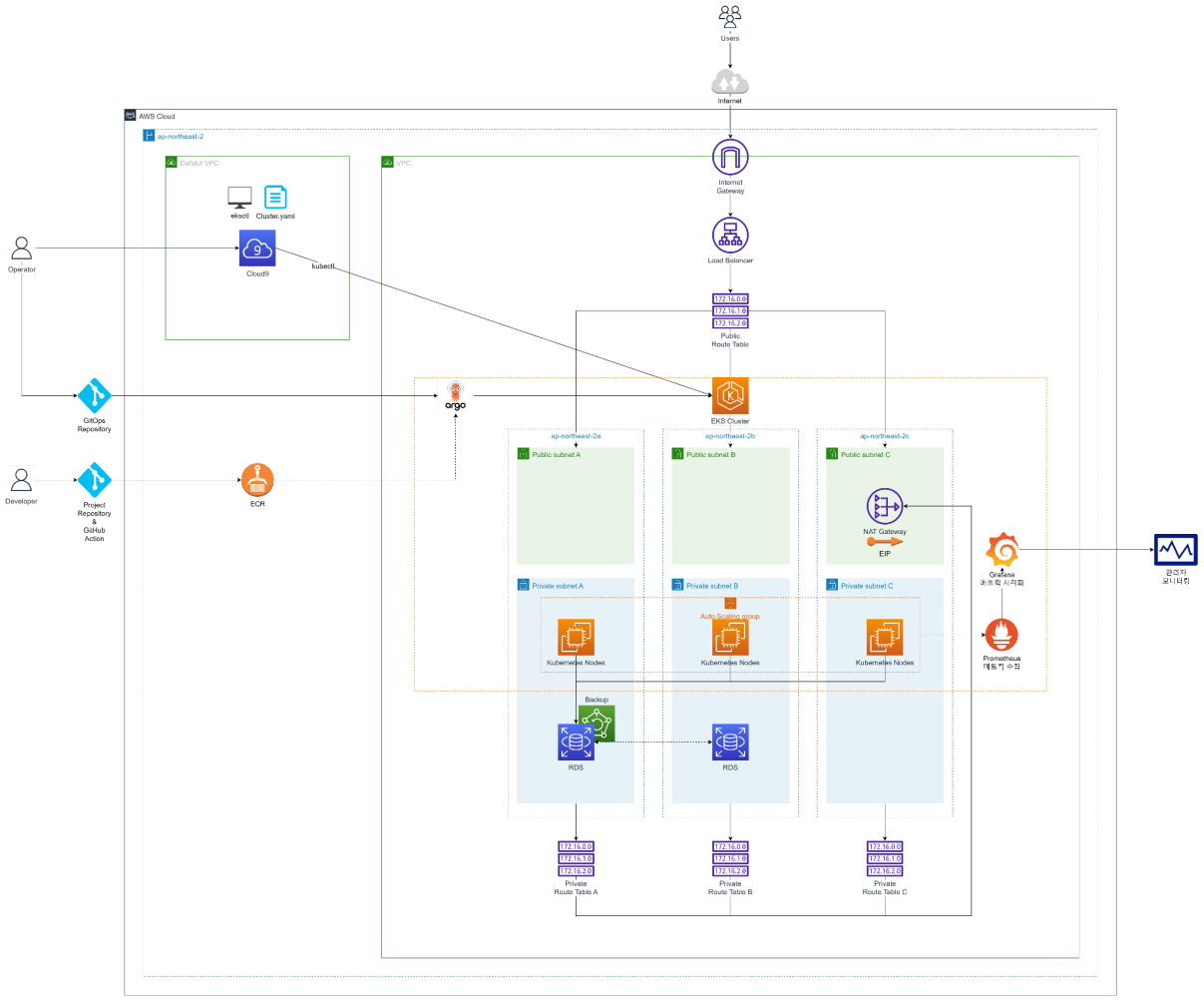
`

구름 **KDT** 쿠버네티스 전문가 양성과정**_6**기 프로젝트 **-** 설계서			
팀 명	1조	일 자	2022년 11 월 23 일
주 제 명	도서정보 인프라 구축		

