마이크로서비스 패턴

이연주

소프트웨어 아키텍처

"컴퓨팅 시스템 소프트웨어 아키텍처는 소프트웨어 엘리먼트와 그들간의 관계 그리고 이 둘의 속성으로 구성된 시스템을 추론하는 필요한 구조의 집합"

애플리케이션 아키텍처를 어떻게 선택하느냐에 따라
서비스 품질 요건(확장성, 신뢰성)과 **개발 시점의 품질요건(관리성, 테스트성, 배포성)**을 얼마나 충족할 수 있을지가 결정

모놀리식 아키텍처

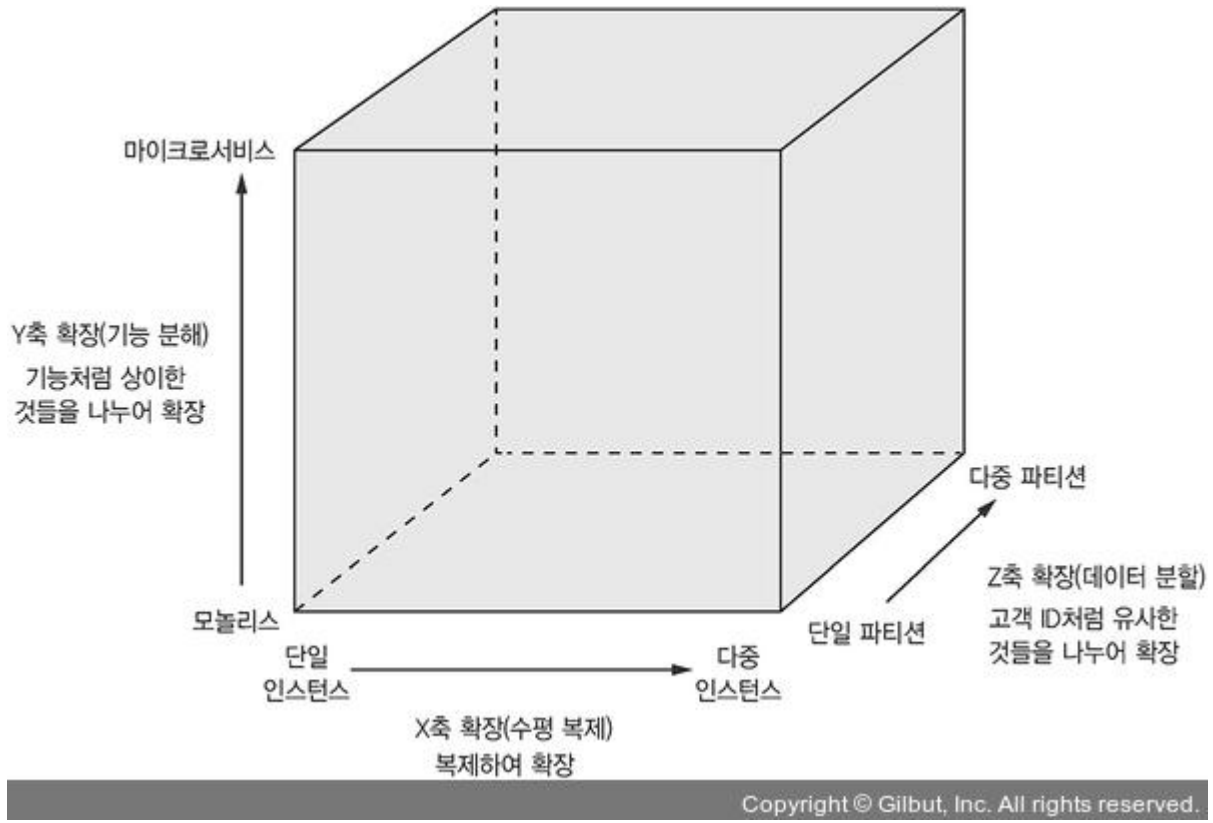
장점

- 개발이 간단하다
- 애플리케이션 쉽게 변경할 수 있다
- 테스트하기 쉽다
- 배포하기 쉽다 (1개의 WAR 파일울 톰캣으로)
- 확장하기 쉽다(로드밸런서 뒤에 애플리케이션 인스턴스 여러 개 실행)

단점

- 단일 코드베이스로 소통/조정 오버헤드
- 코드 수정 후 프로덕션에 반영되기까지 오래 걸림. 변경된 부분 수동테스트가 가능한 시점까지 큐에 대기
- 리소스 요건이 전혀 다른 컴포넌트(메모리 집약적/CPU 집약적) 함께 배포해야하고 확장시 리소스 결정 어려움

확장큐브



1. X축 확장: 다중 인스턴스에 고루 요청분산

   - 로드밸런서로 인스턴스에 부하분산하여 애플리케이션 능력과 가용성 개선

2. Z축 확장: 요청 속성별 라우팅

   - 인스턴스마다 배정된 하위집합 담당하여 특정데이터(userId)로 요청을 라우팅할 목적지(인스턴스) 결정

   - 트랜잭션 및 데이터 볼륨 처리시 사용

3. Y축 확장: 기능에 따라 애플리케이션 서비스로 분해

   - 애플리케이션 복잡도 해결 방안

   - 1개의 애플리케이션을 여러 서비스로 기능 분해

-> **서비스에 따라 X축/Z축 확장**

SOA

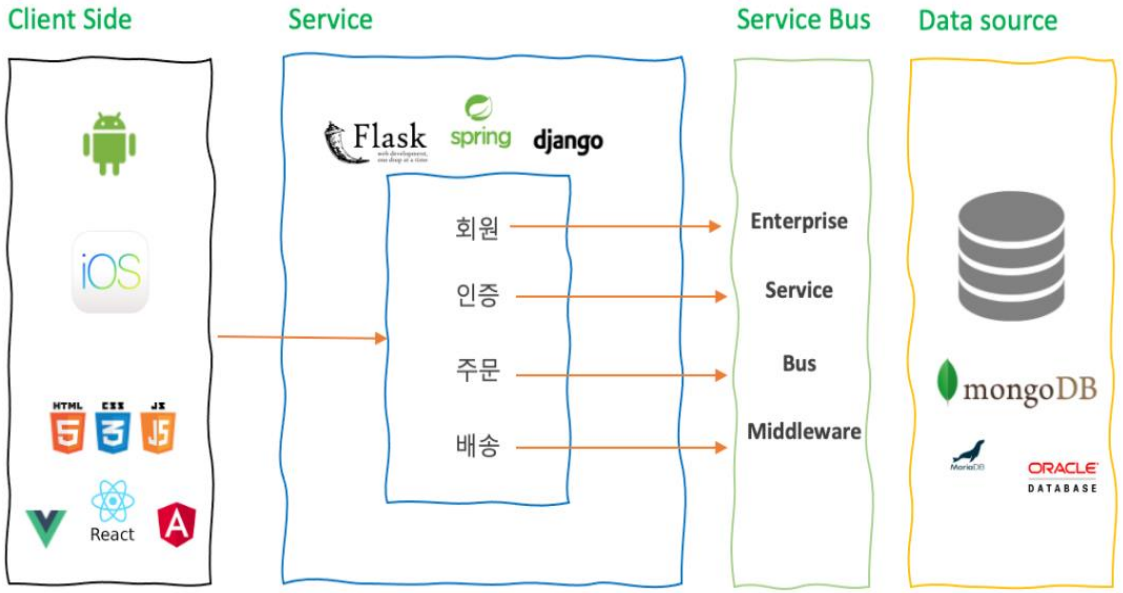
공통된 서비스를 **ESB**에 모아
사업 측면에서 공통 기능의 집합을 통해 서비스

