포팅 메뉴얼

포팅 매뉴얼
- 개발 환경
  - FRONTEND
  - BACKEND
- 환경 구성 - EC2 Docker 기반
  - EC2
  - Jenkins
  - React Server
  - Spring Boot Servers
  - Nginx
  - Logging System - EFK
  - Micrometer & Zipkin : Tracing Tx
- 환경 구성 - Kubernetes 환경
  - Google Cloud Platform
  - Kubernetes
  - k8s Autoscaling
  - Grafana & Prometheus
- 변수 및 설정 파일
  - Spring yml 파일 설정
  - Secret Files & GIT Ignored
  - k8s 수동 배포
- 사용 포트

포팅 매뉴얼

개발 환경

FRONTEND

- Framework - Node 20.11.1

    - React 18.2.0
- Build tool - Vite 5.1.4
    - PWA 0.19.2
- State management - Zustand 4.5.2
- Language - TypeScript 5.2.2
- HTTP client - axios 1.6.7
- Style - Tailwind, Styled Components
- IDE - VS code
- 문서화 라이브러리 - StoryBook 7.6.17

BACKEND

- JVM : Oracle Open JDK version 17.0.9
- 웹 서버 : nginx 1.18.0 (ubuntu)
- WAS : tomcat 3.2.1
- IDE : Intellij 2023.3.2
- Jenkins
- Nginx
- Docker
- SpringBoot 3.2.3
- Mysql 8.0.36
- Spring Cloud Netflix Eureka 4.1.0
- Spring Cloud Netflix Gateway 4.1.0
- Spring Cloud OpenFeign 4.1.0

환경 구성 - EC2 Docker 기반

EC2

- `ssh -i K10C102T.pem ubuntu@k10c102.p.ssafy.io`

Jenkins

- Docker 설치
- Jenkins 설치

```
#설치해줄 디렉토리 이동 및 생성
cd /home/ubuntu && mkdir jenkins-data

# 방화벽 세팅
sudo ufw allow *8080*/tcp
sudo ufw reload
sudo ufw status

#jenkins:2.444 이미지 기반 컨테이너 실행
sudo docker run -d -p 8080:8080 -v /home/ubuntu/jenkins-dat
a:/var/jenkins_home --name jenkins jenkins/jenkins:2.444

sudo docker logs jenkins

sudo docker stop jenkins
sudo docker ps -a
```

- Jenkins 환경 설정

```
cd /home/ubuntu/jenkins-data

    mkdir update-center-rootCAs

    wget <https://cdn.jsdelivr.net/gh/lework/jenkins-update
-center/rootCA/update-center.crt> -O ./update-center-rootCA
s/update-center.crt

    sudo sed -i 's#<https://updates.jenkins.io/update->
    center.json#<https://raw.githubusercontent.com/lework/j
enkins-update->
    center/master/updates/tencent/update-center.json#' ./hu
dson.model.UpdateCenter.xml
```

```
sudo docker restart jenkins
```

React Server

다음은 B2C 서비스, ddib을 실행하기 위한 설정이다.

- npm install

```
    # npm을 install 하기 위해서는 노드 버전 관리자인 nvm을 설치해야 한다, 여러 nodejs의 설치 및 전환 가능
    curl -o- <https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.sh> | bash

    # nvm 활성화
    . ~/.nvm/nvm.sh

    # 최신 lts 버전 nodejs 설치

    nvm install --lts
```

- FE Dockerfile

```
# 가져올 이미지를 정의
FROM node:20

# 경로 설정하기
WORKDIR /app

# package.json과 package-lock.json 워킹 디렉토리에 복사 (.은 설정한 워킹 디렉토리를 뜻함)
COPY package.json package-lock.json ./

# 명령어 실행 (의존성 설치)
RUN npm install

# 현재 디렉토리의 모든 파일을 도커 컨테이너의 워킹 디렉토리에 복사한다.
COPY . .
```

```
# 애플리케이션을 빌드한다.
RUN npm run build

# 3000번 포트 노출
EXPOSE 3000

# 빌드된 애플리케이션을 실행
CMD ["npm", "run", "start"]

# 그리고 Dockerfile로 docker 이미지를 빌드해야한다.
# $ docker build .
```