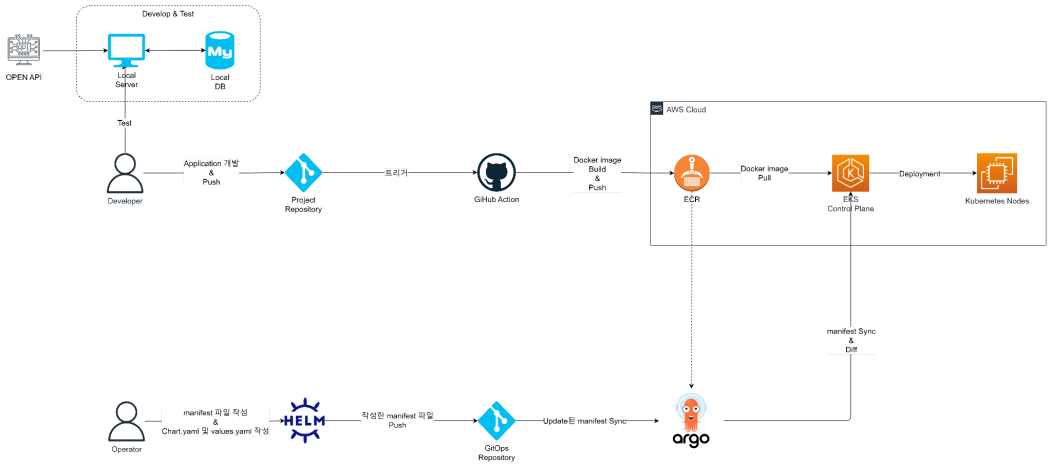
CONTENTS

1. 전체 **System** 구성도

1.1. 전체 **System** 구성도



1.2. 개발/배포 **System Pipeline**



2. 네트워크 설계

2.1. VPC - 2개

2.2. 서브넷 설정

- 개발환경이 존재하는 Default vpc 내에 public 서브넷(개발자 서브넷)을 생성한다.
- 실제 서비스하는 워커노드들이 있는 eks vpc 안에는 외부(public), 내부(private)서버로 구성되어있다.
- EIB

2.3. 게이트웨이, 라우팅 설정

2.3.1. 설정 전 참고사항

- ELB 사용
- https://docs.aws.amazon.com/ko_kr/elasticloadbalancing/latest/gateway/getting-started.html
- 사용자들이 게이트웨이 통해서 퍼블릭라우팅 테이블로 연결되어 각각의 인스턴스와 연결된다.
- Private Route Table에서 두번째 타겟으로 NAT인스턴스를 지정해준다.

2.3.2. 설정

- default vpc : 172.31.0.0/16
  - eks vpc와 Eks control plane 통해 연결 되어있음
  - cloud9 bastion 서버 인스턴스가 Public subnet에 생성되어있음
- eks vpc : 192.168.0.0/16
  - 유저들은 게이트웨이를 통해 ELB에 연결되어 실제 서비스에 연결
  - private 서브넷A에 RDS가 생성되어있음
  - 각각의 Private 서브넷에서 외부로 통신할 때는 Private 라우팅테이블로, 이후 Public 라우팅테이블, 인터넷 게이트웨이 통해서 통신한다
  - Subnet

- 3곳의 AZ (a, b, c)에 public, private subnet은 a, b, c를 생성
- private subnet들에 auto scaling group을 생성,
- 각 subnet의 cider는 /19이다
    - AZ-a의 public subnet (192.168.0.0/19) private subnet (192.168.160.0/19)

- NAT gateway
    - private subnet에서 출발하는 통신을 위해 구성
    - 각각의 public subnet에 개별로 존재한다.
- NAT IP : IP바뀌지 않게 고정하는 용도, 하나의 공인 IP 주소를 사용하여 인터넷에 접속하기 위함.

