|  |  |  |
| --- | --- | --- |
| 결  재 | 담당 | 원장 |
|  |  |

**K-Digital Training**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 클라우드컴퓨팅과 보안솔루션을 활용한  DC 엔지니어 양성 과정 | |  | | |
|  |  | | |  | | |
|  | 1차 프로젝트 기술 보고서 | | |  | | |
|  | - [K8S] 쿠버네티스를 활용한 3-Tier 웹서비스 구축 - | | |  | | |
|  |  | | |  | | |
|  | 2024.07.26.(금) | | |  | | |
|  |  | | |  | | |
|  | 구성원 : 황기창  왕진우  임희진  박지영 | | |  | | |
|  |  |  | | |

## 목차

1. **프로젝트 개요…………………………………………………………………….…………………... 3**
2. **프로젝트 진행 순서…………………………………………………………….…………………… 4**
3. **프로젝트 전체 구성도………………………………………………………….…………………… 5**
4. **프로젝트 구성 요소…………………………………………………………….…………………… 6**
5. **프로젝트 배경기술……………………………………………...…………………………………… 7**
6. **프로젝트 추진 체계……………………………………………………………………………….…** **8**
   1. **프로젝트 참여명단……………………………………………………………………….… 8**
   2. **프로젝트 역할 분담……………………………………………………………………….. 8**
7. **프로젝트 시나리오…………………………………………………………………...……………… 9**
8. **프로젝트 구성 과정………………………………………………………………..……………… 11**
   1. **HA 멀티 클러스터 환경 구성하기………………………………………………..… 11**
   2. **3-Tier 구성…………………..…………………………………………………………….. 13**
      1. **Nginx………………………………………………………………………………. 13**
      2. **Tomcat…………………………………………………………………………….. 13**
      3. **DB…………………………………………………………………………………… 14**
   3. **JSP 웹페이지 만들기……………………………………………………………………. 15**
   4. **Dockerfile 작성 및 경량화………………………………………………………..….. 20**
      1. **Nginx Dockerfile 작성………………………………….……………………. 20**
      2. **Tomcat Dockerfile 작성…………………………….……………………….. 21**
   5. **DB 이중화………………………………………………………………………………….. 22**
   6. **YAML 파일 생성………………………………………………………………………….. 25**
      1. **Deployment……………………………………………………………………… 26**
      2. **Service (svc) ………………………………………………………………….… 30**
      3. **Persistent Volume (pv) 및 Persistent Volume Claim (pvc)……... 34**
      4. **Horizontal Pod Autoscaler (hpe) ……………………………………….. 38**
      5. **Ingress…………………………………………………………………………….. 43**
   7. **Rolling Update 및 Rollback……………………………………………………….... 45**
   8. **WhaTap 설치……………………………………………………….…………………….. 48**
   9. **ArgoCD와 Github 연동…………………………………….…………………………. 49**
   10. **Metric Server 구성………………………………………………..…………………….. 51**
   11. **JMeter 부하 테스트…………………………………………………………………….. 52**
9. **결론 및 제언(Trouble Shooting)………………………………..…………………………… 54**