ESB(Enterprise Service Bus)
- 서비스들을 컴포넌트화된 논리적 집합으로 묶는 미들웨어
- 이벤트 및 서비스에 대한 요청과 처리를 중개하여 인프라 전체 스트럭처에 분포시킴

서비스 = 기업의 업무

보통 전역 데이터 모델링하여 데이터 처리

한계
- SOAP 및 WS* 표준 등 무거운 기술 사용
- 하나의 DB를 사용한다는 점에서 끊어질 수 없는 의존성이 존재



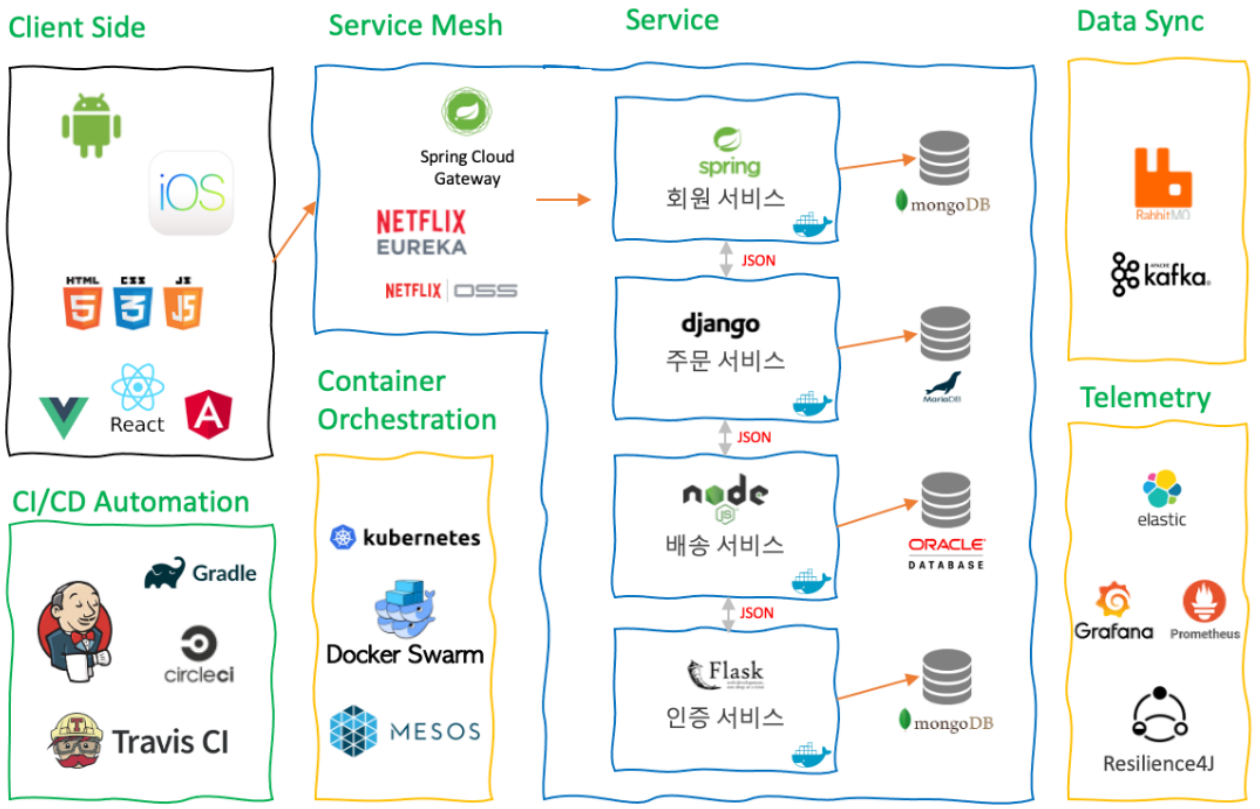
구분	SOA	마이크로서비스
서비스 간 통신	SOAP, WS* 표준처럼 무거운 프로토콜을 응용한 엔터프라이즈 서비스 버스 중심의 스마트 파이프(smart pipe)	REST나 gRPC처럼 가벼운 프로토콜을 응용한 메시지 브로커 또는 서비스 간 통신 중심의 덤 파이프(dumb pipe)
데이터	전역 데이터 모델 및 공유 DB	서비스 개별 데이터 모델 및 DB
주요 사례	대규모 모놀리식 애플리케이션	소규모 서비스

마이크로서비스 아키텍처

작고, 독립적으로 배포 가능한
각각의 기능을 수행하는 서비스로 구성하여
서비스를 나눠서 관리가능
(개발 언어, 환경, 기술 도입, 배포)

#decoupling #loosely
관리성, 테스트성, 배포성이 높은 애플리케이션 구축

- 서비스를 모듈성의 단위로 사용
- 한 가지 작은 서비스에 집중 강조
- 독립적인 서비스 (서비스간 공유 최소화 지향)
- 서비스마다 DB 따로 있다
    -> 런타임 시 서로 다른 서비스가
    Db lock을 획득해 서비스 블록킹 방지
    -> 자체DB 를 설치해야한다는 것이 아니다



마이크로서비스 아키텍처

장점

- 크고 복잡한 애플리케이션 **지속적으로 전달/배포** 가능
- 개발 생산성 증가(코드 이해도, IDE 실행 및 빌드 시간)
- 서비스를 독립적으로 배포/확장 가능
- 팀이 자율적으로 서비스 개발 및 운영 가능(신기술 도입, 스키마 수정)
- 결함 격리