- FE JenkinsFile

```
pipeline {
    agent any
    environment {
        DOCKER_IMAGE_NAME = 'kimyusan/ddib_front'
        DOCKERFILE_PATH = './FrontEnd/DDIB/Dockerfile'
        CONTAINER_NAME = 'ddib_front'
        REGISTRY_CREDENTIAL = 'dockerhub_IdPwd'
        DOCKER_IMAGE = ''
        DOCKER_IMAGE_TAG = 'latest'



    }

    stages {
        //프로젝트 클론
        stage('GitLab Clone') {
            steps {
                git branch : 'dev-front', credentialsId: 'j
enkins', url: 'https://lab.ssafy.com/s10-final/S10P31C102.g
it'
            }
        }

        //Dockerfile로 생성된 빌드 파일로 도커 이미지 생성
```

```
        stage('Docker Build Image') {
            steps {
                dir('./FrontEnd/DDIB') {
                    script {
                        DOCKER_IMAGE = docker.build("${DOCK
ER_IMAGE_NAME}:${DOCKER_IMAGE_TAG}", "-f Dockerfile .")
                    }
                }
            }
        }

        //이미지 도커 허브에 올리기
        stage('Push Image to DockerHub') {
            steps {
                script {
                    docker.withRegistry('', REGISTRY_CREDEN
TIAL) {
                        DOCKER_IMAGE.push()
                    }
                }
            }
        }

        stage('Delete Previous frontend Docker Container'){
            steps {
                script {
                    // 컨테이너가 실행중이 아니거나 중지되어 있는 경
우 아무런 동작하지 않고 넘어가도록
                    sh "docker stop ${CONTAINER_NAME} || tr
ue"
                }
            }
        }

        stage('Prune Docker Object'){
            steps {
                echo '##### delete stopped containers, netw
orks, volumes, images, cache... #####'
```

```
                script {
                    sh "docker system prune --volumes -f"
                }
            }
        }

        stage('Pull from DockerHub') {
            steps {
                script {
                    sh 'docker pull ${DOCKER_IMAGE_NAME}'
                }
            }
        }

        stage('Run Docker Container') {
            steps {
                script {
                    sh 'docker run -d --name ${CONTAINER_NA
ME} -p 3000:3000 ${DOCKER_IMAGE_NAME}'
                }
            }
        }

    }
}
```

Spring Boot Servers

- DockerFile

```
# 빌드 관련
FROM openjdk:17-alpine

CMD ["./gradlew", "clean", "build"]

VOLUME /tmp

# 만들어진 jar파일 복사
```

```
COPY build/libs/*.jar user.jar

# 실행할 명령어
ENTRYPOINT ["java", "-jar", "user.jar","--spring.profiles.a
ctive=prod"]
```