3. 서버 설계**(EKS)**

3.1. **AWS** 환경에서 구축**,** 서비스**,** 배포를 진행

3.2. 코드 예시

- https://github.com/Goorm-Project-Aladin/infra/blob/main/aladinEKS.yaml
- yaml 파일 실행시 구성되는 과정에 대해서 설명 필요

3.3. 사양

- Cloud9 인스턴스(t3.medium)
    - 2 Core CPU
    - 4 GiB Memory
    - 30 GiB Volume
- Kubernetes Node(m5.large)
    - 2 Core CPU
    - 8 GiB Memory
    - 10 GiB Volume

3.4. 보안

- 3306, 8080 포트 - DB를 관리
- 22 포트 - SSH 접속속

3.5. **Node**에 배포될 **Pod** 목록

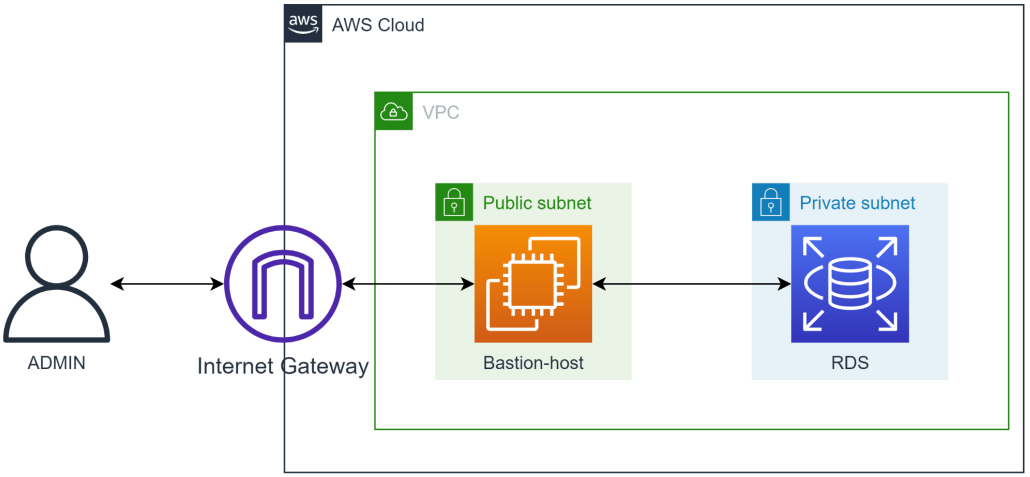
- Kubernetes 기본 pod

- Prometheus
- Grafana
- ArgoCD
- Project Pod(OpenJDK, Spring)

3.6. Region

- 서울 Region을 사용 (ap-northeast-2)
- https://aws.amazon.com/ko/premiumsupport/knowledge-center/public-load-balancer-private-ec2/ ~ 로드벨런서 private 연결하기(public에 연결)
- https://docs.aws.amazon.com/ko_kr/AWSCloudFormation/latest/UserGuide/quickref-rds.html ~ RDS yaml 파일