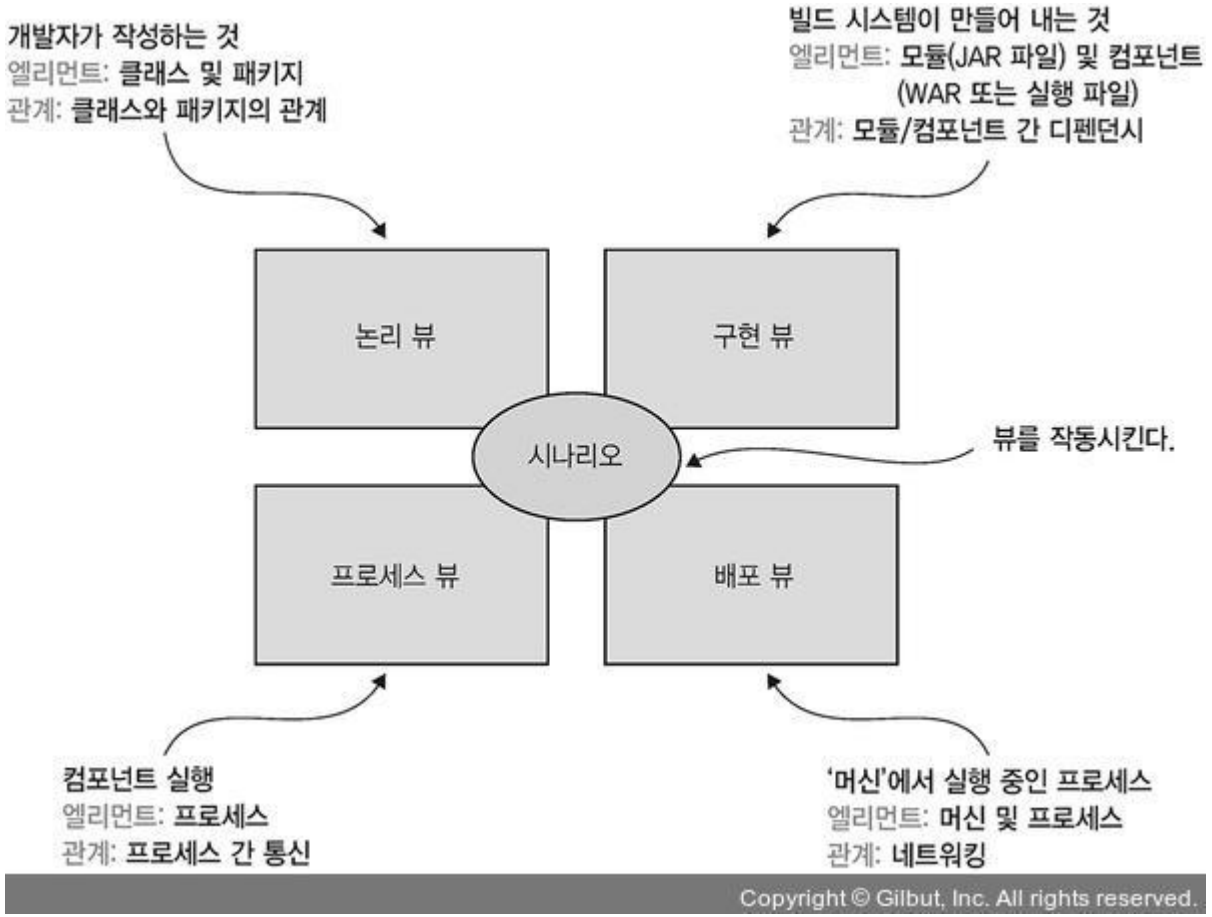
단점

- 서비스 분해
- MSA 도입시점 결정 어려움
- 분산시스템 구성 어려움(데이터 일관성을 위해 사가 패턴 적용, API 조합하거나 CQRS 뷰로 쿼리)

모놀리식, SOA,마이크로 아키텍처

아키텍처	특징	결합도	확장성	성공 사례	기술적 난이도	비용
Monolithic	**하나**의 프로젝트	매우 높음	매우 낮음	**매우 많음**	**쉬움**	**낮음**
SOA	**여러** 서비스, **하나**의 버스	낮음	높음	적음	어려움	중간
Microservice	**독립된 여러** 서비스	**매우 낮음**	**매우 높음**	많음	어려움	매우 높음

