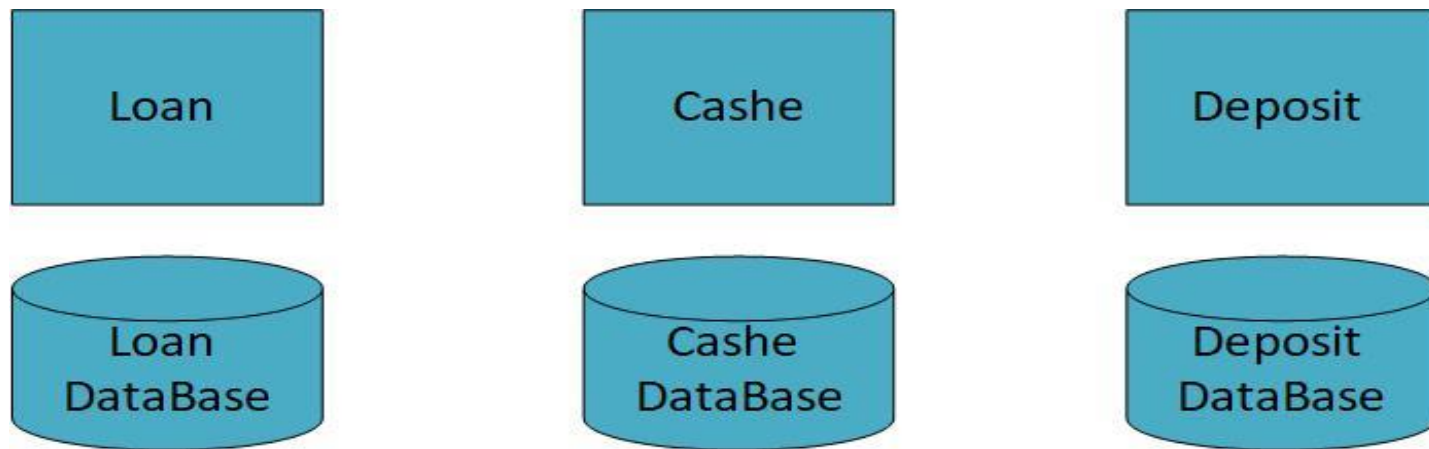
مهاجرت به میکرو سرویس - شکست دیتابیس و حذف دیتابیس مشترک



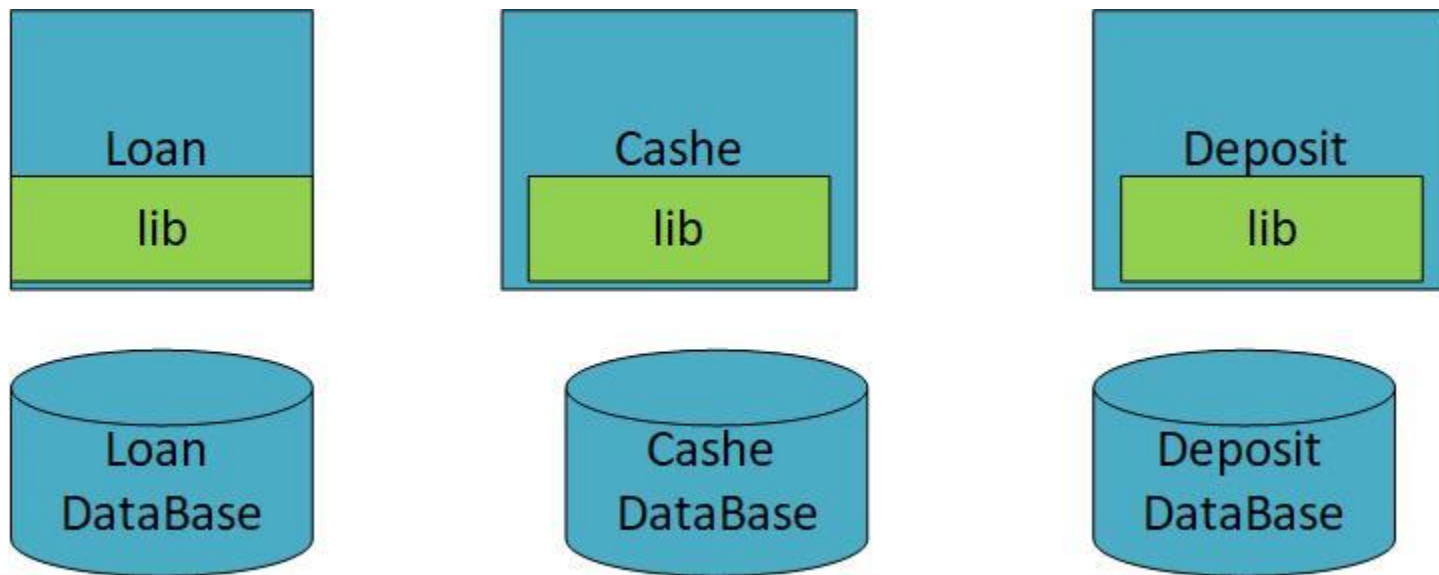
Private Database != Private Database Server



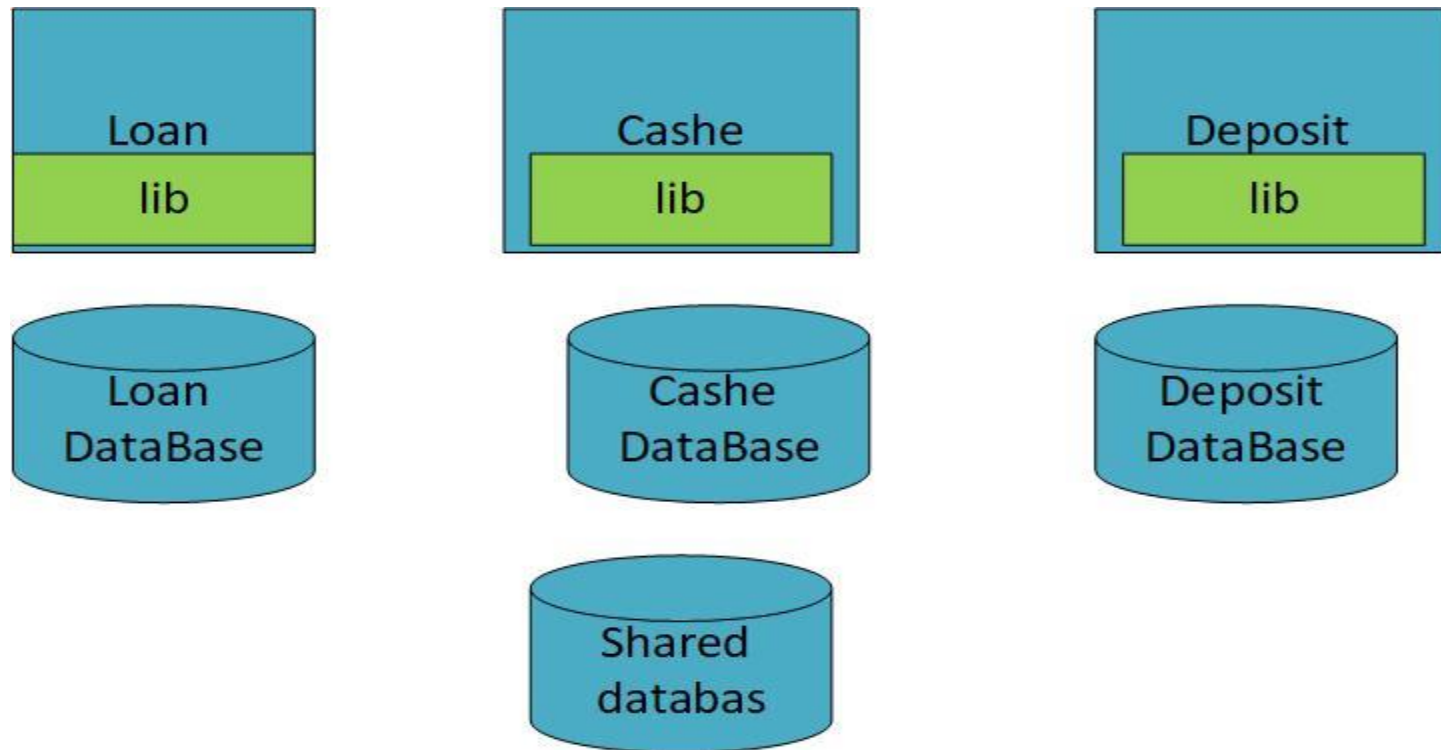
مهاجرت به میکرو سرویس - حداقل کردن فراخوانی های بین سیستمی

