다중 컨테이너와 Docker 활용을 통한 MSA DevOps 구현

1. 프로젝트 프로세스
   1. 필요 기능 :
      1. Python과 Node JS로 작성된 poll, result 프론트 엔드 소스 코드
      2. Redis 와 PostgreSQL을 활용한 데이터 전송 및 저장 기능
      3. Java로 작성된 worker 백엔드 소스코드
   2. 클라우드 : digital ocean
   3. Monolithic한 어플리케이션을 Microservice로 분리
      1. 소스코드에서 하나의 Monolithic한 어플리케이션의 서비스단, 비즈니스 로직, 데이터 엑세스 부분을, 즉, 프론트엔드, 백엔드, DB 등의 마이크로서비스들로 나눠 각기 다른 컨테이너에 놓는다.
   4. 도커 활용해 각 Microservice의 컨테이너화
      1. 도커 파일로 빌드하여 이미지로 만들어 컨테이너화한 다음, docker compose 단일 명령을 사용하여 모두 실행하여, 하나의 어플리케이션을 실행한다.
   5. 구체적 프로세스
      1. poll, result, worker 각각의 디렉터리에, 각 소스코드의 언어에 맞게 Dockerfile을 작성한다. ( Dockerfile에 대한 자세한 설명은 각 디렉터리 내 Dockerfile의 주석 참고 )
      2. Sources 디렉터리에서 “Docker compose up” 후, <http://localhost:5000> 로 접속하면, poll 투표 화면이 보인다.
      3. 투표를 하면, 투표 데이터가 캐시로 Redis에 저장된다.
      4. Worker 백엔드는 이것을 Postgres DB에 vote 데이터테이블을 만든 후 저장한다.
      5. <http://localhost:5001> 로 redirect 되어 투표가 집계된 후의 result 결과 화면이 보인다.
2. Microservice 아키텍쳐의 개념 및 장점
   1. 개념 : 개별서비스들이 단독적으로 어느 때든지 배포가 가능한 형태
   2. 구체적으로, 하나의 어플리케이션을 DB, 프론트, 백엔드 등의 작은 어플리케이션들로 나누어 개발하는 것.
   3. 장점 :
      1. 분업을 통한 개발의 효율성 증가.
      2. 회복력 : 장애가 발생했을 때 최대한 빨리 복구될 수 있는 능력
      3. 오류의 아이솔레이션 : 한 어플리케이션에서 발생한 오류가 다른 어플리케이션으로 전파되지 않도록 하는 것.
3. 데브옵스의 개념 및 장점
   1. CI/CD를 최적화하는 것
   2. 디자인 후 (코딩, 테스트, 배포) 이터레이션을 반복하는 것.
      1. 배포를 가장 마지막에 하는 워터폴, 애자일 방식과는 다르게 배포 시 오류가 발생하여 프로젝트가 실패할 위험이 줄어든다
   3. 마이크로서비스 빌딩을 CI/CD 파이프라인을 통해 플랫폼에 자동화.
      1. 환경 정비, 테스트 및 빌드 등에 코딩과 비견되는 시간을 사용해야 한다. 개발자의 비효율적인 작업을 야기한다.