- JenkinsFile

```
pipeline {
    agent any
    // tools {
    //     gradle 'Gradle 8.7'
    // }
    environment {
        DOCKER_IMAGE_NAME = 'kimyusan/ddib_product'
        DOCKERFILE_PATH = './product/Dockerfile'
        CONTAINER_NAME = 'ddib_product'
        REGISTRY_CREDENTIAL = 'dockerhub_IdPwd'
        DOCKER_IMAGE = ''
        DOCKER_IMAGE_TAG = 'latest'
    }
    stages {
        stage('GitLab Clone') {
            steps {
                git branch : 'dev-product', credentialsId:
'jenkins', url: 'https://lab.ssafy.com/s10-final/S10P31C10
2.git'
            }
        }
        stage('Add Env') {
            steps {
                dir('./product') {
                    withCredentials([file(credentialsId: 'k
ey', variable: 'key')]) {
                        sh 'chmod -R a=rwx src/main/resourc
es'
                        sh 'cp ${key} src/main/resources/ap
```

```
plication-key.yml'
                    }
                }
            }
        }
        stage('Gradle Build') {
            steps {
                echo 'Building..'
                dir('./product') {
                    sh 'chmod +x gradlew'
                    sh './gradlew clean bootjar'
                }
            }
        }
        stage('Docker Build Image') {
            steps {
                dir('./product') {
                    script {
                        DOCKER_IMAGE = docker.build("${DOCK
ER_IMAGE_NAME}:${DOCKER_IMAGE_TAG}", "-f Dockerfile .")
                    }
                }
            }
        }
        stage('Push Image to DockerHub') {
            steps {
                script {
                    docker.withRegistry('', REGISTRY_CREDEN
TIAL) {
                        DOCKER_IMAGE.push()
                    }
                }
            }
        }
        stage('Delete Previous back Docker Container'){
            steps {
                script {
                    // 컨테이너가 실행중이 아니거나 중지되어 있는 경
```

```
우 아무런 동작하지 않고 넘어가도록
                    sh "docker stop ${CONTAINER_NAME} || tr
ue"

//                    def exitedContainers = sh(script: "do
cker ps --filter status=exited -q", returnStdout: true).tri
m()
//                    if (exitedContainers) {
//                        sh "docker rm ${exitedContainer
s}"
//                    } else {
//                        echo "No exited containers to rem
ove."
//                    }
                }
            }
        }

        stage('Prune Docker Object'){
            steps {
                echo '##### delete stopped containers, netw
orks, volumes, images, cache... #####'
                script {
//                    sh "docker image prune -f"
                    sh "docker system prune --volumes -f"
                }
            }
        }

        stage('Pull from DockerHub') {
            steps {
                script {
                    sh 'docker pull ${DOCKER_IMAGE_NAME}'
                }
            }
        }

        stage('Run Docker Container') {
```

```
            steps {
                script {
                    sh 'docker run -d --name ${CONTAINER_NA
ME} -p 8082:8082 ${DOCKER_IMAGE_NAME}'
                }
            }
        }
    }
}
```

Nginx

- nginx.conf 설정

```
server {
    listen 80;

    # 수정
    server_name j10c201.p.ssafy.io;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }

    # 이 부분 추가
    location /.well-known/acme-challenge/ {
        allow all;
        root /var/www/certbot;
    }

    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

- 리버스 프록시 설정

```
server {
    listen 80;
    server_name k10c102.p.ssafy.io;
#    return 301 https://$host$request_uri;
}
server {
    # listen [::]:443 ssl ipv6only=on;
      listen 443 ssl ;
            server_name k10c102.p.ssafy.io;
            ssl_certificate /etc/letsencrypt/live/p.ssaf
y.io/fullchain.pem;
            ssl_certificate_key /etc/letsencrypt/live/p.
ssafy.io/privkey.pem;

        ssl_prefer_server_ciphers on;


    location /jenkins {
        proxy_pass http://k10c102.p.ssafy.io:9999;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_fo
rwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_ssl_server_name on;
 }

   location / {
        proxy_pass http://k10c102.p.ssafy.io:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
       proxy_set_header X-Forwarded-For $proxy_add_x_for
warded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_ssl_server_name on;
 }

       location /api/ {
```

```
        proxy_pass http://k10c102.p.ssafy.io:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_fo
rwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_ssl_server_name on;
 }
  location /login/oauth2/code/dibb {
        proxy_pass http://k10c102.p.ssafy.io:8081;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_fo
rwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_ssl_server_name on;
 }

  location /login/oauth2/code/bidd {
        proxy_pass http://k10c102.p.ssafy.io:8085;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_fo
rwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_ssl_server_name on;
 }


  location /bidd {
        proxy_pass http://k10c102.p.ssafy.io:3001;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_fo
rwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_ssl_server_name on;
 }
```

```
  location /api/seller {
        proxy_pass http://k10c102.p.ssafy.io:8085;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_fo
rwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_ssl_server_name on;
 }