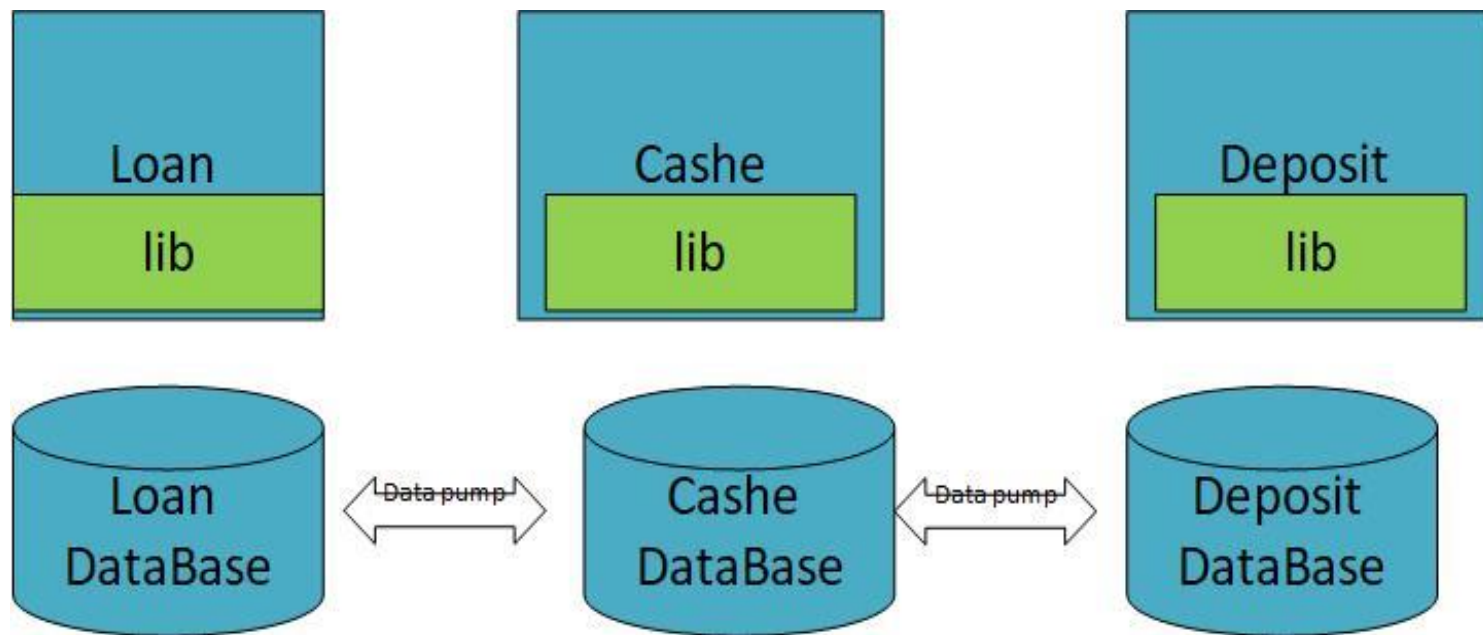


مهاجرت به میکرو سرویس - ایجاد کتابخانه مشترک

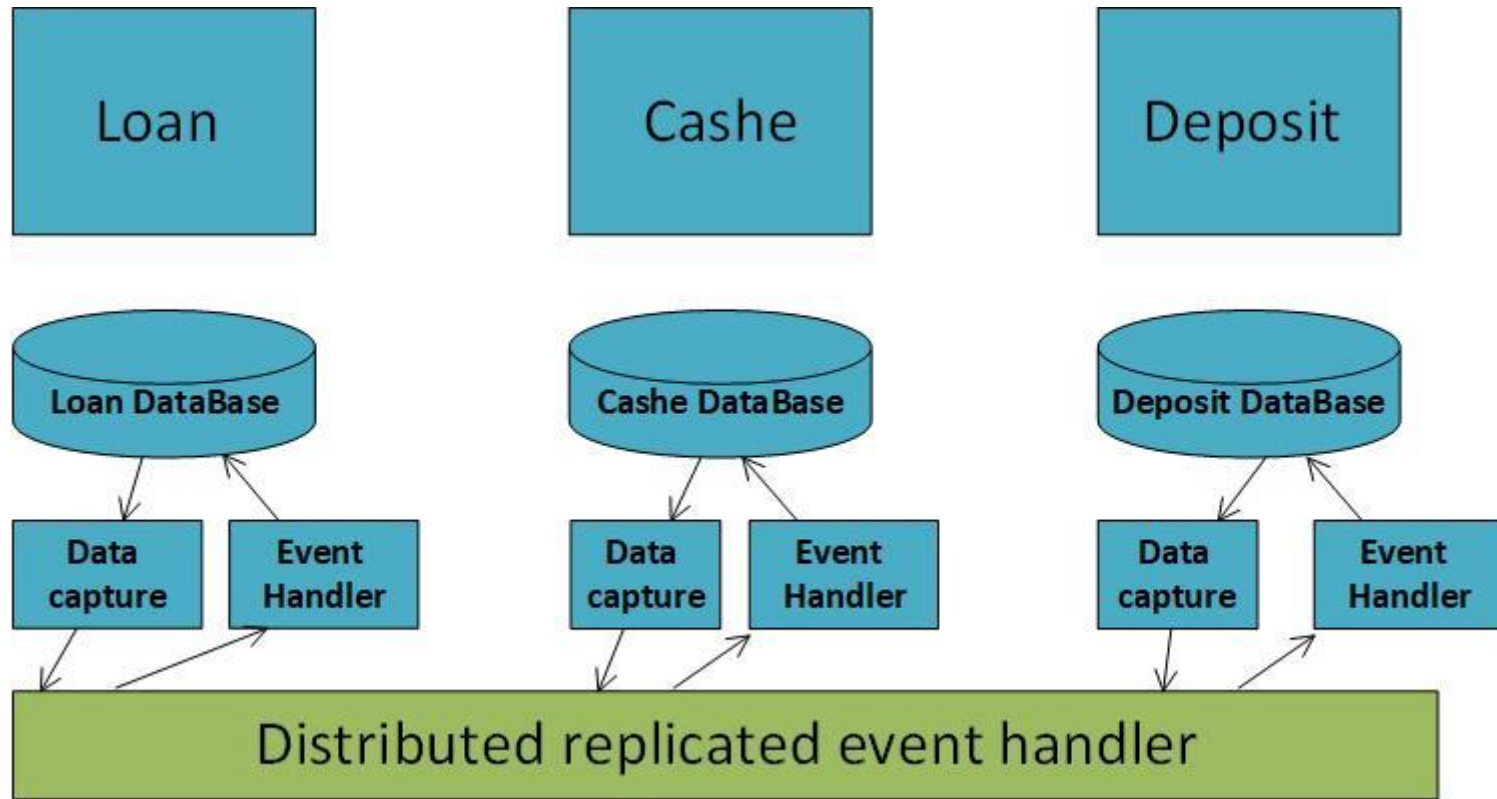


مهاجرت به میکرو سرویس - داده های مشترک

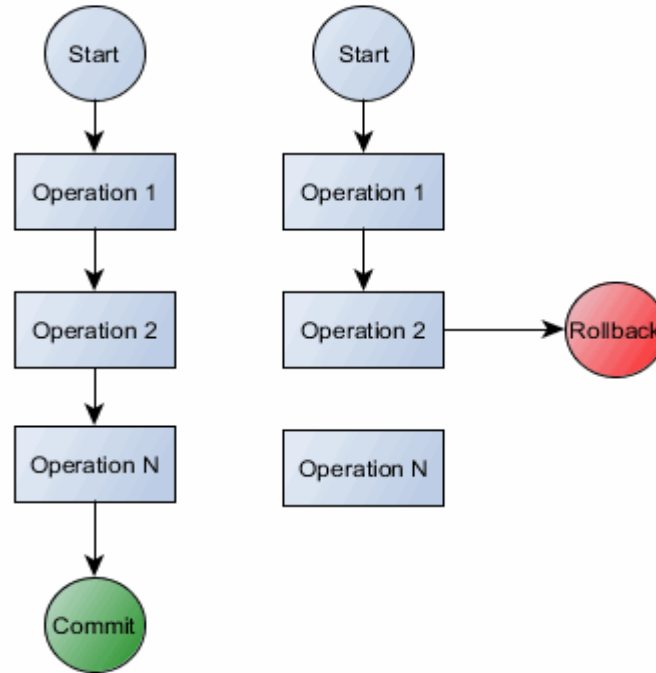




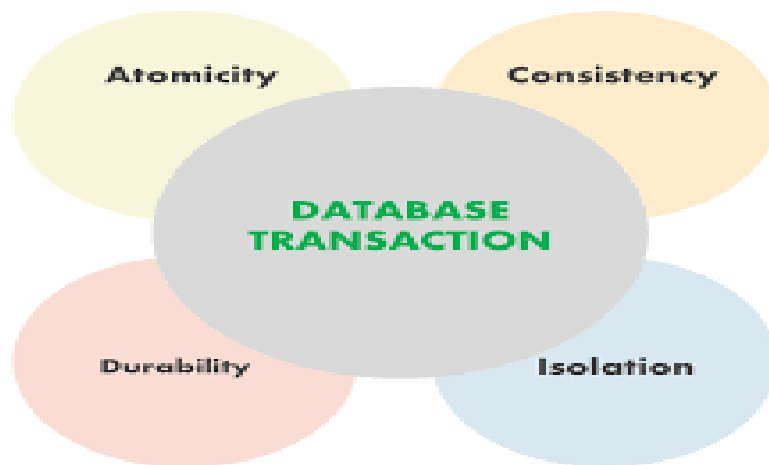