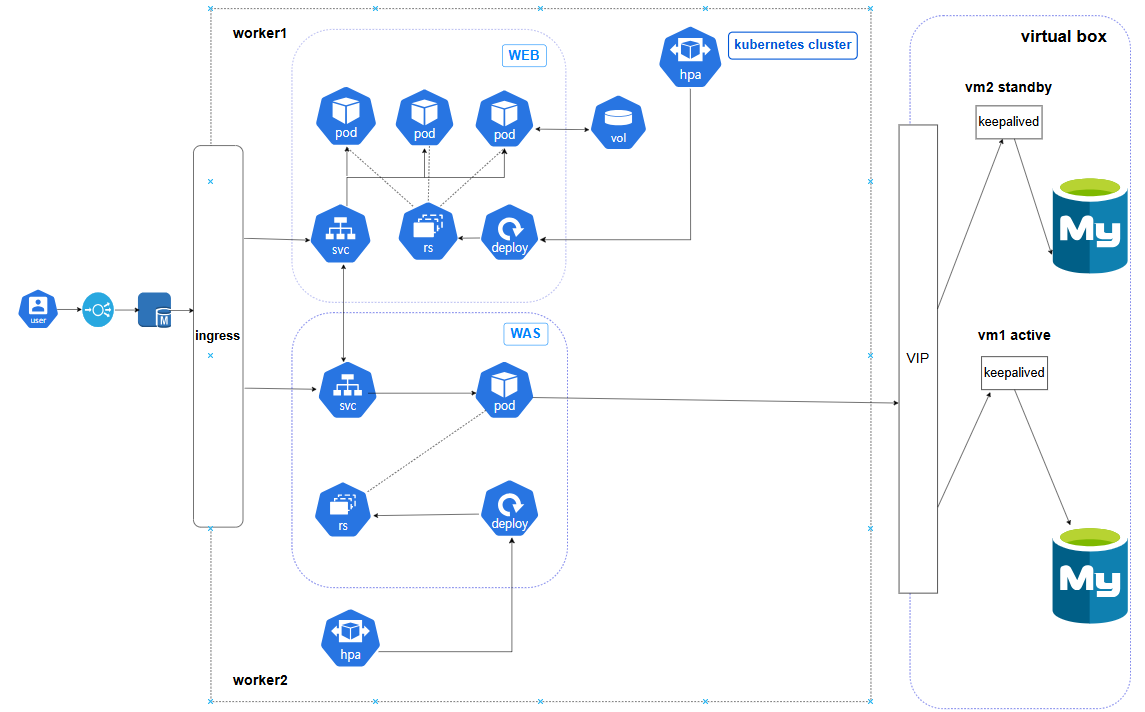
**프로젝트개요**

|  |  |
| --- | --- |
| 항목 | 내용 |
| 교육 기관 | 한국정보교육원 |
| 과정명 | 클라우드컴퓨팅과 보안솔루션을 활용한 DC엔지니어 양성 |
| 팀명 | 1조 |
| 팀원 | 황기창, 왕진우, 임희진, 박지영 |
| 생성 배경 | 현대의 카페 환경은 빠르게 변화하고 있기 때문에 고객들은 더 나은 서비스 경험을 요구하고 있다. 이에 1조는 고객이 직접 주문하고 결제할 수 있는 자가 서비스 방식을 제공하여, 대기 시간을 단축하고 주문 과정을 간소화할 수 있는 홈페이지를 구축하였다. |
| 프로젝트 목표 | 3-Tier 웹서비스 구축에 따른 오토스케일링 및 경량화. 쿠버네티스(Kubernetes)를 활용하여 프론트엔드(Nginx), 백엔드(Tomcat), 데이터베이스(MySQL)의 3-Tier 아키텍처를 설계하고 구현한다. 이를 통해 애플리케이션의 자동 확장성과 경량화를 달성하고, 안정적이고 효율적인 서비스를 제공한다. 또한, MySQL 데이터베이스의 이중화를 통해 데이터의 가용성과 무결성을 보장한다. |