}
```

Logging System - EFK

MSA 기반에서는 모든 로그를 각각의 서버에 접속하여 확인해야 하는 어려움이 있기에, 이를 한 곳에서 모든 서버에서 실행된 로그를 확인할 수 있도록 EFK 로깅 시스템을 구축한다.

Docker Compose 를 통해 Elastic Search, Fluentd, Kibana 를 구동하고, logback 설정을 통해 fluentd 로 로그 수집을 하며, Elastic Search 에 저장한 후 Kibana를 통해 로그를 시각화 한다.

1. Docker Compose 파일 작성 → `https://github.com/gnoesnooj/efk` 에 작성 되어 있음.
2. EC2 에 해당 docker compose 파일 저장 → 여기선 github 에 저장 후, clone 하는 방식 선택

`https://github.com/gnoesnooj/efk.git`

3. Spring Boot Log Back 설정 후 Fluentd 로 전송 - `logback.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <property name="CONSOLE_LOG_PATTERN"
```

```
                value="%d{yyyy-MM-dd HH:mm:ss.SSS} %highlight(%

    <appender name="CONSOLE" class="ch.qos.logback.core.Conso
        <encoder class="ch.qos.logback.classic.encoder.Patter
            <pattern>${CONSOLE_LOG_PATTERN}</pattern>
        </encoder>
    </appender>

    <appender name="FLUENT_TEXT" class="ch.qos.logback.more.a
        <filter class="ch.qos.logback.classic.filter.LevelFil
            <!-- 에러로그를 설정하고 로그의 레벨이 맞으면 onMatch, 아
            <level>info</level>
        </filter>
        <tag>applog</tag>
        <label>log</label>
        <remoteHost>localhost</remoteHost>
        <port>9882</port>
    </appender>

    <root level = "INFO">
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="FLUENT_TEXT"/>
    </root>
</configuration>
```

이후 해당 logback 파일의 경로 설정

```
logging:
  level:
    com:
      ddib:
        product: info
  config: classpath:logback-dev.xml
```

4. Fluentd 에서 Elastic Search 으로 전송

```

<source>
```

```
    @type forward
    port 9880
    bind 0.0.0.0
    tag gateway
    <parse>
        @type json
    </parse>
</source>
<source>
    @type forward
    port 9881
    bind 0.0.0.0
    tag user
    <parse>
        @type json
    </parse>
</source>
<source>
    @type forward
    port 9882
    bind 0.0.0.0
    tag product
    <parse>
        @type json
    </parse>
</source>
<source>
    @type forward
    port 9883
    bind 0.0.0.0
    tag payment
    <parse>
        @type json
    </parse>
</source>
<source>
    @type forward
    port 9884
```

```
    bind 0.0.0.0
    tag notification
    <parse>
        @type json
    </parse>
</source>
<source>
    @type forward
    port 9885
    bind 0.0.0.0
    tag seller
    <parse>
        @type json
    </parse>
</source>
<source>
    @type forward
    port 9910
    bind 0.0.0.0
    tag gateway
    <parse>
        @type json
    </parse>
</source>
<match *.**>
    @type copy
    <store>
        @type elasticsearch
        host es01
        port 9200
        logstash_format true
        logstash_prefix spring
        logstash_dateformat %Y%m%d
        include_tag_key true
        type_name access_log
        tag_key @log_name
        flush_interval 1s
    </store>
```

```
    <store>
        @type stdout
    </store>
</match>
```

5. Kibana 에서 Elastic Search 에 저장 된 로그 데이터 시각화

`htttp://{실행한 서버주소 또는 로컬 호스트}:5601` 로 접속 후, `http://es01:9200` 입력 후 설정한 tag 로 검색하면 ES에 수집된 로그들을 확인할 수 있다. 여기선 tag 가 spring-* 으로 설정되어 있다.

`kibana : DASHBOARD - DISCOVER - create data view - Tag 에서 spring-*`

Micrometer & Zipkin : Tracing Tx

MSA 환경에서 하나의 서버가 다른 서버를 호출하는 경우가 많기에, 하나의 요청이 어떤 트랜잭션의 흐름을 가지고 실행되는지 확인하기 어려운 경우가 많다. 따라서 Tx 추적을 위해 Micrometer 와 Zipkin을 사용한다.