مهاجرت به میکرو سرویس - مدل غیر همزمان



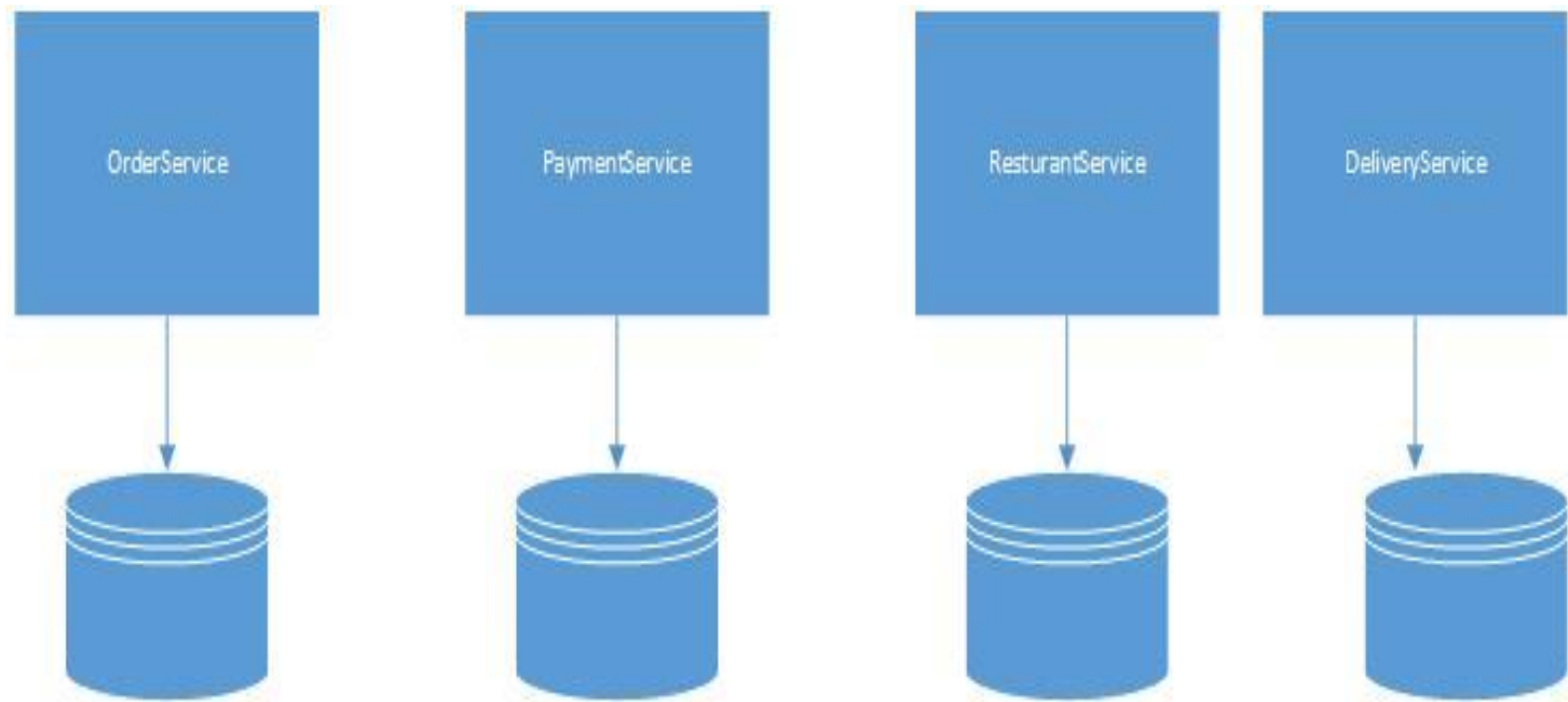
تراکنش مجموعه‌ای از عملیاتی پایگاه داده‌ها است که یا بایستی به طور کامل انجام شود یا در صورت بروز مشکل وضعیت به حالت قبلی بازگردانده شود.



