1) 자바는 어떻게 탄생했는가 / 무엇을 위해 만들어 졌나?
90년대 초반에 만들어짐

C는 왜 만들어졌는가 60년대 만들어짐
IBM 멀틱스 - 기능도 많고 복잠함
펀치 카드 언어 - 포트란(전쟁) 언어

톰슨 / C언어의 아버지들이 복잡함을 느낌
UNIX C 한가지만을 담당할것이다

코직스 호환성이 있냐 없냐
IBM 1-5 까지의 호환성
C : 간단하게 만들어서 명령어들을 조합해서 원하는 일을 한다
간결하고 구현하기 쉽다 범용적으로 쓰임

JAVA : 8-90년대 만들어짐
미국 새로운 것들이 많이 만들어지기 시작함
제임스 고스링
목적 :각 기기들에 프로그램이 들어 갈 것이라 생각하고 만들어진 언어 / 소규모 기기에 들어가서 통신을 위해 만들려했다

인베디드 기기에 들어가기위해 만들어졌다
언어에 탄생 기반과는 다르게 웹 서비스에 많이 쓰임

핸드폰에 채택 되어 자바가 들어감
피쳐폰에 들어가는 어플리케이션(위피 C 자바)

개념자체가 처음에는 없었다
우리는 자바의 일부만 알고 사용하고 있다

소형장비에서부터 서버 프로그램 어플리케이션 다 만들 수 있게 발전해왔다
웹서비스로 많이 성공함

JS - 네스케이프에서 동적인 화면을 만들고 싶다

2) OSGI 설명
이클립스 기반으로 만들어짐
99년
SOA 아키텍쳐를 구현
홈게이트웨이 텔레매틱스 단말 모바일 단말
PNP 플러그 앤 플레이

MS 에서 추진한 그룹
PC에 카드를 곶거나

DLNA

자바 프로젝트에 부합했다

이클립스 마켓 플레이스
의존성 (?)

서로 버전 호환성을 기술

OSGI :20년가다가 버전이 더이상 나오지 않고 이클립스에게 권한을 넘겨버림
기술명, 스펙

유명무실 해지고 이클리스만 남게됨

이클립스에서의 제품명 : 이퀴녹스

이클립스 P2 폴더 : 코어
이클립스 플러그인 폴더 : 플러그인

OSGI 의존성 관리를 위한
일반사람들이 알지는 못함
개발자라면 한번쯤 봐두는 것도 좋다

이클립스가 이퀴녹스 두번째 버전으로 만들어졌다
이퀴녹스OSGI 구현체

OSGI 스펙 구현체 - 이클립스가 OSGI를 구현한 것이 이퀴녹스

JAVA : JCP에서 사양이다
JAVA 웹 뿐만 아니라 어플리케이션도 만들 수 있다

번외 오라클 VS 구글 (소송)
썬마이크로시스템즈

3) vkRest/ vkDB / Maven 설명

sts 같은 경우 단일 프로젝트로서 pom 파일이 다 존재

npm : OSGI 는 아직 안만들어짐
mvn repository : 얘가 먼저

우리 프로젝트)
플러그인들은 어디서 가져왔는가?

네티 프레임워크는 ?

- maven
jar 로 프로젝트가 나오게 설정을 해놓음 (우리는)

vkPrent
한꺼번에 버전 관리를 위해 만듬
디펜던씨 버전을 다 명시해놓음

JAVA 라이브러리 모두 정의해 놓았다

vkRest
NETTY 기반의 REST 구현체를 만들어 놓음

네티... 많은 것들을 가져다가 구현해놓음

스콥 : 메이븐에서 빌드만 할건지/ 빌드하면서 배포할건지/런타임
단일어플리케이션으로 만들어질것이기 때문에 생략 (생략하면 다 들어감)
웹어플리케이션을 만들 때 잘 생각해야함 (어디서 돌릴건지 버전도 맞춰주어야한다 / 직접 설정함)