1. 의존성 추가

```
// micrometer & zipkin
    implementation 'io.zipkin.reporter2:zipkin-reporter-brave
    implementation 'org.springframework.boot:spring-boot-star
    implementation 'io.micrometer:micrometer-tracing-bridge-b
```

Spring Boot 버전 업에 따라 기존 사용하던 `sleuth` 는 사용 불가, 따라서 `micrometer` 를 사용한다.

이후 Zipkin 을 docker run 해준다.

`docker run -d -p 9411:9411 openzipkin/zipkin`

2. 로깅 설정 - application.yml - 모든 서버에 설정해준다.

```
management:
  tracing:
    sampling:
      probability: 1.0
```

```
    propagation:
      consume: b3
      produce: b3_multi
  zipkin:
    tracing:
      endpoint: "http://127.0.0.1:9411/api/v2/spans"
```

전송할 Zipkin 서버의 주소와 로그 설정을 담는다.

3. 위의 `logback.xml` 의 `CONSOLE` 에 로깅 출력 방식에 대해 명시가 되어 있다. 다음은 해당 파일의 일부분이다.

```
<property name="CONSOLE_LOG_PATTERN"
              value="%d{yyyy-MM-dd HH:mm:ss.SSS} %highlight(%

    <appender name="CONSOLE" class="ch.qos.logback.core.Conso
        <encoder class="ch.qos.logback.classic.encoder.Patter
            <pattern>${CONSOLE_LOG_PATTERN}</pattern>
        </encoder>
    </appender>
```

4. Zipkin 서버 `http://{실행 서버 주소 또는 로컬 호스트}:9411` 에서 수집된 로그들과 Tx 트레이싱을 확인한다.

환경 구성 - Kubernetes 환경

Google Cloud Platform

기존 스프링 환경 2개의 인스턴스에서 GCP 의 무료 크레딧 $300를 통해 4개의 인스턴스 구축

Kubernetes

1개의 마스터 노드와 3개의 워커 노드 구성

k8s Autoscaling

k8s의 HPA 를 통해 cpu 사용량 등 자원 사용량에 따라 오토스케일링을 적용한다.

hpa 설정

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: ddib-product-hpa
  namespace: ddib
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: ddib-product-deployment
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

서버 deployment 설정

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ddib-product-deployment
  namespace: ddib
spec:
  replicas: 3
  selector:
    matchLabels:
      app: ddib-product
  template:
    metadata:
```

```
      labels:
        app: ddib-product
    spec:
      containers:
      - name: ddib-product
        image: kimyusan/k8s_product:latest  # 이미지 이름을 적절히
        ports:
        - containerPort: 8082
```

서버 service 설정

```
# -----------------------------------
# nodeport 설정
#apiVersion: v1
#kind: Service
#metadata:
#  name: ddib-front-service
#spec:
#  type: NodePort
#  selector:
#    app: ddib-front
#  ports:
#    - port: 3000
#      targetPort: 3000
#      nodePort: 30010
#-------------------------------------
# clusterip 를 사용한 경로 매핑 설정정
apiVersion: v1
kind: Service
metadata:
  name: ddib-product-service
  namespace: ddib
spec:
  type: NodePort
  selector:
    app: ddib-product
```

```
  ports:
    - protocol: TCP
      port: 8082
      targetPort: 8082
      nodePort: 30002
```

Grafana & Prometheus

1. prometheus 설치

```
1.nfs 서버 생성

2. kubectl apply -f prompv.yaml

3.. kubectl apply -f prompvc.yaml

4. helm install promehteus prometheus-comunity/prometheus -f
```

- prompv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: prom-pv
spec:
  capacity:
    storage: 10Gi  # NFS 서버의 용량에 따라 조정
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany  # 여러 파드에서 읽기 및 쓰기 가능하도록 설정
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /mnt/shared  # NFS 서버의 공유 경로
    server: 10.182.0.12  # NFS 서버의 주소
```

- prompvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: prom-pvc
spec:
  accessModes:
    - ReadWriteMany
  volumeName: prom-pv # 위에서 생성한 pv 의 이름
  resources:
    requests:
      storage: 10Gi  # 필요한 스토리지 용량
```

- values-prometheus.yaml

```
server:
  enabled: true


  persistentVolume:
    enabled: true
    accessModes:
      - ReadWriteOnce
    existingClaim: "prom-pvc"
    size: 10Gi
  replicaCount: 1

  retention: "15d" #프로메테우스 데이터 보유 기간
```

2. grafana 설치

```
1.nfs 서버 생성
```

```
2. kubectl apply -f grafanapv.yaml

3.. kubectl apply -f grafanapvc.yaml

4. helm install grafana grafana/grafana -f values helm-grafan
```

- grafanapv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: grafana
spec:
  capacity:
    storage: 10Gi  # NFS 서버의 용량에 따라 조정
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany  # 여러 파드에서 읽기 및 쓰기 가능하도록 설정
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /mnt/shared/grafana-pv  # NFS 서버의 공유 경로
    server: 10.182.0.12  # NFS 서버의 주소
```

- grafanapvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: grafana-pvc
spec:
  accessModes:
    - ReadWriteMany
  volumeName: grafana-pv # 위에서 생성한 pv 의 이름
  resources:
```

```
    requests:
      storage: 10Gi  # 필요한 스토리지 용량
```

- helm-grafana-pv-pvc.yaml

```
apiVersion: v1
kind: PersistentVolume      ## PV
metadata:
  name: grafana
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    server: 10.182.0.12
    path: /nfs_shared/grafana
---
apiVersion: v1
kind: PersistentVolumeClaim ## PVC
metadata:
  name: grafana
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

3. spring boot

- 3-1. gradle 추가

```
    // metric 수집
    implementation 'org.springframework.boot:spring-boot-star
    implementation 'io.micrometer:micrometer-core'
    implementation 'io.micrometer:micrometer-registry-prometh
```

- 3-2. yml 설정 추가

```
# metric 설정
management:
  endpoint:
    metrics:
      enabled: true
    prometheus:
      enabled: true
  endpoints:
    web:
      exposure:
        include: prometheus
      base-path: /actuator/product
  metrics:
    export:
      prometheus:
        enabled: true
```

4. prometheus server 에 매트릭 정보 추가

- prometheus-configmap.yaml

```
data:
    prometheus.yml:
    ...
        scrape_configs:
            - job_name: "spring-waiting"
              metrics_path: '/actuator/waiting/prometheus'
```

```
      scrape_interval: 1s
      static_configs:
      - targets: ['ddib-waiting-service.ddib.svc.clus
    - job_name: "spring-product"
      metrics_path: '/actuator/product/prometheus'
      scrape_interval: 1s
      static_configs:
      - targets: ['ddib-product-service.ddib.svc.clus
    - job_name: "spring-payment"
      metrics_path: '/actuator/payment/prometheus'
      scrape_interval: 1s
      static_configs:
      - targets: ['ddib-payment-service.ddib.svc.clus
```

변수 및 설정 파일

Spring yml 파일 설정

- 공통 환경
  - application.yml

    > Swagger, JPA, Batch, Spring Mail, Intellij Encoding, properties Management 에 대한 설정 정보

- 개발 Dev 환경
  - application-dev.yml

    > 개발 환경 시 적용되는 DB 및 URL 매핑 설정 정보

- 배포 Prod 환경
  - application-prod.yml

    > 배포 환경 시 적용되는 DB 및 URL 매핑 설정 정보

- Secret keys & Git Ignored (jenkins Credentials 를 통한 Key 관리)
  - application-key.yml

    > Amazon S3 등 Secret Key 에 대한 설정 정보

  - `firebase/ddib-17d11-firebase-adminsdk-cbmvc-22ed36311c.json`

    > Firebase Clouding Messaging 에 사용되는 Admin Key 설정 정보

Secret Files & GIT Ignored

- ignore 된 파일들은 Jenkins Credentials 를 활용하여 key를 주입받도록 해준다.