# 프로젝트 진행 순서

|  |  |
| --- | --- |
| 단계 | 내용 |
| 1. 프로젝트 계획 수립 | 목표 설정 요구사항 분석 일정 계획 수립 |
| 2. 환경 설정 및 인프라 구축 | 개발 환경 및 도구 설정 클라우드 인프라 구축 네트워크 구성 |
| 3. 3-Tier 아키텍처 설계 및 구현 | Nginx 설정 및 Docker 이미지 생성 Tomcat 설정 및 Docker 이미지 생성 MySQL 설정 및 이중화 |
| 4. 애플리케이션 개발 | 프론트엔드 개발 (JSP 웹페이지 작성) 백엔드 개발 (서버 로직 구현) 데이터베이스 스키마 설계 및 테이블 생성 |
| 5. 쿠버네티스 설정 | YAML 파일 작성 (Deployment, Service, PV, PVC, HPA, Ingress) 클러스터에 애플리케이션 배포 오토스케일링 설정 |
| 6. 지속적 통합 및 배포 (CI/CD) | ArgoCD와 Github 연동 자동 배포 파이프라인 설정 |
| 7. 모니터링 및 성능 최적화 | WhaTap 설치 및 모니터링 설정 JMeter를 이용한 부하 테스트 |
| 8. 테스트 및 검증 | 기능 테스트 성능 테스트 사용자 수용 테스트 |
| 9. 문서화 및 교육 | 사용자 매뉴얼 작성 팀원 교육 및 기술 전수 |
| 10. 프로젝트 종료 및 평가 | 최종 보고서 작성 프로젝트 평가 및 피드백 수집 차후 개선 사항 도출 |

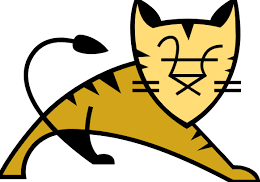
**프로젝트 전체 구성****도**



**프로젝트 구성요소**

|  |  |  |
| --- | --- | --- |
| **구성 요소** | **버전/종류** | **설명** |
| **웹 서버** | Nginx 1.14.2-alpine | 경량화된 웹 서버로, HTTP 요청을 처리하고 로드 밸런싱을 수행. |
| **애플리케이션 서버** | Tomcat 9.0.91 | Java 서블릿 컨테이너 및 웹 서버, JSP 처리. |
| **데이터베이스** | MySQL 최신버전 (8.x) | 관계형 데이터베이스, 데이터 저장 및 관리. |
| **JDK** | openjdk-8-jdk | Java Development Kit, Java 애플리케이션 개발 환경. |
| **JSP** | - | JavaServer Pages, 동적 웹 페이지 생성. |
| **MySQL 커넥터** | mysql-connector-j-8.2.0 | MySQL과 Java 애플리케이션 간의 연결을 위한 JDBC 드라이버. |
| **Metrics Server** | - | Kubernetes 클러스터의 메트릭 수집 및 제공. |
| **ArgoCD** | - | Kubernetes의 GitOps 배포 및 관리 도구. |
| **Github** | - | 소스 코드 저장 및 버전 관리를 위한 플랫폼. |
| **JMeter** | apache-jmeter-5.6.3 | 부하 테스트 도구, 성능 테스트 수행. |
| **WhaTap** | - | 실시간 모니터링 및 성능 관리 도구. |

**프로젝트 배경 기술**

****

****

### 프로젝트 참여명단

|  |  |  |
| --- | --- | --- |
| **이름** | **역할** | **담당 업무** |
| 황기창 | 팀장 | 전체 프로젝트 관리 및 Nginx 설정 |
| 왕진우 | 백엔드 개발자 | Tomcat 설정 및 JSP 페이지 작성 |
| 임희진 | 프론트엔드 개발자 | 웹페이지 디자인 및 구현 |
| 박지영 | 데이터베이스 관리자 | MySQL 설정 및 DB 이중화 구현 |

### 역할 분담

|  |  |  |
| --- | --- | --- |
| 구성 요소 | 담당자 | 업무 내용 |
| HA 멀티 클러스터 환경 구성 | 황기창 | 멀티 클러스터 환경 설계 및 구축 |
| Nginx 설정 | 황기창 | Nginx 서버 설정 및 Dockerfile 작성 |
| Tomcat 설정 | 왕진우 | Tomcat 서버 설정 및 Dockerfile 작성 |
| DB 설정 및 이중화 | 박지영 | MySQL 데이터베이스 설정 및 이중화 구현 |
| JSP 웹페이지 작성 | 왕진우 | JSP 파일 작성 및 웹페이지 연동 |
| YAML 파일 생성 | 임희진 | Kubernetes 매니페스트 파일 작성 및 적용 |
| Rolling Update/Rollback | 황기창 | Rolling Update 및 Rollback 절차 구현 |
| WhaTap 설치 | 임희진 | 시스템 모니터링을 위한 WhaTap 설치 |
| ArgoCD와 Github 연동 | 박지영 | ArgoCD 설정 및 GitHub 연동 |
| JMeter 부하 테스트 | 왕진우 | JMeter를 이용한 시스템 부하 테스트 |