vkDB
엔티티 (테이블)
ORM -> 하이버네이트
DBCP -> hikari

target 폴더 / classess /

프로젝트 참조를 통해 단일 실행 프로그램으로 되게끔 만들어준다
버전 관리 및 api 동일버전

start UML
공부할 때 상속관계를 그림
시퀀스 다이어그램

vkPrent : java library 모두 정의
  |
  |-- vkRest :NETTY 기반 / Rest 구현체
  |
  |
  |-- vkDB : Entity(테이블) / ORM(하이버네이트) / DBCP(Hikari)

vkRest 동작 (?)
1) 서비스 api 어떻게 path 매칭되는지
2) 실제 기동될 때(동작) 어떻게 되는지

시작
vkREST - 시작점이 없음 (호출이 있어야 동작)
서비스.java 들어가봄

main.java 들어가봄

vkDB - 시작점이 없음 (호출이 있어야 동작)

4) 호출구조

메인 / 빌더 들어감

GOF 개발 패턴 (빌드업 패턴 / 생성패턴)
마지막에 하는 일은 빌더 옵션 준것들을 반환한다
httpsever를 타고 들어가본다

Netty 기반의 구현체는 HttpSever 를 만듦

Netty : 비동기 서버 (비동기 IO) - 코덱을 얹어놓음
Netty 로 비동기 서버를 만들어놓음 (netty 로 http 서버 만들기 검색)

채널 핸들러
파이프라인의 형태

마지막에 http 코덱을 까서 내용을 볼 수 있게 한다

class 파일 하나에다가 다 넣어서 간단한 서버를 만들 수 있다

메인에서 값을 던질 때
setHttpService hashSet 서비스들을 넘긴다

http서비스 플러그인 = 엔드포인트에 들어감

netty = http 서버를 담당

엔드포인트에서 돌면서 어노테이션을 확인

라우터 매쳐를 생성 - 메소드 확인 (crud) 맵핑 - 바인드웨이트(메인) - 핸들러를 구동시키는 것이 바인더 - (바인드웨잇) 서비스가 종료될 때 까지 기다린다 - 보낼때 인코딩 받을때 디코딩

프록시(?)

