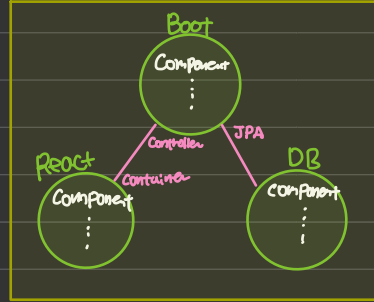
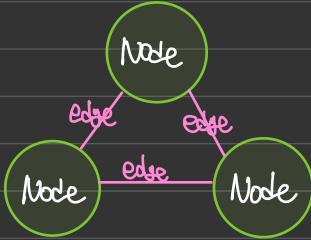


## 그래프 자료구조

Node 와 Edge.

Node: 하나의 객체를 표현 → Component 이다.

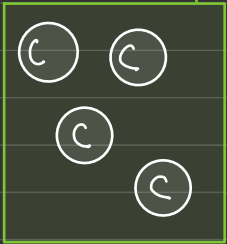
Edge: Node 간의 관계를 이어주는 것.



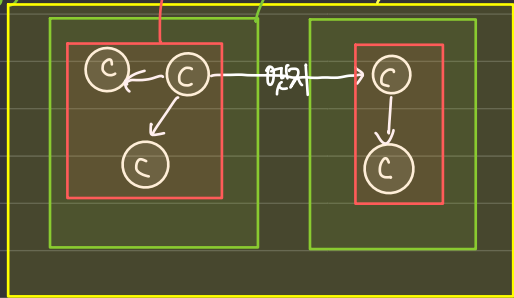
스프링클라우드

## 마이크로 서비스

### \* 모놀리식 서비스



### \* 마이크로



플랫폼 (컴포넌트의 집합)

플랫폼

노드

배포한다.

도커 / 쿠버네티스  
리미트

플랫폼: 컴포넌트의 집합 / 패키지 (패키징) → 웹 컨테이너: 배포를 위해 컴포넌트들은 다른 공간!

노드: 모놀리식의 컴포넌트들을 분리하는 작업은 아키텍처 → 분리를 해주는 것이 Node.js

익스포트: 독립적인 각 컴포넌트들을 연결해주는 것.

React: Axios / Spring: RESTful (G/P/P/D)

배포: 어느 한 곳에 놓고 소비자들이 가져가는 상태

\* Pub-Sub Model. (구독 시스템).

독립적인 소프트웨어 컴포넌트 의 장점.

: 플랫폼 API !!!

↳ 새로운 BM ex) 쿠팡의 100조 주식상장 / 구글어스

⇒ 이것을 만들어 내는 수가 많아 vs 많아? 0 ⇒ Micro / X ⇒ Mono

- 기존 시스템과 새로운 시스템의 통합이 수월.
- > H법적인 배포, 업그레이드가 가능하.


• 여러서버로 SCALE OUT 가능.


Scala는 크기를 따짐

Vector는 카운팅 (length/size)

아무튼 따짐.

\* Scale OUT vs UP (성능향상 방식) \* Scala = Scale

① Out: (수직스케일링)  + ... Count UP. Scala를 Vector로 (Micro)

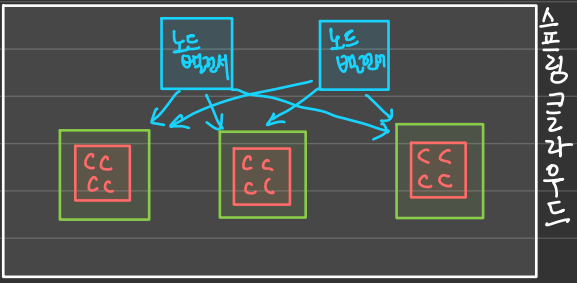
② UP: (수직스케일링)  서버의 고도화 (Mono)

Scala의 크기 ↑

마이크로 서비스: Scale out으로 스케일링을 함.

→ 여러서버에 수동으로 load balancer를 설정하는 방식으로 함.

(단점: 비용 ↑ / 다중기대응.)



마이크로 서비스의 기준

① 아무것도 공유하지 않는 자기자치를 유지, 데이터 서비스의 데이터를 공유하지 않는다

⇒ STATELESS

② 동기처리를 비동기 처리로 바꿔야한다

⇒ ASync : Axios

③ 개별적인 런타임 프레임워크 배포해야한다

⇒ JAVA : Tomcat / react : NPM → NODE(Spring, react)

⇒ ③은 개략도가 관여 X → ①, ②을 고려해서 마이크로 서비스를 만들어야함

\* 특히 ①이 중요 why? ①은 안전코드 프로그래밍 가능

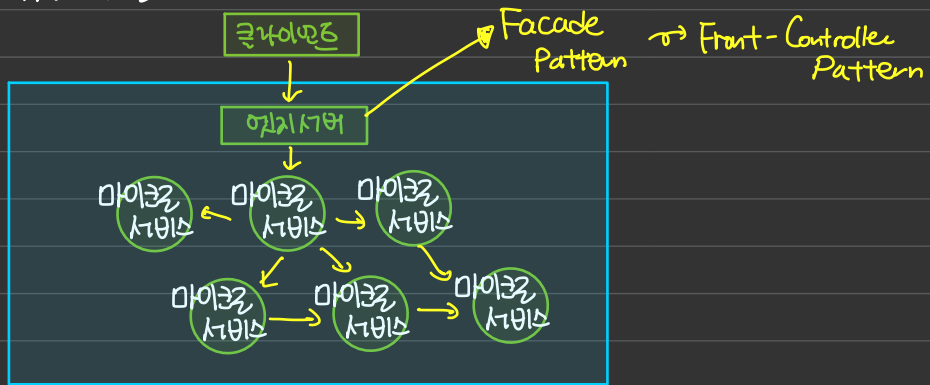
⇒ Debug가 쉽다.

- 마이크로 서비스의 디자인 패턴

(스프링부트 / 스프링클라우드 / 쿠버네티스(도커컴patible) / 리액티브 마이크로 서비스(리액트) )

- 스케일 아웃을 함으로써 로드 밸런서를 통해 연결이 필요하다!

- 연결 : 메시징 서버가 필요



클라이언트 즉 메시징 : 리액티브 라우터 /