소프트웨어 아키텍처의 4+1 뷰 모델

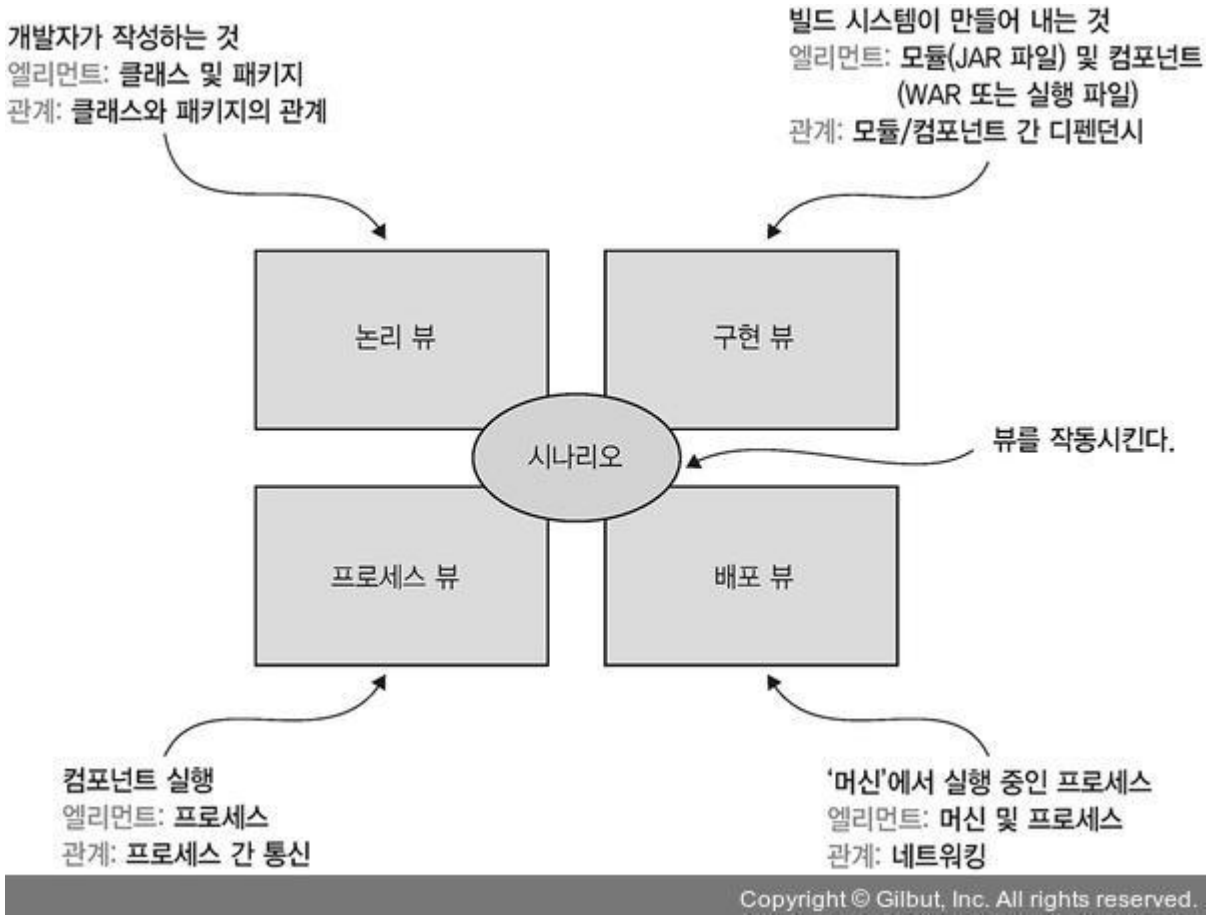


1. 애플리케이션 아키텍처를 바라보는 관점

2. 애플리케이션 아키텍처를 명쾌하게 표현

- 4뷰: 중요한 아키텍처 측면
- 시나리오: 뷰의 여러 엘리먼트가 협동하는 과정

소프트웨어 아키텍처의 4+1 뷰 모델



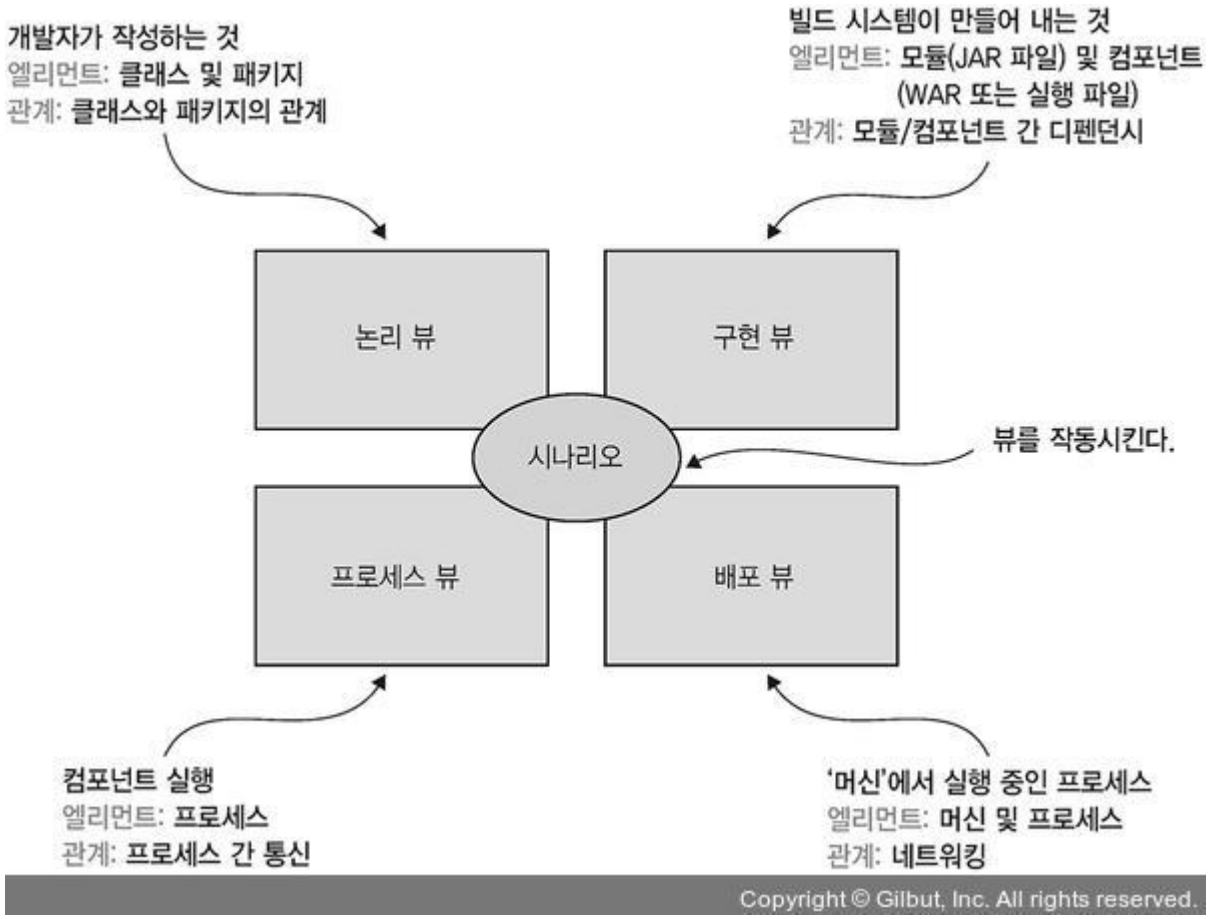
1. 논리 뷰(logical view): 개발자가 작성한 소프트웨어 엘리먼트

2. 구현 뷰(implementation view): 빌드 시스템의 결과물. 모듈(패키징된 코드)과 컴포넌트(하나 이상의 모듈로 구성된 실행/배포 가능 단위)로 구성됩니다. (EX) 자바에서 모듈은 보통 JAR 파일, 컴포넌트는 WAR 파일이나 실행 가능한 JAR 파일)

3. 프로세스 뷰(process view): 런타임 컴포넌트

4. 배포 뷰(deployment view): 프로세스가 머신에 매핑되는 방법

소프트웨어 아키텍처의 4+1 뷰 모델



5. 시나리오

각 뷰 내에서 얼마나 다양한 아키텍처 요소가 협동하여 요청을 처리하는지 기술

논리 뷰의 시나리오: 클래스가 협동하는 방법

프로세스 뷰의 시나리오: 프로세스가 서로 어떻게 협동하는 방법

계층화 아키텍처

1. 표현(프레젠테이션) 계층(presentation layer): 사용자 인터페이스 또는 외부 API가 구현된 계층

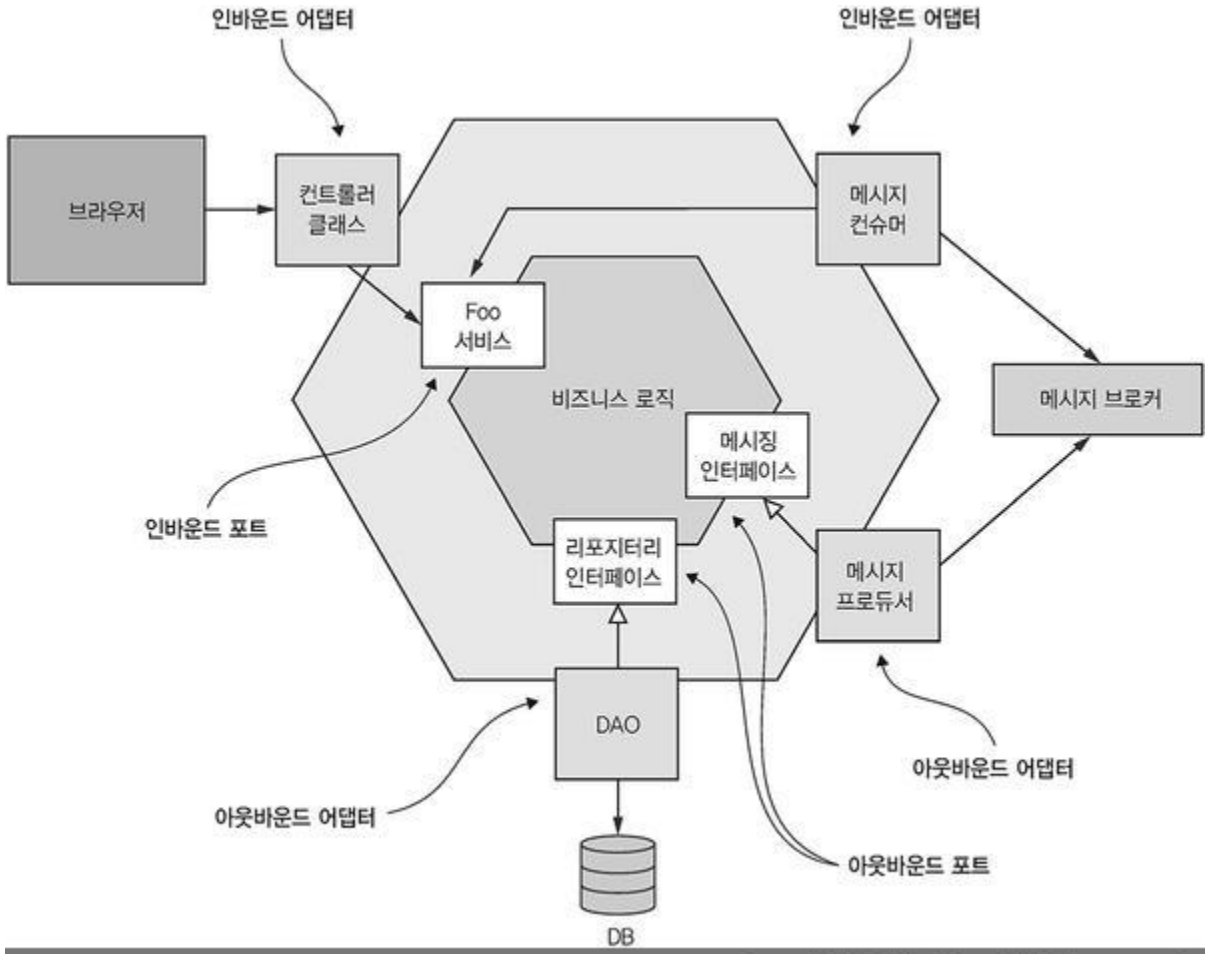
2. 비즈니스 로직 계층(business logic layer): 비즈니스 로직이 구현된 계층

3. 영속화(퍼시스턴스) 계층(persistence layer): DB 상호 작용 로직이 구현된 계층

한계

- 표현 계층이 하나뿐이다
- 영속화 계층이 하나뿐이다
- 비즈니스 로직 계층을 영속화 계층에 의존하는 형태로 정의한다
  (DB 없이 비즈니스 로직을 테스트 불가능)

육각형 아키텍처



'비즈니스 로직이 어댑터에 전혀 의존하지 않는다'