**프로젝트 시나리오**

|  |  |
| --- | --- |
| **항목** | **설명** |
| **개요** | 저희는 쿠버네티스 상에서 3티어 아키텍처를 설계하여 카페의 DB에 등록된 회원만 온라인 웹 페이지에서 메뉴를 구입할 수 있는 시스템을 구현하였습니다. 일반 카페에서 운영하는 것처럼 회원 여부에 따른 할인 적용은 개발에 큰 부담이 되어 제외하고, 단순한 시나리오로 구성하였습니다. |
| **시스템 구성** | 저희가 구축한 시스템은 다음과 같은 흐름을 따릅니다: |
| **1. 사용자 접근** | 사용자는 웹 페이지에 접근하기 위해 먼저 Nginx 서버를 통해 Tomcat 서버로 연결됩니다. 이는 Tomcat 서버의 부하를 분산하기 위함입니다. |
| **2. 파일 저장소** | HTML 파일과 DB와 연동된 JSP 파일들은 Tomcat 서버에 저장됩니다. |
| **3. 접속 구조** | Tomcat 서버에 부하가 발생하지 않도록 사용자는 Nginx 서버를 반드시 거치도록 구성하였습니다. |
| **4. 메인 페이지** | 사용자는 Nginx 서버를 통과하여 Tomcat 서버에 저장된 main.html 파일을 보게 됩니다. 여기서 로그인을 하거나 회원 가입 페이지로 이동할 수 있습니다. |

**시나리오** **상세**

**1. 로그인 시나리오**

* 사용자가 main.html에서 바로 로그인을 시도합니다.
* main.html은 main.jsp로 연결되고, 이 파일은 카페 DB의 users 테이블을 참조하여 회원 여부를 확인합니다.
* 회원 정보가 없으면 main.html로 돌아가고, 정보가 일치하면 "환영합니다" 메시지와 함께 order.jsp 페이지로 이동합니다.
* order.jsp 페이지에서 사용자는 메뉴 이미지를 클릭하여 장바구니에 담을 수 있습니다.

**2. 회원 가입 시나리오**

* 회원 가입을 하지 않은 사용자는 main.html의 회원 가입 버튼을 클릭하여 signup.jsp 페이지로 이동합니다.
* 여기서 아이디와 비밀번호를 입력하면, 해당 정보가 DB에 저장됩니다.
* 이후 사용자는 order.jsp 페이지에서 메뉴를 장바구니에 담을 수 있습니다.

**Ingress 구성**

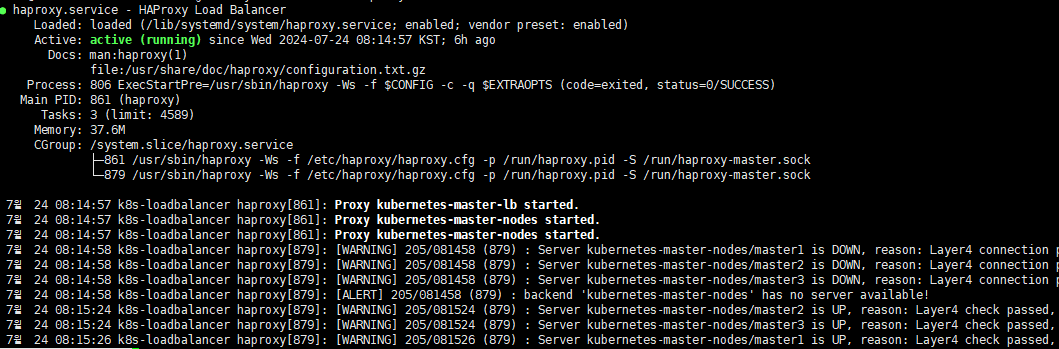
* ingress.yaml 파일에서 HTML과 JSP 파일들은 main, sign, order 부분으로 분류되었습니다.
* 사용자가 signup.jsp와 order.jsp 파일에 직접 접근하지 못하도록 main.html과 main.jsp 파일은 루트 디렉토리에, signup.jsp와 order.jsp 파일은 하위 디렉토리에 배치하였습니다.