4. DB 설계

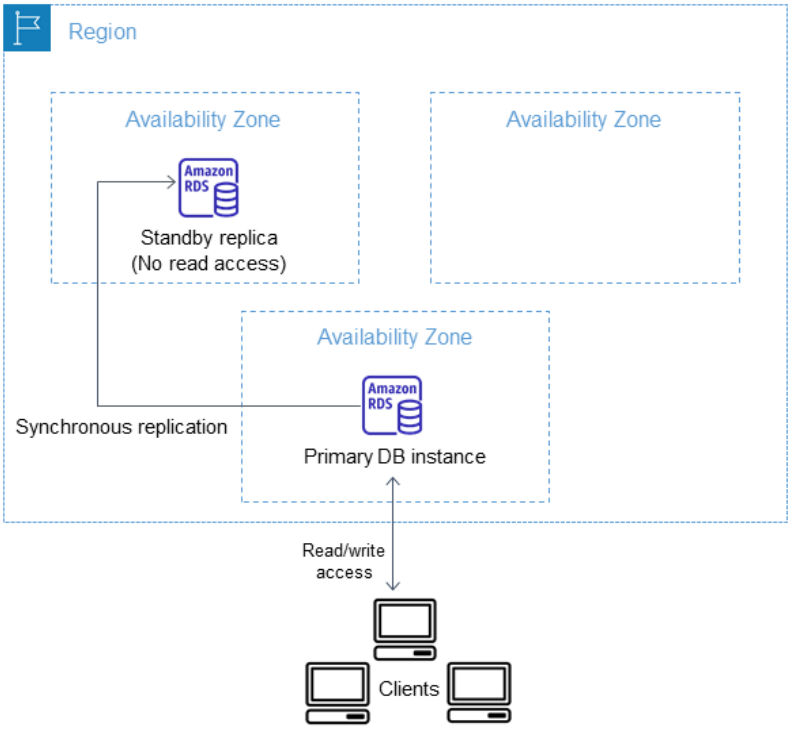


4.1. 인스턴스(EC2)

4.1.1. 인스턴스 정보

- 용도 - RDS 연결할 Bastion host
- 이름 및 태그 - RDS-BASTION-SERVER
- OS - Ubuntu Server 20.04LTS SSD Volume Type
- 인스턴스 유형 - t2.micro
- 키 페어(로그인) 사용
- 네트워크 설정 - VPC, public subnet, public IP 자동 할당
- 방화벽 - SSH(22), MYSQL(3306) 허용
- 스토리지 - 8GIB, GP3

4.2. AWS RDS



4.2.1. RDS 정보

- DB - MySQL8.0.28 MySQL Community
- 템플릿 - 프로덕션
- 가용성 및 내구성 - 다중 AZ DB 인스턴스
- DB 인스턴스 식별자(미정) - BOOK_DB
- 마스터 아이디 / 암호
- 인스턴스 구성 - 버스터블 클래스(db.t3.micro)
- 스토리지(default)
    - 프로비저닝된 IOPS SSD(io1)
    - 할당된 스토리지 - 400
    - 프로비저닝된 IOPS - 3000
    - 스토리지 자동 조정 활성화
    - 최대 스토리지 임계값 - 1000
- 연결
    - EC2 컴퓨팅 리소스에 연결 - RDS-BASTION-SERVER
    - VPC 보안 그룹 - SSH(22), MYSQL(3306)
- 데이터베이스 인증 - 암호 및 IAM 데이터베이스 인증
- 모니터링(default)

- 추가 구성(default)
- 백업
    - 자동 백업 활성화
    - 백업 보존 기간
        - 7일(default)
    - 백업 기간
        - 시작시간 02 : 00
        - 기간 1시간

4.3. DB Table

4.3.1. ERD

5. 보안 설계

5.1. EC2 보안

5.1.1. Control Plane

프로토콜	방향	포트 범위	용도	사용 주체
TCP	인바운드	6443	쿠버네티스 API 서버	All
TCP	인바운드	2379-2380	etcd 서버 클라이언트 API	kube-apiserver, etcd
TCP	인바운드	10250	kubelet API	Self, Control Plane
TCP	인바운드	10259	kube-scheduler	Self
TCP	인바운드	10257	kube-controller-manager	Self

5.1.2. Worker Node

프로토콜	방향	포트 범위	용도	사용 주체
TCP	인바운드	10250	kubelet API	Self, Control Plane
TCP	인바운드	30000-3276	NodePort	All

		7	서비스	

5.1.3. 개발자

프로토콜	방향	포트 범위	용도	사용 주체
TCP	인바운드	22	SSH	

5.1.4. DataBase

프로토콜	방향	포트 범위	용도	사용 주체
TCP	인바운드	22, 3306	SSH, MYSQL	

5.1.5. Bastion-host

프로토콜	방향	포트 범위	용도	사용 주체
TCP	인바운드	22	SSH	

5.2. DB 보안

5.2.1. 접근 통제

- 데이터베이스의 계정 중 인가되지 않은 계정(업무에 사용하지 않은 불필요 계정) 삭제
- 일반 사용자에 대한 데이터베이스 공통 계정 사용 금지

5.2.2. 데이터 암호화

- 사용자 정보(비밀번호) SHA-256이상 암호화
- 사용자 비밀번호 복잡도 설정(대문자,소문자, 숫자 등 조합)
- 개인정보, 민감정보에 대한 식별 및 관리