- 표현 계층 -> 인바운드 어댑터
  컨트롤러 클래스, 메시지컨슈머

- 영속화 계층 -> 아웃바운드 어댑터
  DAO, 메시지프로듀서

'**포트**'로 비즈니스 로직이 외부와 상호작용한다
자바에서 포트 역할을 하는 건 인터페이스
- 인바운드 포트
  서비스의 퍼블릭 메서드가 정의된 서비스 인터페이스
- 아웃바운드 포트
  데이터 접근 작업이 정의된 리포지토리 인터페이스

아키텍처 스타일 비교

모놀리식 아키텍처: 구현 뷰를 **단일 컴포넌트**로 구성

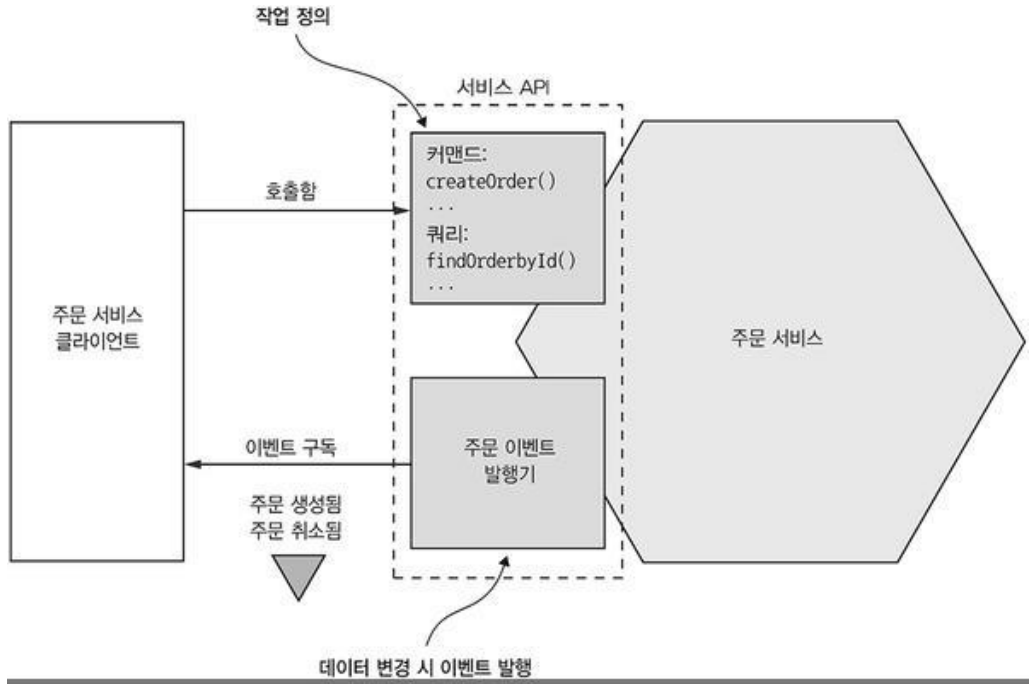
마이크로서비스 아키텍처: 구현 뷰를 **다수의 컴포넌트**로 구성
- 컴포넌트 = 서비스
- 서비스 = 자체 논리 뷰 아키텍처

마이크로서비스에서 서비스란?

어떤 기능이 구현되어 단독 배포가 가능한 소프트웨어 컴포넌트

서비스 역할 : 클라이언트가 해당 서비스의 기능을 쓸 수 있도록 커맨드, 쿼리, 이벤트로 구성된 api 제공

서비스 작업 종류
- 커맨드: CUD
- 쿼리: 조회 R



마이크로서비스 아키텍처 정의

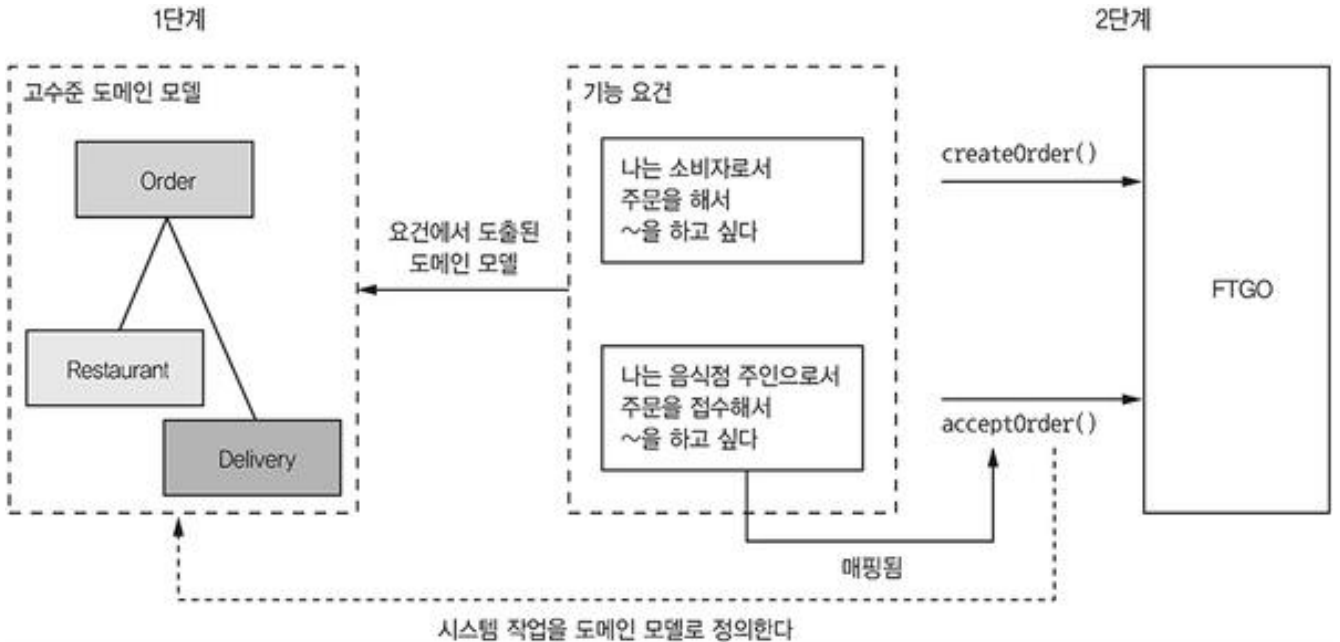
1단계 시스템 작업 식별

2단계 서비스 식별

3단계 서비스 API 및 협동 정의

1단계 시스템 작업 식별

1. 사용자 스토리 작성
2. 도메인 모델 추출
   - 도메인 모델 : 사용자 스토리의 명사 또는 이벤트 스토밍을 통해 도출
3. 시스템 작업 추출
   - 시스템 작업 : 사용자 스토리의 동사에서 도출;
   1개 이상의 도메인 객체와 도메인 객체 사이 관계로 기술



1단계 시스템 작업 식별

1. 사용자 스토리 작성

전제(Given**)**
소비자가 있다.
음식점이 있다.
음식점은 소비자의 주소로 제시간에 음식을 배달할 수 있다.
주문 총액이 음식점의 최소 주문량 조건에 부합한다.

조건(When**)**
소비자가 음식점에 음식을 주문한다.

결과(Then**)**
소비자 신용카드가 승인된다.
주문이 PENDING_ACCEPTANCE 상태로 생성된다.
생성된 주문이 소비자와 연관된다.
생성된 주문이 음식점과 연관된다.