**DB 설계**

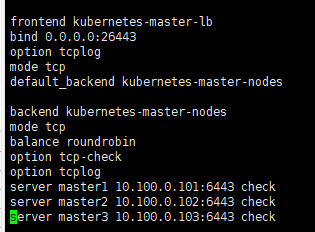
* 저희는 카페의 DB에 users 테이블만 생성하여 회원 정보 저장을 구현하였습니다.
* 특정 메뉴가 매진되었을 때 "Sold Out" 이미지로 변경되고, 다시 판매가 가능할 때 롤백이 되는 롤링 업데이트 기능을 구현하였습니다.

1. **HA 멀티 클러스터 구성**

* **로드밸런서 노드 1개**
* **역할:** 고가용성 보장, 클러스터 간 연결
* haproxy 패키지 설치 후 /etc/haproxy 디렉토리에 haproxy.cfg 파일을 생성하여 로드밸런서 노드 설치를 진행했습니다.

****

* /etc/haproxy/haproxy.cfg 파일에 마스터 노드 3개의 정보를 추가하여 haproxy 데몬을 재활성화하여 부하 분산을 시작하였습니다.



* **마스터 노드 3개**
* **마스터 노드 1개 (k8s-master1)**
* 컨트롤 플레인 엔드포인트를 로드밸런서 노드의 IP로 설정하여 HA 클러스터를 생성하였습니다.

****

**→ 의미: 10.100.100.0/24 CIDR 대역**을 Pod 네트워크 대역으로 설정하여 로드밸런서 노드 IP인 10.100.0.100을 엔드포인트로 하는 클러스터를 생성하는 것입니다.

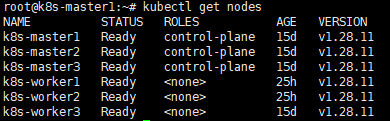
* **나머지 마스터 노드 2개**
* HA 클러스터의 두 개의 조인 문장 중 상단 조인 문장을 실행하였습니다.

****

* **워커 노드 2개**

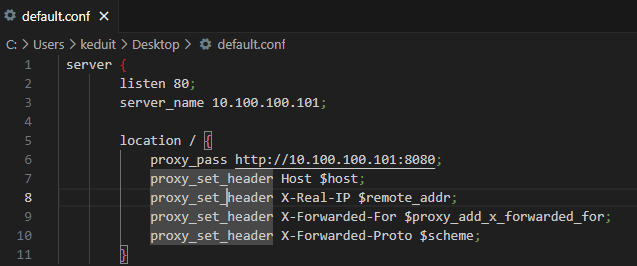
****

* 워커 노드 3개는 두 개의 조인 문장 중 하단 조인 문장을 실행하였습니다.
* **성공 화면**

****

1. **3티어 아키텍처 구축**
2. **nginx**

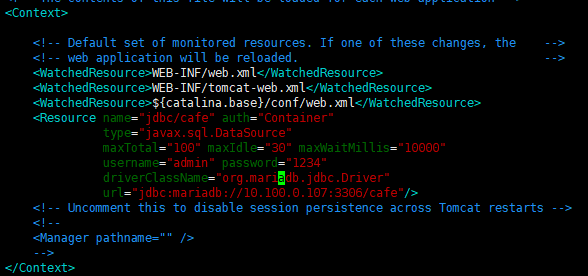
* **역할:** Client가 접속 후 Tomcat으로 이동하는 역할
* (Tomcat을 proxy 서버로서 설정)
* **/etc/nginx/conf.d/default.conf**