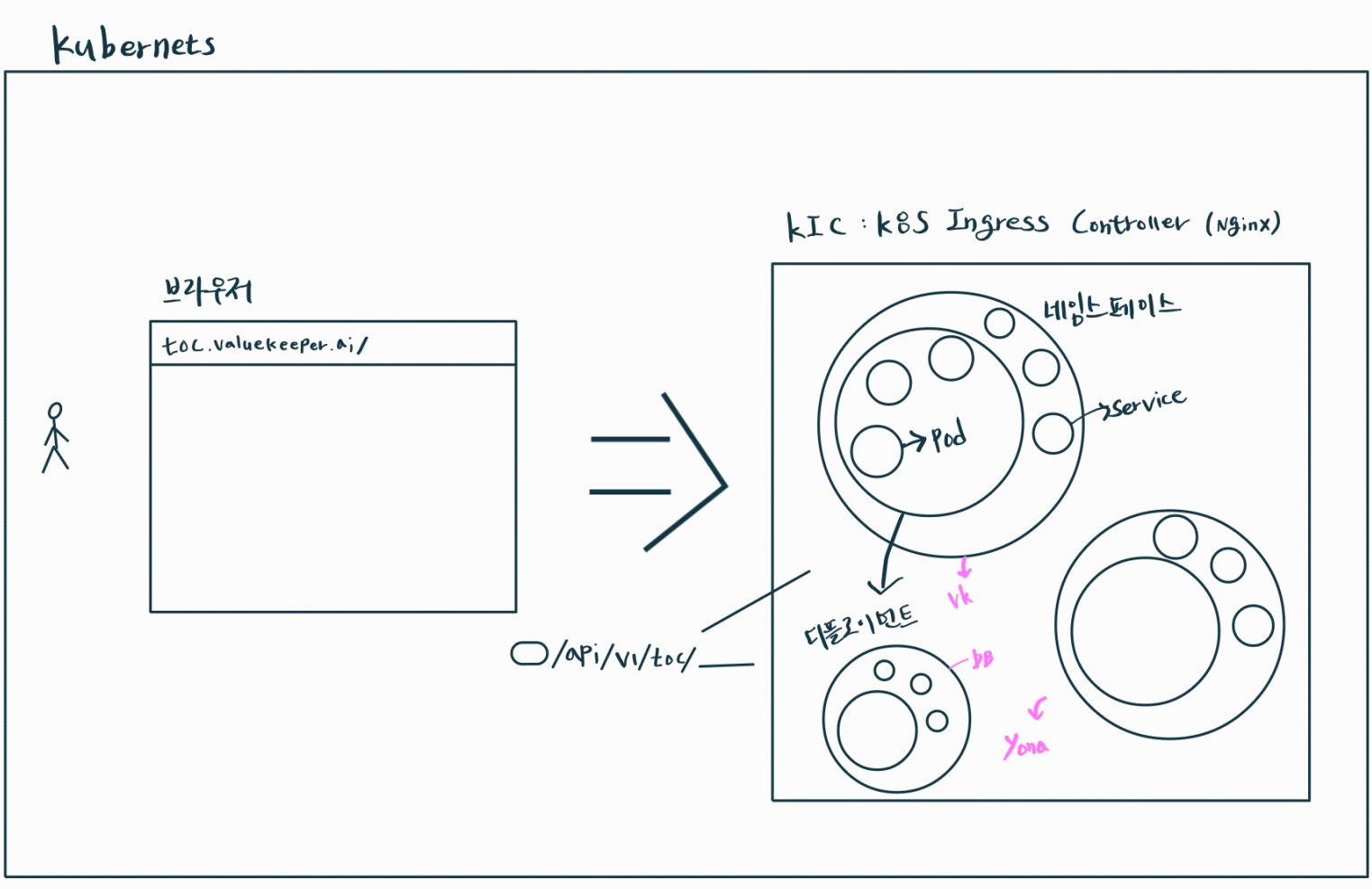
netty가 준 도구로 http 서버를 구축했음

vkDB
겟 세션 펙토리 dv 랑 연결 - 하이버네이트 유틸
싱글톤 펙터는 한번만 생성해서 다시 실행 안되고 호출되면 쭉 사용

로컬과 쿠버네티스의 디비연결 방식이 다름
잘 구분해서 써야된다

하이버네이트 벨리데이터

nginx 기반의 컨트롤러를 세팅하면 http 만 처리된다
우리는 k8s ingress contrller 사용 (nginx 기반)
웹 요청 - ingress - nginx - 어떤 pod으로 갈지 라우팅 시킴 -



namespace : 복잡한 클러스터 환경에서 리소스 관리와 관리를 간소화, 다양한 프로젝트 및 팀 간격에 격리를 유지하는데 도움
service : 서비스에 대한 안정적인 연결을 제공ㅇ하고 외부 노출을 통해 외부 클라이언트가 서비스에 접근 할 수 있도록 한다.
deployment : pod 관리, 라이프사이클 단순화, 안정적인 애플리케이션 업데이트 및 배포를 지원
pod : pod 내에서 하나 이상의 컨테이너를 실행하는 역할하며 관리. 웹 서버와 데이터베이스 서버를 함께 실행하는 경우, 각각의 웹 서버 컨테이너와 데이터베이스 컨테이너를 하나의 pod 내에서 실행하여 물리적인 가까운 위치 및 공유 네트워크 환경을 활용가능

ValuKepper 내 k8s 호출 구조 분석
웹브라우저(toc.valuekeeper.ai) => Service => Deployment =>POD (TOC-FE:NginX +ReactWeb)
웹브라우저(fetch : api/v1/toc) => Service => Deployment =>POD (TOC-BE:NginX +vkApiToc)

FE : 프론트엔드
BE : 백엔드

웹서버 / 웹 어플리케이션 서버 역할 구분