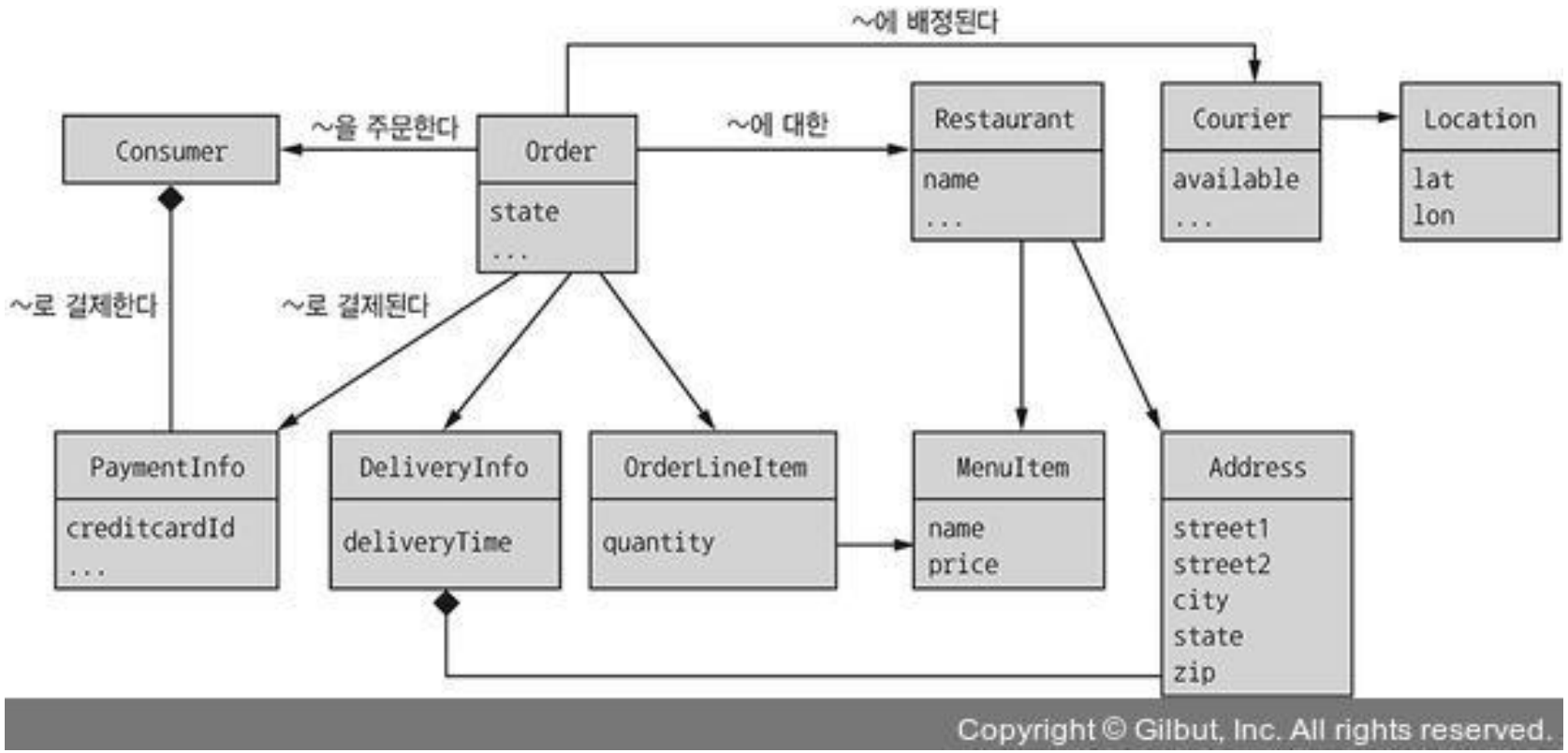
전제(Given**)**
현재 주문은 PENDING_ACCEPTANCE 상태다.
주문 배달 가능한 배달원이 있다.

조건(When**)**
주문을 접수한 음식점은 언제까지 음식을 준비할 수 있다고 약속한다.

결과(Then**)**
주문 상태가 ACCEPTED로 변경된다.
주문의 promiseByTime 값을 음식점이 준비하기로 약속한 시간으로 업데이트 한다.
주문을 배달할 배달원을 배정한다.

1단계 시스템 작업 식별

2. 고수준 도메인 모델 생성



1단계 시스템 작업 식별

3. 시스템 작업 추출

- 커맨드(command, 명령) : 데이터 생성, 수정, 삭제(CUD)

액터	스토리	커맨드	설명
소비자 (Consumer)	주문 생성	createOrder()	주문을 생성한다.
음식점 (Restaurant)	주문 접수	acceptOrder()	음식점에 주문이 접수되었고 주어진 시각까지 음식을 준비하도록 지시한다.
	주문 픽업 준비됨	noteOrderReadyForPickup()	주문한 음식이 픽업 가능함을 알린다.
배달원 (Courier)	위치 업데이트	noteUpdatedLocation()	배달원의 현재 위치를 업데이트한다.
	배달 픽업	noteDeliveryPickedUp()	주문한 음식을 배달원이 픽업했음을 알린다.
	주문 배달됨	noteDeliveryDelivered()	주문한 음식을 배달원이 소비자에게 배달했음을 알린다.

작업	createOrder(소비자 ID, 결제 수단, 배달 주소, 배달 시각, 음식점 ID, 주문 품목)
반환값	orderId, …
선행 조건	• 소비자가 존재하고 주문을 할 수 있다. • 주문 품목은 음식점의 메뉴 항목에 들어 있다. • 배달 주소/시각은 음식점에서 서비스할 수 있다.
후행 조건	• 소비자 신용카드는 주문 금액만큼 승인 처리되었다. • 주문은 PENDING_ACCEPTANCE 상태로 생성되었다.

사용자 시나리오에 따라 **선행/후행 조건** 반영해야 함

1단계 시스템 작업 식별

3. 시스템 작업 추출

사용자 스토리에서 쿼리(R, 조회)도 추출

1. 사용자는 배달 주소 및 시간을 입력합니다.
2. 시스템은 배달 가능한 음식점을 표시합니다.
3. 사용자는 음식점을 고릅니다.
4. 시스템은 메뉴를 표시합니다.
5. 사용자는 원하는 메뉴를 선택한 후 체크아웃합니다.
6. 시스템은 주문을 생성합니다.

- findAvailableRestaurants(deliveryAddress, deliveryTime)
  주어진 장소/시간으로 배달 가능한 음식점 목록을 조회

- findRestaurantMenu(id)
  메뉴 항목 등 음식점 정보를 조회

2단계 서비스 식별(정의)

전략1 **비즈니스 능력 패턴별 분해**

전략2 **하위도메인 패턴별 분해**

2단계 서비스 식별(정의)

전략1 **비즈니스 능력 패턴별 분해**

비즈니스 능력?