- هر تراکنش باید مطلق باشد یا همه یا هیچ چیز باشد (A).
- باید محدودیت‌های رخ داده شده در سیستم را دنبال کند و سیستم را از یک حالت پایدار به حالت پایدار دیگر ببرد (C)
- باید مستقل باشد و برای سایر تراکنش‌ها قابل مشاهده باشد (I)
- پس از تکمیل شدن به طور مسلم باید یک وضعیت جدید در سیستم ایجاد کند ، یعنی باید پس از انجام حالت سیستم طبق قوانین پایگاه داده تغییر کند (D)

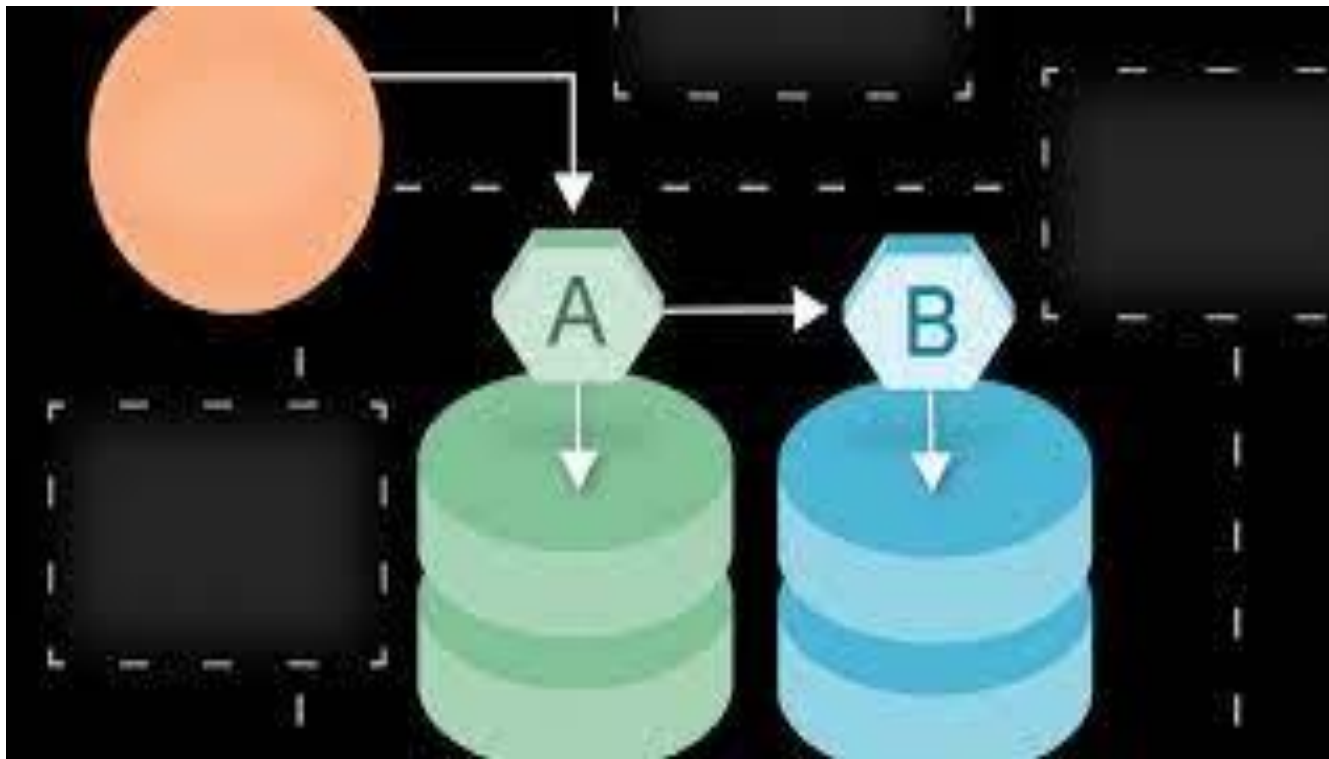


ACID در معماری های میکروسرویس؟

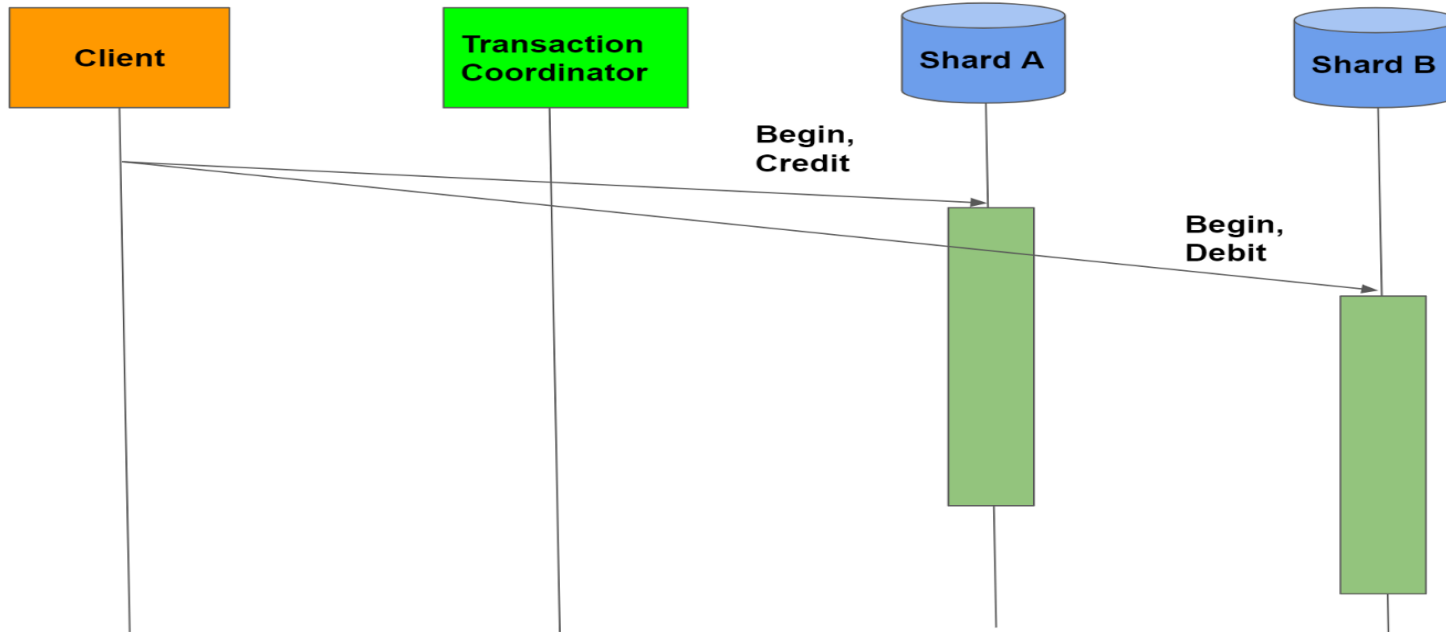


جامعیت داده ها در معماری های مبتنی بر میکروسرویس

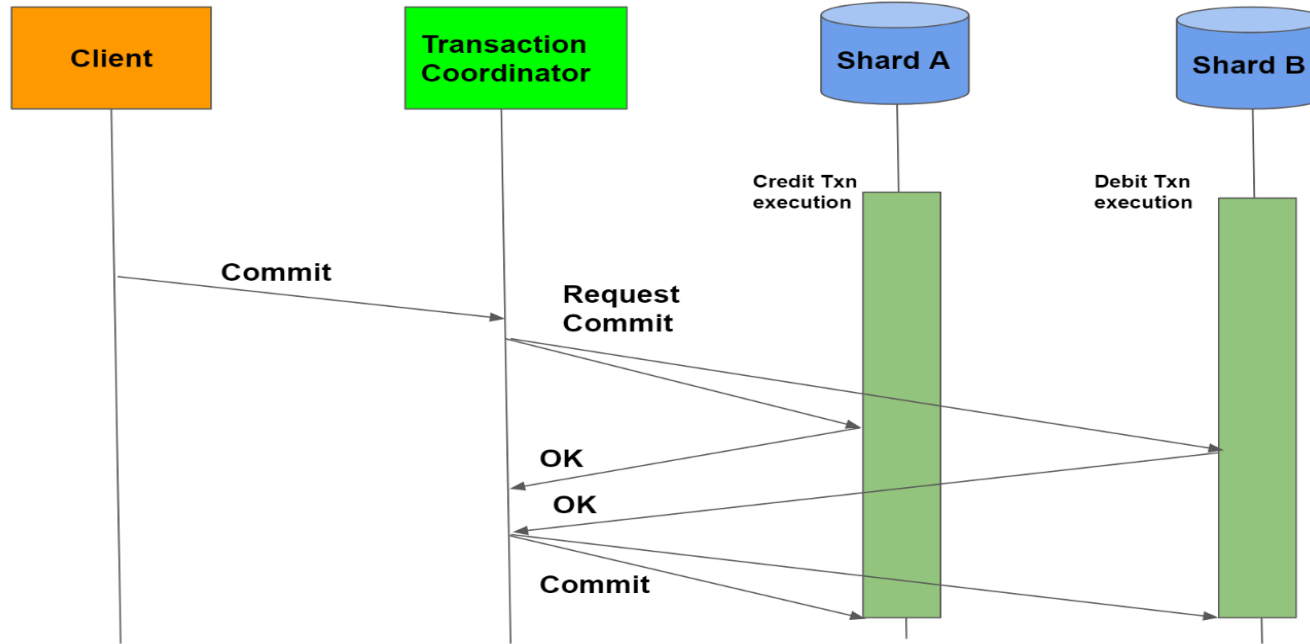
Data consistency in microservice based architecture