****

1. **Tomcat**

* **역할:** Tomcat 서버에 접속한 Client가 DB와 통신을 하게 도와주는 역할
* **/usr/local/tomcat/conf/context.xml**

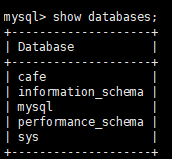
(연동할 DB 정보를 저장)

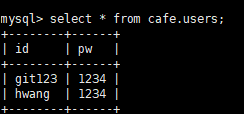
****

1. **DB**

* **역할:** Data를 Database 내부의 Table에 저장

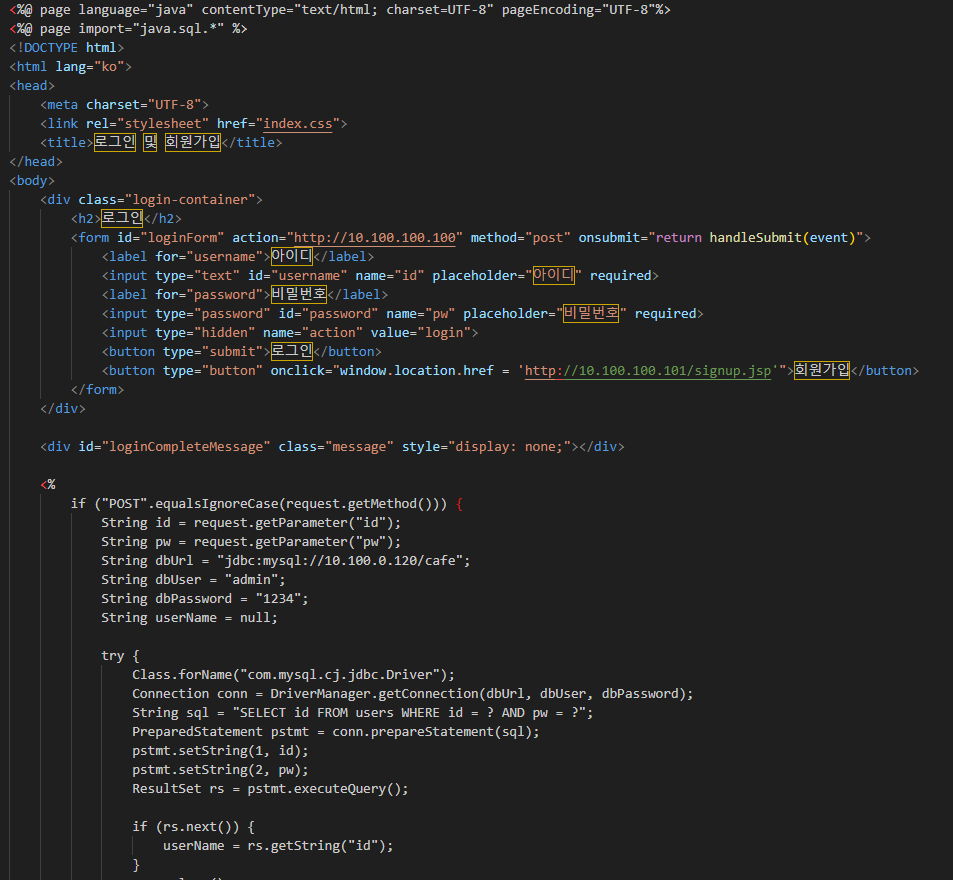