비즈니스가 가치를 생산하기 위해 하는 일

예) 온라인 쇼핑몰 - 주문 관리, 재고 관리 등

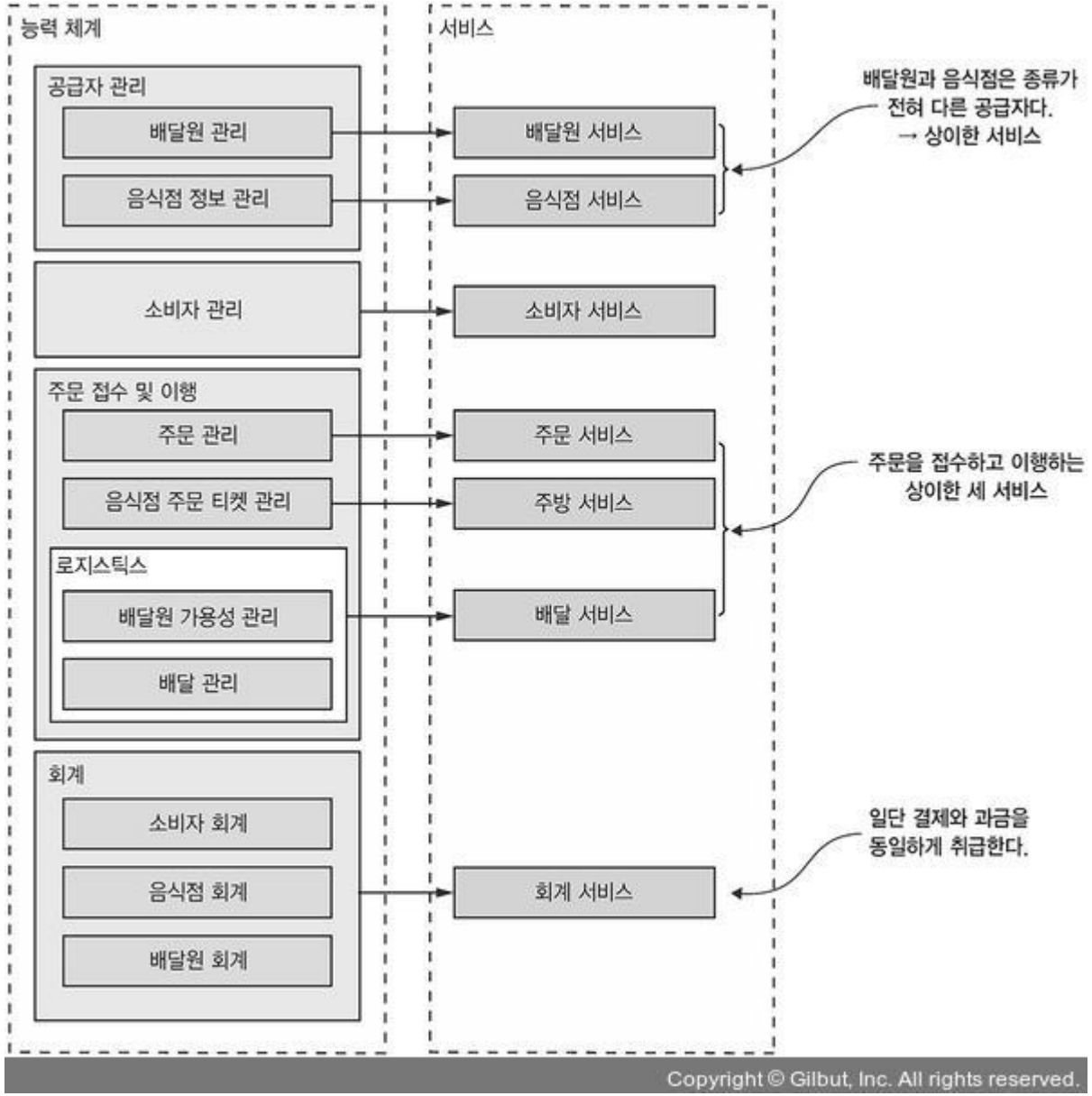
2단계 서비스 식별(정의)

전략1 **비즈니스 능력 패턴별 분해**

- **공급자 관리**
  - 배달원 관리: 배달 정보 관리
  - 음식점 정보 관리: 음식점 메뉴, 위치, 영업 시간, 기타 정보 관리
- **소비자 관리**: 소비자에 관한 정보 관리
- **주문 접수 및 이행**
  - 주문 관리: 소비자가 주문을 생성/관리할 수 있게 합니다.
  - 음식점 주문 관리: 음식점의 주문 준비 상태를 관리
  - 로지스틱스(logistics, 실행 계획)
  - 배달원 가용성 관리: 배달원이 배달 가능한지 실시간 관리
  - 배달 관리: 주문을 소비자에게 배달
- **회계**
  - 소비자 회계: 소비자 과금 관리
  - 음식점 회계: 음식점 지불 관리
  - 배달원 회계: 배달원 지불 관리
- …

조직의 목표, 구조, 비즈니스 프로세스를 분석하여

특정 조직의 비즈니스 능력 계층화하여 식별



능력에 따라 또는 연관된 능력 그룹에 따라 서비스 정의

2단계 서비스 식별(정의)

분해 지침

1 **단일 책임 원칙:** 하나의 책임만 가진 작고 응집된 서비스를 정의

2 **공동 폐쇄 원칙:** 동일한 사유로 변경되는 컴포넌트를 모두 같은 서비스로 묶기

2단계 서비스 식별(정의)

분해 장애물

- 네트워크 지연
- 동기 통신으로 인한 가용성 저하
- 여러 서비스에 걸쳐 데이터 일관성 유지
- 데이터의 일관된 뷰 확보
- 분해를 저해하는 만능 클래스

2단계 서비스 식별(정의)

분해 장애물: 네트워크 지연

해소방안 1 **왕복 1번해서 여러 객체를 한 번에 가져오는 배치 API 구현**

해소방안 2 **값비싼 IPC를 언어 수준의 메서드나 함수 호출로 대체**

2단계 서비스 식별(정의)

분해 장애물: 동기 IPC로 인한 가용성 저하

예) 타 서비스 중 하나라도 불능일 경우, 주문은 생성되지 않는다

비동기 메시징으로 강한 결합도 제거

2단계 서비스 식별(정의)