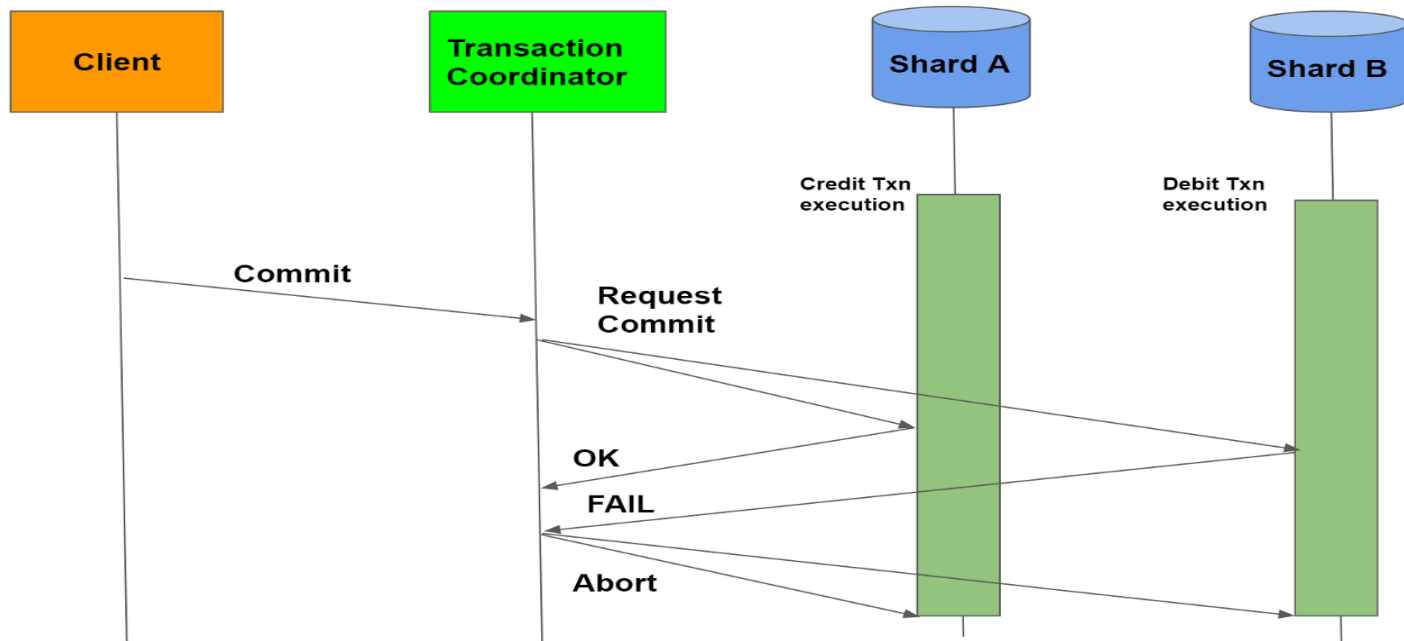
تراکنش های توزیع شده - فاز 1



تراکنش های توزیع شده - فاز 2 (حالت موفق)



تراکنش های توزیع شده - فاز 2 (سناریو خطا دار)



- مرتبه زمانی $O(n)$ به علاوه $O(2^n)$
- بازدهی و زمان پاسخ پایین به خاطر Lock های متعدد
- عدم پشتیبانی توسط خیلی از دیتابیس های NOSQL
- تاثیر روی availability در CAP

2PC is So Yesterday , Maintaining data Consistency with SAGA

SAGA pattern was first introduced to the world in 1987 by Hector Garcia-Molina and Kenneth Salem in an article named SAGAS.

SAGAS

*Hector Garcia-Molina
Kenneth Salem*

Department of Computer Science
Princeton University
Princeton, N J 08544

Abstract

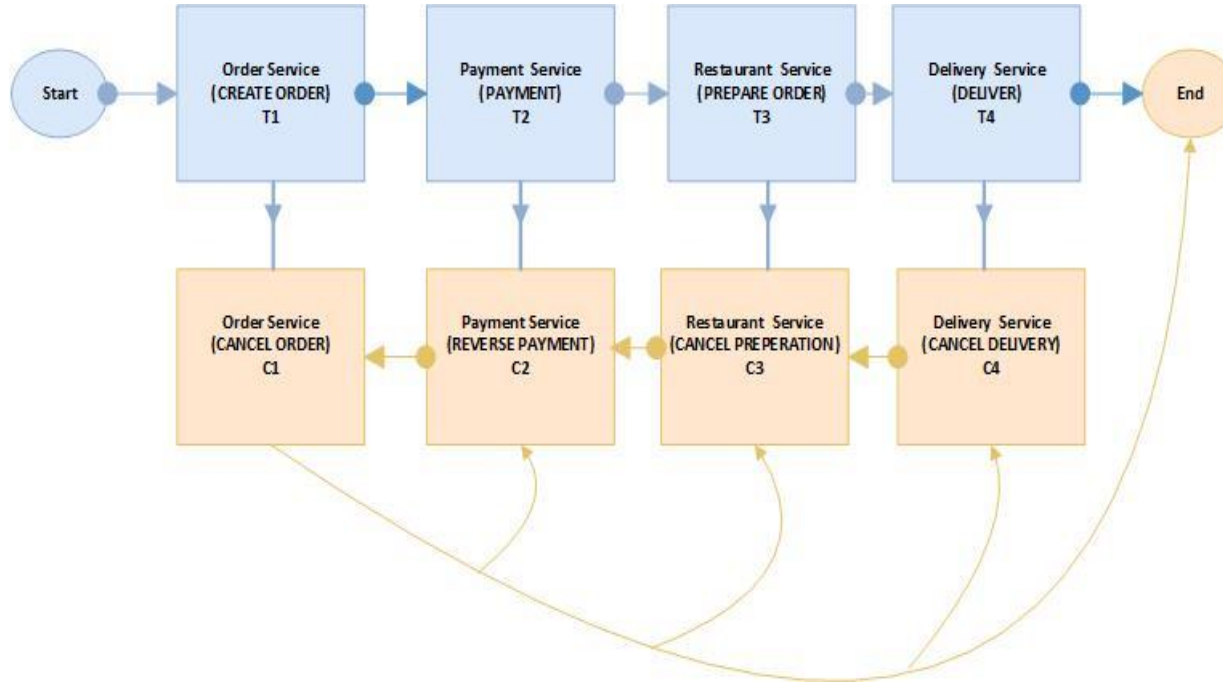
Long lived transactions (LLTs) hold on to database resources for relatively long periods of time, significantly delaying the termination of shorter and more common transactions. To alleviate these problems we propose the notion of a saga. A LLT is a saga if it can be written as a sequence of transactions that can be interleaved with other transactions. The database management system guarantees that either all the transactions in a saga are successfully completed or

the majority of other transactions either because it accesses many database objects, it has lengthy computations, it pauses for inputs from the users, or a combination of these factors. Examples of LLTs are transactions to produce monthly account statements at a bank, transactions to process claims at an insurance company, and transactions to collect statistics over an entire database [Gray81a].