→ Table을 통해 서비스를 Query문 형태로 처리



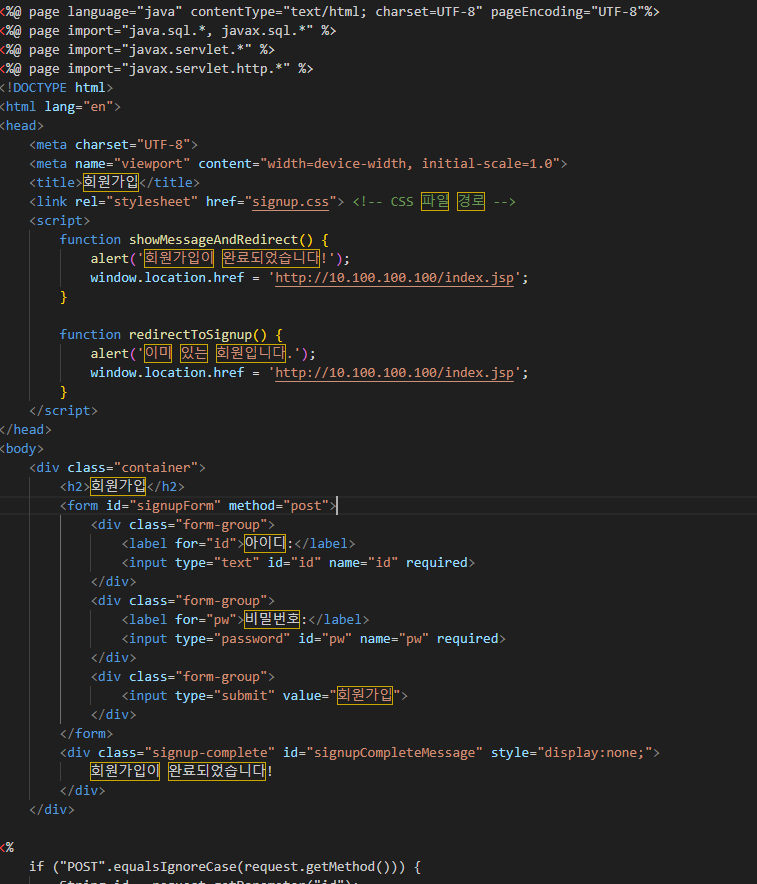
****

1. **JSP 웹페이지 만들기**

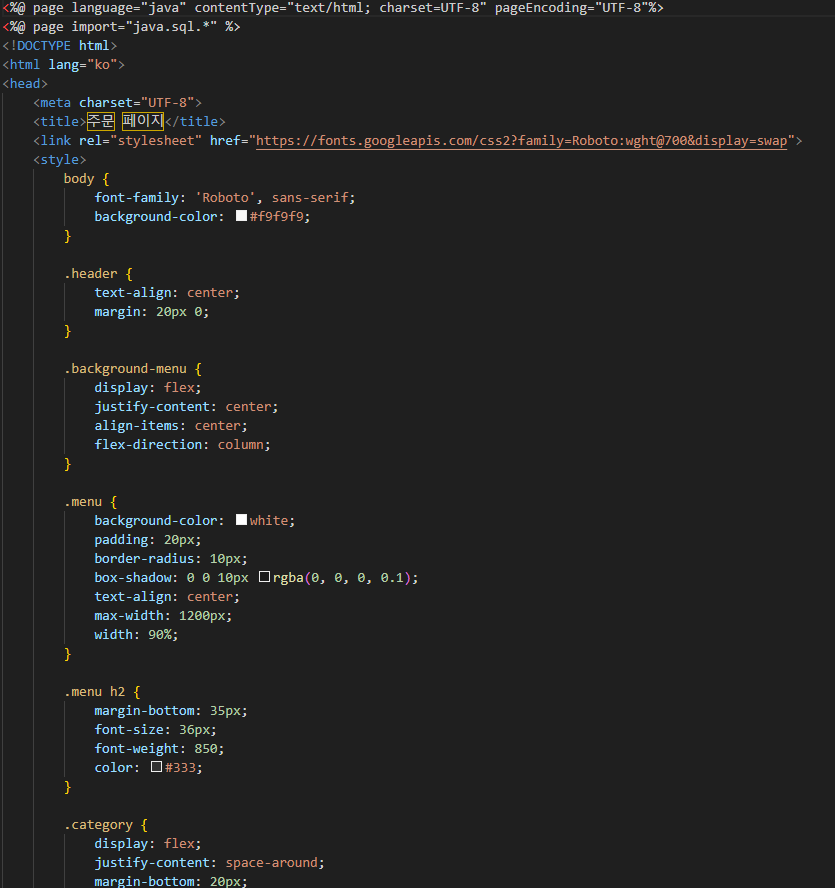
* **JSP 역할:** JDBC Driver를 사용하여 데이터베이스와 연결하고, 쿼리문을 통해 데이터를 조회하거나 업데이트를 합니다.
* **main.jsp**
* 아이디와 비밀번호를 입력하여 로그인합니다.
* 또는 회원가입 버튼을 눌러 signup.jsp로 이동합니다.