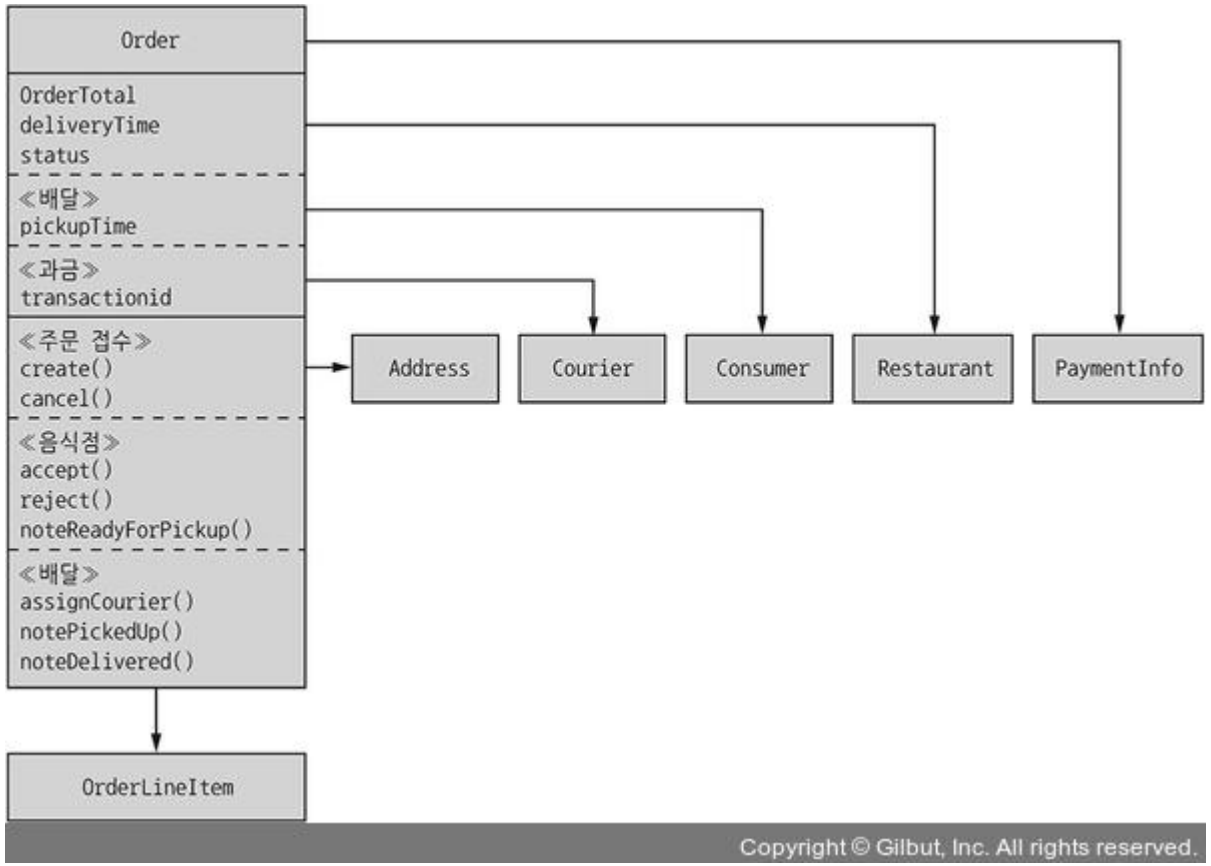
분해 장애물: ~~일관된 데이터 뷰 확보~~

MSA에서 각 서비스의 DB가 일관적이라 해도
전역 범위에서 일관된 데이터 뷰는 확보 불가

실질적으로 MSA 서비스로 운영할 때 크게 문제 안됨

2단계 서비스 식별(정의)

분해 장애물: 만능 클래스는 분해의 걸림돌



DDD를 적용하여

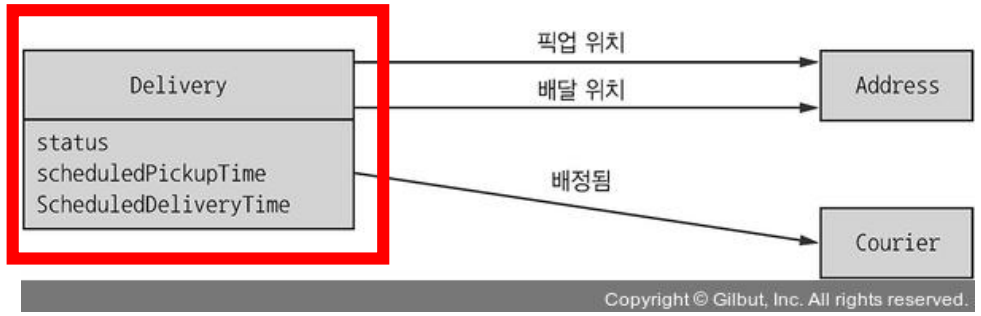
의존 관계가 뒤엉켜 분해를 가로막는 만능 클래스를 제거

각 서비스를 자체 도메인 모델을 갖고 있는 하위 도메인으로 취급

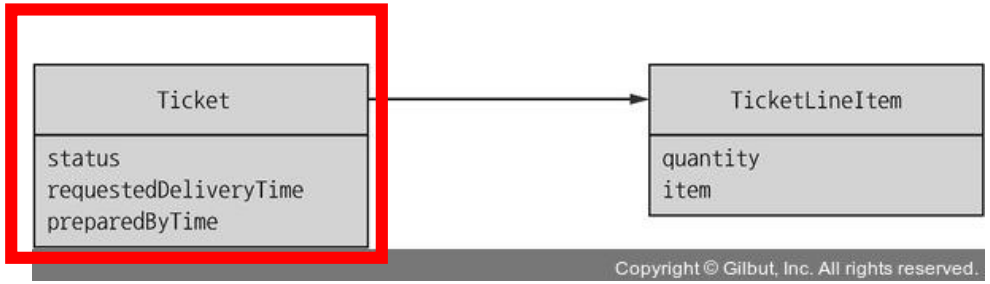
예) 주문과 조금이라도 연관된 서비스는

모두 각자 버전 Order 클래스를 가진 도메인 모델을 따로 두는 것

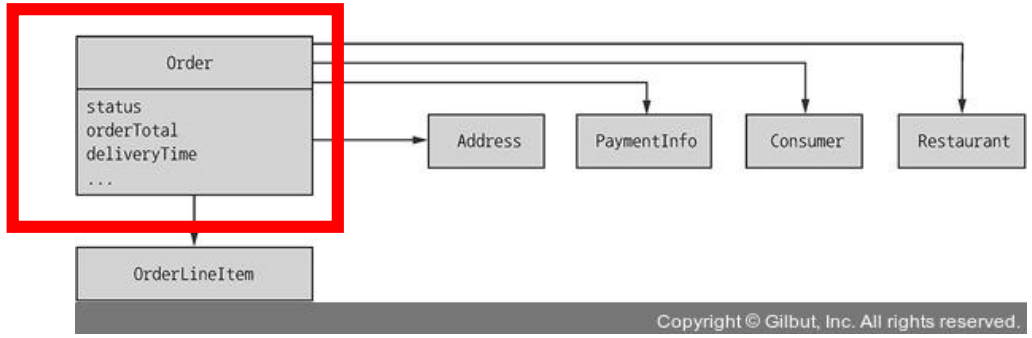
배달 서비스의 Order 뷰



주방 서비스의 Order 뷰



주문 서비스의 Order 뷰



서비스 간 일관성을 위한 노력이 필요

예) 주문 서비스는 소비자 신용카드를 승인 후, 반드시 주방 서비스의 Ticket 생성을 트리거

3단계 서비스 API 및 협동 정의

서비스별 API 정의

서비스 API 구성

1. 작업

    1) 외부 클라이언트 또는 타 서비스가 호출 전용

    2) 서비스 간 협동을 지원하기 위해 타 서비스 호출 전용

2. 이벤트: 주로 타 서비스와 협동하기 위해 발행

    - 이벤트로, 사가를 구현하고 서비스 간 데이터 일관성 유지

    - 이벤트로, CQRS 뷰를 업데이트하고 쿼리를 효과적으로 지원

    - 이벤트로, 애플리케이션이 외부 클라이언트에 알림

3단계 서비스 API 및 협동 정의

서비스별 API 정의

1단계 시스템 작업을 서비스로 배정

2단계 서비스 간 협동 지원에 필요한 API 확정

3단계 서비스 API 및 협동 정의

1단계 시스템 작업을 서비스로 배정

1. 요청 진입점인 서비스 결정
2. 시스템 작업을 서비스로 매핑
   - 어떤 작업이 제공하는 정보가 필요한 서비스에 그 작업을 배정

예) noteUpdatedLocation(): 배달원 위치 업데이트

배달원 서비스에 배정? X

배달원의 위치가 필요한 주체는 **배달 서비스**

서비스	작업
소비자 서비스	createConsumer()
주문 서비스	createOrder()
음식점 서비스	findAvailableRestaurants()
주방 서비스	acceptOrder() noteOrderReadyForPickup()
배달 서비스	noteUpdatedLocation() noteDeliveryPickedUp() noteDeliveryDelivered()

2단계 서비스 간 협동 지원 API 확정

요청을 처리하는 데 필요한 데이터가 여러 서비스에 흩어져 있을 경우,

선행 조건을 확인하고 후행 조건을 충족시키기 위해 특정 서비스를 호출

서비스	작업	협동자
소비자 서비스	verifyConsumerDetails()	-
주문 서비스	createOrder()	소비자 서비스: verifyConsumerDetails() 음식점 서비스: verifyOrderDetails() 주방 서비스: createTicket() 회계 서비스: authorizeCard()
음식점 서비스	findAvailableRestaurants() verifyOrderDetails()	-
주방 서비스	createTicket() acceptOrder() noteOrderReadyForPickup()	배달 서비스: scheduleDelivery()
배달 서비스	scheduleDelivery() noteUpdatedLocation() noteDeliveryPickedUp() noteDeliveryDelivered()	-
회계 서비스	authorizeCard()	-

감사합니다!

질문받겠습니다ㅎㅎ