In most cases, LLTs present serious performance problems. Since they are transactions, the

- مجموعه ای از تراکنش های مستقل محلی
- تراکنش ها با هم در ارتباط هستند.
- گارانتی این که یا همه تراکنش ها انجام می شوند یا خیر.
- هر میکروسرویس یک تراکنش دارد (Ti)
- هر تراکنش موظف است یک سرویس اصلی تراکنش و یک سرویس رولبک تراکنش را به بیرون ارائه دهد (Ci)

مدل یک اکوسیستم خرید از رستوران



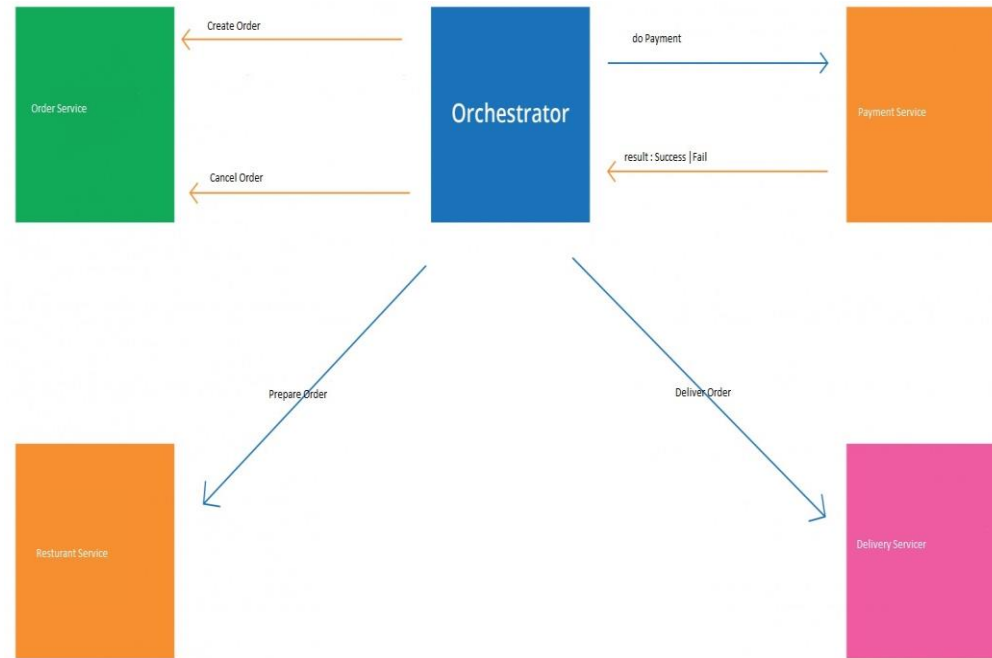
assumption 1 : for $n > 0$ if $T(n+1)$ fails then all $T(1)..T(n)$ should be failed
 $\Rightarrow$ All $C(1)..C(n)$ must be called.

assumption 2: compensation service must be idempotent and can not abort, they must be retried until succussed.

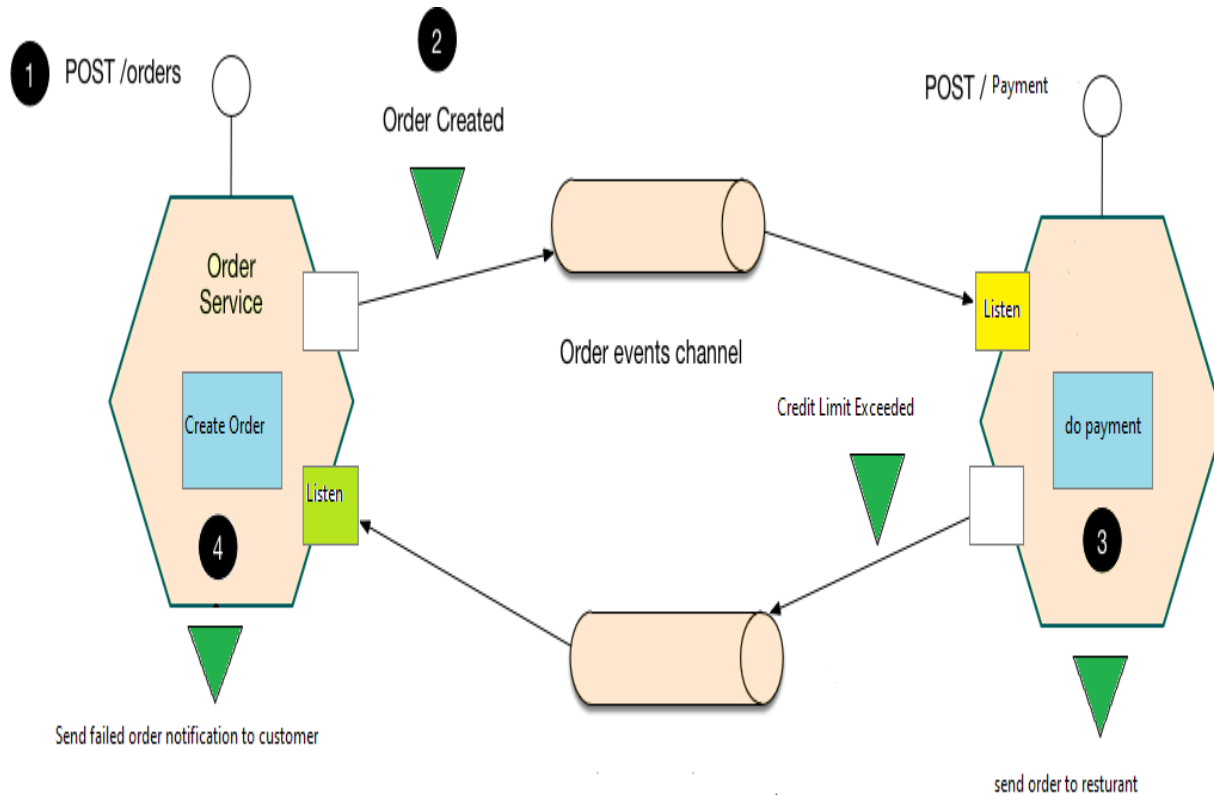
در اگو سیستم های مبتنی بر SAGA باید :

- واسط کاربری باید پیچیدگی های فراخوانی های غیر همزمان را از دید کاربر مخفی کند
- زمان پاسخ زیر ۱۰۰ میلی ثانیه باشد
- در صورت طول کشیدن پاسخ سیستم به هر دلیل باید سیستم پیغام انتظار مناسب بدهد
- از روش های مثل *push notification* می توان استفاده کرد