- 운영DB 주요 정보 존재 시 비식별(마스킹) 관리

6. 백업 설계

6.1. 데이터 백업 방식

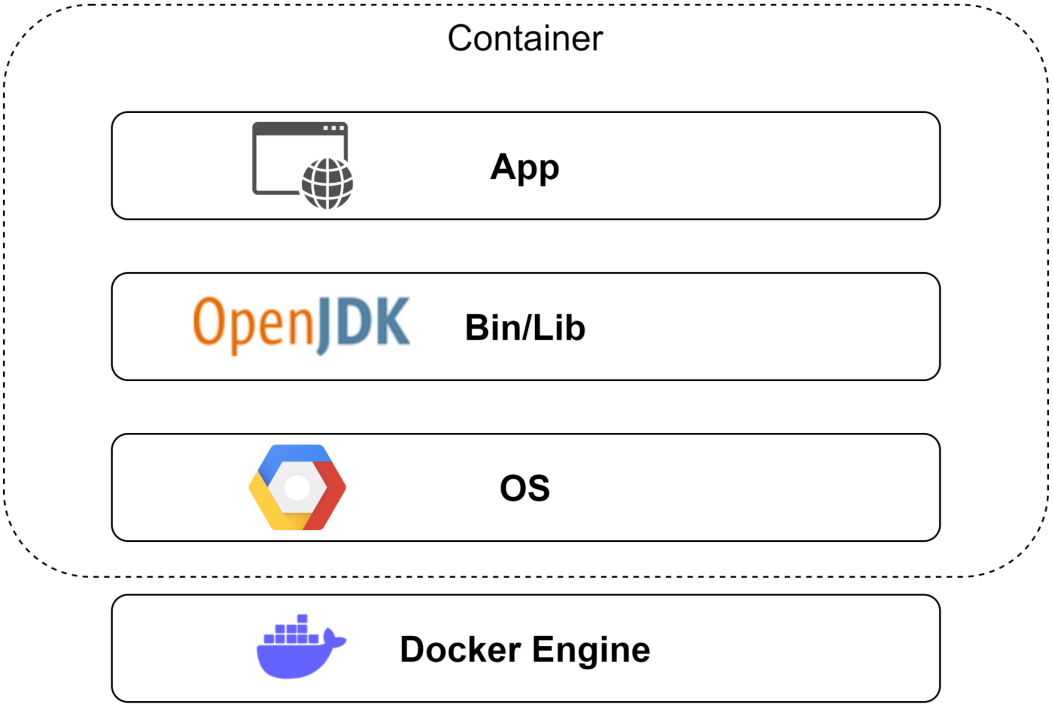
- RDS 자동 스냅샷 생성 이용
- 매일 자동 백업: 오전 02 ~ 05시 사이에 백업 1회(1시간 소요)
- 백업 보존 기간은 7일
- 자세한 사항은 위의 DB설정에 첨부

6.2. 백업 데이터 보호 여부

- 백업 데이터를 이동할 때는 암호화 통신이 이뤄져야 함
- 데이터가 보관되는 백업 스토리지는 외부 IP가 접근할 수 없도록 사설 네트워크로 구성