  Jenkins Credentials 적용 Flow

  - Jenkins 접속 후 `Jenkins 관리` - `Credentials` 이동
  - `Credentials` 의 저장된 `scoped` 중에서, `domains` 의 `global` 클릭
  - `Add Credentials` 클릭
  - 현재 숨겨지고 있고, `crendentials` 에 등록하려느 파일은 key 파일이므로, `kind` 는 `secret file` 설정
  - `id` 는 편한 것으로 네이밍하되, 향후 파이프라인 내에서 사용 시 필요한 value
  - `description` 역시 편하게 작성한다.
  - 이후 Jenkins Pipeline 이 설정되어 있는 파일로 이동한다. (JenkinsFile.groovy)
  - 해당 파이프라인 내에서, 적절한 위치에서 `application-key.properties` 가 작성되어야 한다. 현재 `GitLab` 내에 `Back-end` 폴더를 git clone 한 후, 빌드를 진행하므로 빌드 전에 해당 properties 가 존재해야만 정상적인 빌드, 배포가 가능해질 것이다.
  - 따라서 `Git clone stage` 와 `Back Build` 사이에 `properties` 를 추가하는 stage 가 위치하도록 한다.
  - stage 작성 시, 해당 계정이 `properties` 가 위치할 directory에 대해 작성 권한이 없을 수 있다. 따라서 디렉터리에 대해서 작성을 하려면 읽기 권한이 있어야 하고 (x), `properties` 를 작성해야 하며(w), 디렉토리 하위 레벨에 모두 적용해야 그 하위에 위치할 `properties`가 적용되므로 하위 전파 옵션을 적용해야하며(-R), 파일이 위치할 디렉토리 하위에 대해서 읽기가 가능해야 전파 옵션 적용이 가능하다 (r).
  - 따라서 해당 디렉토리에 `R a=rwx` 옵션을 적용해준다.

```
stage('Git clone') {...}
stage('Add Env') {
        steps {
            dir('./backend') {
                withCredentials([file(credentialsId: 'ke
y', variable: 'key')]) {
                    sh 'chmod -R a=rwx src/main/resource
s'
                    sh 'cp ${key} src/main/resources/app
lication-key.properties'
                }
            }
        }
    }
stage('build backend') {...}
```

---

k8s 수동 배포

1. intellij 에서 gradle - clean, gradle - bootjar 를 해준다.
2. cli 이동
3. docker build -t 만들 도커이미지 이름 : 태그 . ← . 을 붙여야 한다 !
4. docker push 이미지 푸시할 경로
5. k8s 서버 접속
6. 파드 조회

   `kubectl get pod -n ddib`
7. 삭제할 파드 또는 deployment 확인 후 삭제

   `kubectl delete pod 삭제할 파드 이름 -n ddib`
8. 해당 파드를 관리하는 파일이 있는 폴더 위치로 이동

   `cd 이동할 파일 경로, ~ 랑 / 잘 구분할 것`
9. 파일 재적용

```
kubectl apply -f 해당파일.yaml
```

사용 포트

PORT	DESCRIPTION
80	Http
443	Https
3000	Frontend
3306	MySQL DB
5601	Kibana
6379	Redis
8000	Spring Gateway
8081	Spring Microservice - user
8082	Spring Microservice - product
8083	Spring Microservice - payment
8084	Spring Microservice - notification
8085	Spring Microservice - seller
8761	Spring Eureka
9010	Spring Microservice - waiting
9200	Elastic Search
9300	Elastic Search
9411	Zipkin
9881	Logging - user
9882	Logging - product
9883	Logging - payment
9884	Logging - notification
9885	Logging - seller
9910	Logging - waiting
9999	Jenkins

Database Dump File

> exec 폴더 내에 dump 파일 참조