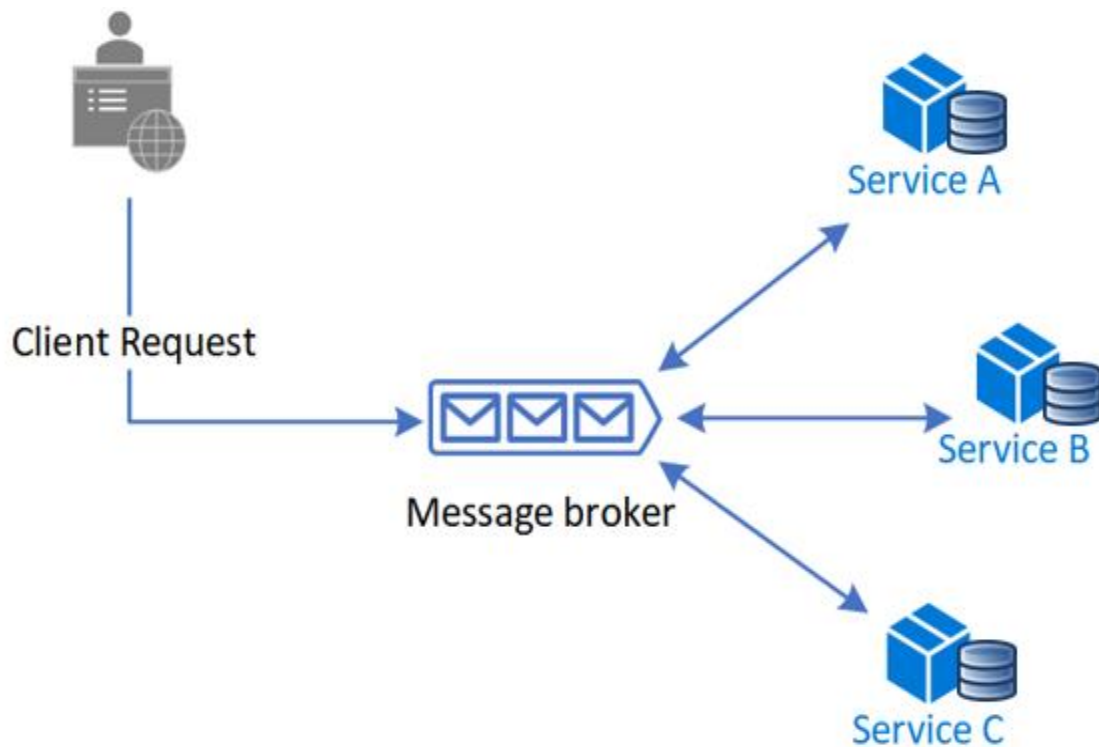
SAGA - Orchestration-Based SAGA

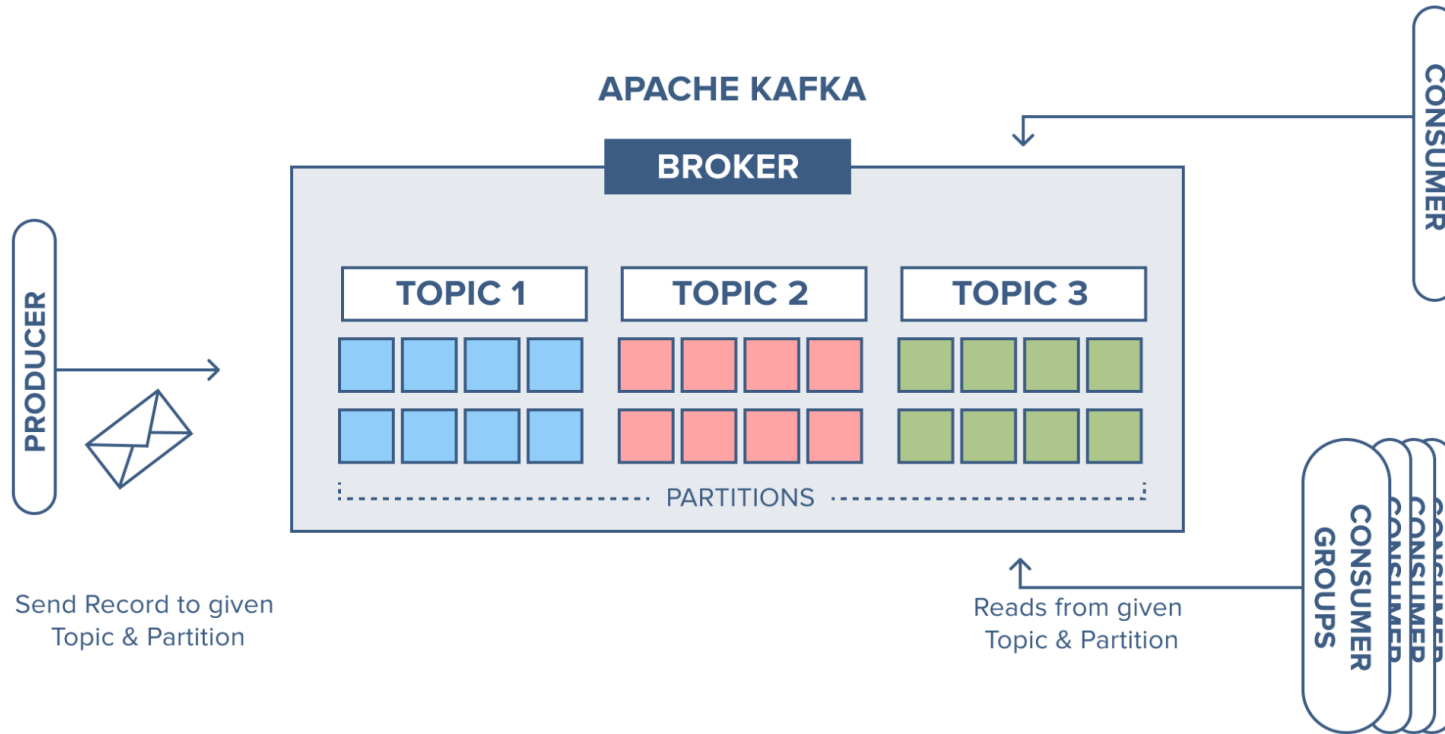


SAGA - Choreography-Based SAGA



Message broker یکی از ارکان اساسی SAGA





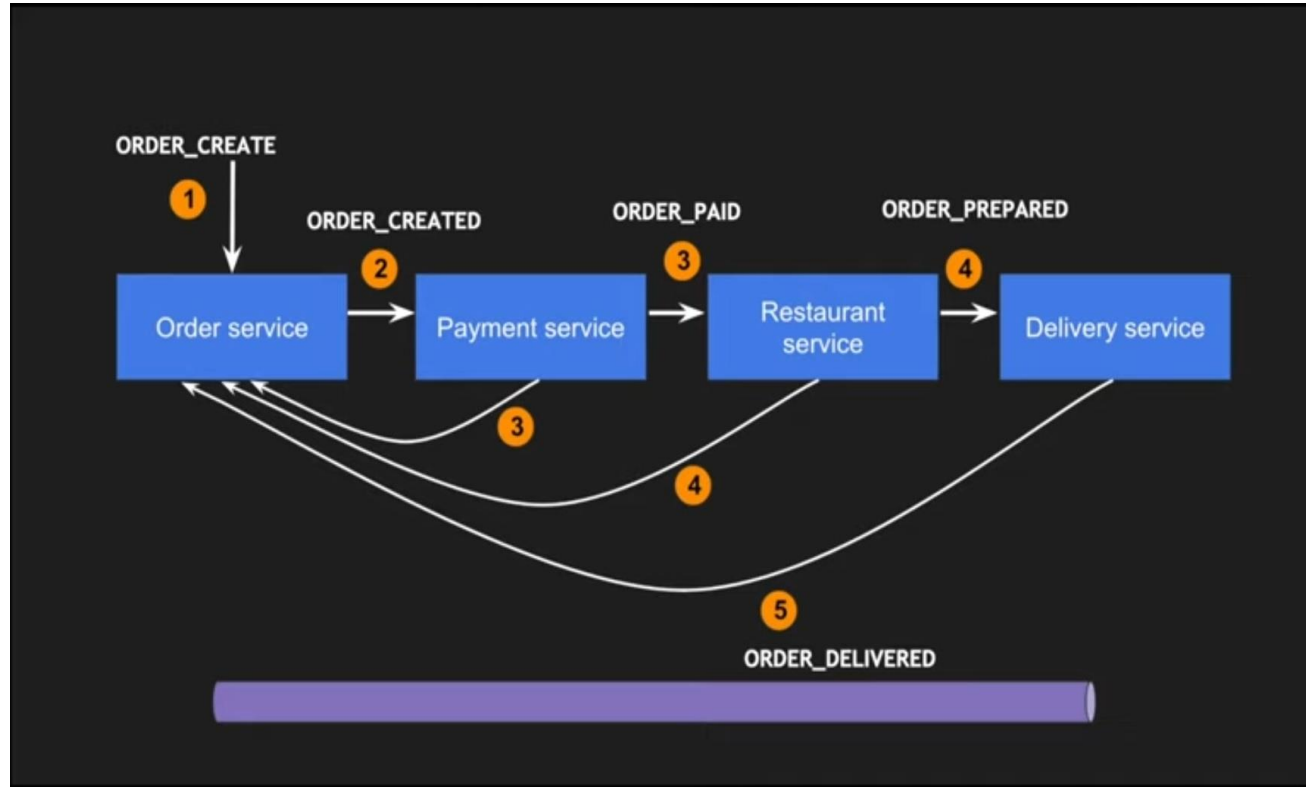
پشتیبانی از Isolation در SAGA ؟

- Atomicity**: All transactions should be executed or all compensated.
 - Consistency**: SAGA has two referential integrities: first, integrity within all microservices handled by local databases; second, integrity between microservices that handle the application.
 - Durability**: Handled by local databases
- For reasons as illustrated in the table below, we have a lack of Isolation in SAGA:

update	one microservice reads the data that can be changed with other microservice: 1- T(i) reads data 2- T(j) changes that data 3-T(i) changes the data =>T(j) will lose the data
Dirty Reads	microservice one writes the data; another one reads the data; then microservice one compensate the transaction : 1- T(i) writes the data 2-T(j) reads the data 3-T(i) compensates the transaction =>T(j) does not have a valid data
Non - repeatable/fuzzy reads	1- T(i) reads the data 2-T(k) writes the data 3-T(j) reads =>T(i) and T(j) have a different



مورد کاربردی (خرید از رستوران)







HOME > CONTROLCENTER.CLUSTER > TOPICS >

Cluster overview
Brokers
Topics
Connect
ksqldb
Consumers
Replicators
Cluster settings

Topic name* ⓘ

Number of partitions* ⓘ

Create with defaults

Customize settings

Cancel

Topic Summary

name
first

partitions
1

replication.factor
1

i For verifying high availability required by use cases in a production environment, start your Enterprise trial by adding [x](#) more brokers to your cluster and increasing your topic replication factor.

```
<dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka-streams</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-stream</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-stream-binder-kafka</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-stream-binder-kafka-streams</artifactId>
</dependency>
```

```
@Data@AllArgsConstructor@  
@NoArgsConstructorpublic  
class Customer {  
    private String name;  
    private String cardNo;  
}
```

```
public interface Topics {  
    String FAILED_PAYMENT="failed_payment_orders";  
    String DELIVARIABLE_ORDER="delivarable_orders";  
    String DELIVARIABLE_ORDERS="delivarable_orders";  
    String PENDING_ORDERS="pending_orders";  
    String  
    SUCCESS_PAYMENT_ORDERS="success_payment_orders";  
}
```

```
public interface Topics {  
    String FAILED_PAYMENT="failed_payment_orders";  
    String DELIVARIABLE_ORDER="delivarable_orders";  
    String DELIVARIABLE_ORDERS="delivarable_orders";  
    String PENDING_ORDERS="pending_orders";  
    String  
    SUCCESS_PAYMENT_ORDERS="success_payment_orders";  
}
```

```
@Service
public class OrderService {
    @Autowired private KafkaTemplate<String, Order>
    kafkaTemplate;

    public Order registerOrder(Order order){
        ListenableFuture<SendResult<String, Order>> future
        = kafkaTemplate.send(Topics.PENDING_ORDERS, order);