7. Container 설계

7.1. 설계 구상도



7.2. 명세서

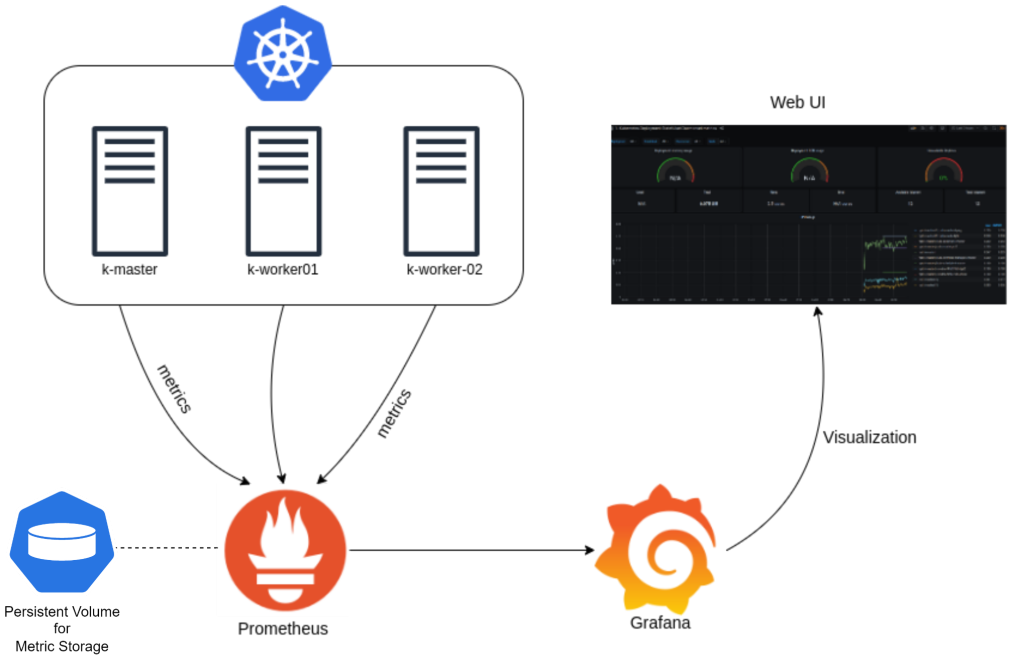
```
# Dockerfile

# Builder
FROM openjdk:11.0-jdk AS builder
LABEL description="Java Application builder"
RUN apt install git
RUN git clone https://github.com/Goorm-Project-Aladin/web.git
WORKDIR web
RUN chmod 700 gradlew
RUN ./gradlew clean build

# Running
FROM gcr.io/distroless/java:11
LABEL description="Java Application run image"
ARG JAR_FILE=/build/libs/web-0.0.1-SNAPSSHOT.jar
COPY --from=builder web/${JAR_FILE} /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

- Multi-stage build
    - 애플리케이션을 빌드할 컨테이너 환경을 생성해 그 안에서 빌드
    - 호스트에 빌드 도구를 설치하지 않고 깔끔하게 빌드하기 위해 사용
- OpenJDK
    - Spring Boot 프로젝트를 실행하기 위한 Runtime 환경
- distroless
    - 구글에서 관리하는 베이스 이미지
    - Small Size OS: 많은 수의 컨테이너를 배포할 것에 대비
    - Secure: 기본적으로 설치된 프로그램이 적어 취약점이 발생할 포인트가 적음

8. 모니터링 설계



8.1. Prometheus

- Prometheus: 쿠버네티스 모니터링 시스템
  - Target System: 수집을 하려는 대상
  - Exporter: 타겟 시스템에서 메트릭을 읽어서 프로메테우스가 풀링(pulling)을 할 수 있도록 함, 단순한 HTTP GET 형식
  - pulling: 프로메테우스가 주기적으로 Exporter로부터 메트릭을 읽어와서 수집
  - Service Discovery: 현재 구동중인 서비스들의 목록과 IP주소를 쿠버네티스 내부에 등록
  - Retrieval: 서비스 디스커버리 시스템으로 부터 모니터링 대상 목록을 받아오고, Exporter로 부터 주기적으로 그 대상으로 부터 메트릭을 수집하는 모듈
  - 저장: 프로메테우스 내의 메모리와 로컬 디스크에 저장
  - 출력(Serving): PromQL을 이용해 Grafana와 연동해 출력