        return order;
    }
}
```

پیاده سازی نمونه یک سرویس Listener

```
import org.springframework.kafka.support.serializer.JsonDeserializer;
import java.util.HashMap;
import java.util.Map;

@Configuration
public class PaymentListener {

    @Value("${spring.kafka.bootstrap-servers}")
    private String bootstrapServers;

    @Autowired
    private KafkaTemplate<String, Order> kafkaTemplate;

    @KafkaListener(topics = Topics.FAILED_PAYMENT, groupId = "order", containerFactory = "kafkaListenerContainerFactory")
    public void listenToFailedPayments( Order order) {
        System.out.println("we have failed ordered and going rollback the order : " + order);
        kafkaTemplate.send(Topics.DELIVARABLE_ORDER, order);
    }

    @Bean
    public Map<String, Object> consumerConfigs() {
        JsonDeserializer<HeaderEnricher.Container> deserializer = new JsonDeserializer<>(HeaderEnricher.Container.class);
        deserializer.setRemoveTypeHeaders(false);
        deserializer.addTrustedPackages("");
        deserializer.setUseTypeMapperForKey(true);
        Map<String, Object> props = new HashMap<>();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
            bootstrapServers);
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
            StringDeserializer.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, JsonDeserializer.class);

        props.put(JsonDeserializer.TRUSTED_PACKAGES, Order.class.getPackage().getName());
        props.put(JsonDeserializer.VALUE_DEFAULT_TYPE, Order.class);
        props.put(JsonDeserializer.USE_TYPE_INFO_HEADERS, "false");

        return props;
    }

    @Bean
    public ConsumerFactory<String, Order> consumerFactory() {
        return new DefaultKafkaConsumerFactory<>(consumerConfigs());
    }
}
```

رضا گنجی

linkedin: <https://www.linkedin.com/in/reza-ganji-a131253a/>

Dzone : [DZone](#): Programming & DevOps news, tutorials & tools

GitHub : <https://github.com/rezaganjis>

email : ganji@tosan.com , rezaganjis@gmail.com



به گروه تخصصی معماری سازمانی انجمن انفورماتیک ایران بپیوندید...



گروه تخصصی معماری سازمانی
انجمن انفورماتیک ایران

[صفحه اول](#) [درباره ما](#) [اخبار و رویدادها](#) [شبکه فعالان معماری سازمانی](#) [پایگاه دانش](#) [کارگروه‌ها](#) [وبلاگ](#)

وبینارهای ماهانه سال ۱۴۰۱

گروه تخصصی معماری سازمانی

 معرفی طرح جاداب: تجربه‌نگاری یک پروژه معماری اشکان نظام‌پور - ایمان مهدوی آذر	 معماری سازمانی و بهره‌وری فاطمه بهلواني مرداد	 جامعیت داده‌ها در معماری میکروسرویس رضا گنجی فروردین
 برنامهریزی قابلیت‌مبنا: معرفی ابزارهای آنلاین علی رضایی دی	 وبینارهای ماهانه سال ۱۴۰۱ گروه تخصصی معماری سازمانی امیر مهجوریان شهریور	 آشنایی با استانداردهای ISO در حوزه معماری سازمانی سیدعلی آذرکار - رضا کریمی اردیبهشت
 آشنایی با معماری مرجع IT4IT رضا کریمی بهمن	 معماری مرجع بیمارستان‌های هوشمند دکتر رولف خیامی مهر	 مدل‌های شایستگی نیروی انسانی در معماری سازمانی حمیدرضا آقیری خرداد
 تحول دیجیتال در بانک‌ها: رهنمودهای عملی محمدجعفر زارعی اسفند	 قابلیت معماری سازمانی در ITIL و IT4IT وحید آخوندی آبان	 معماری سازمانی و تحول دیجیتال بتول ذاکری تیر

www.isi-ea.ir



www.isi-ea.ir