****

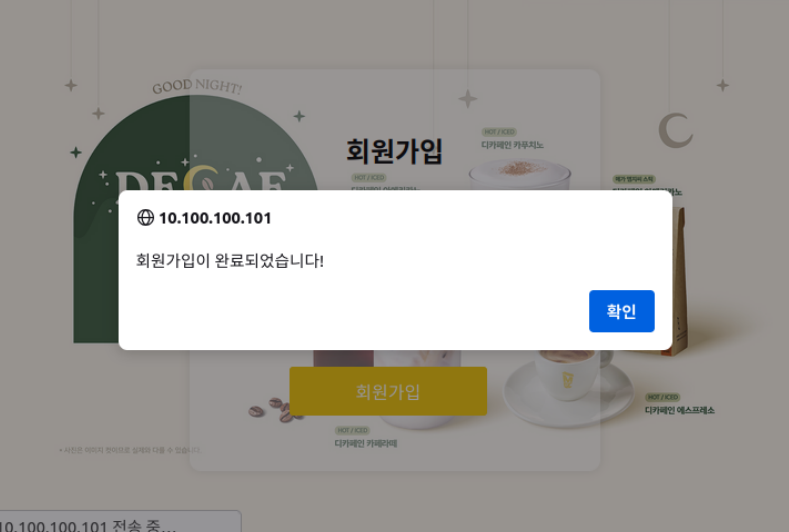
* **signup.jsp**
* DB에 없는 회원일 때, DB에 insert into values 쿼리문을 실행
* DB에 있는 회원일 때, main.html로 돌아갑니다.

****

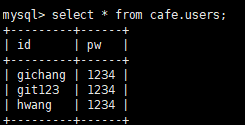
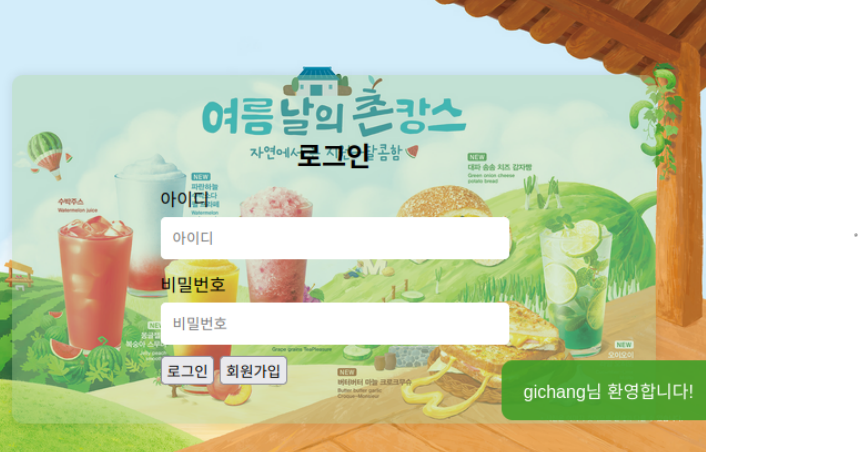
* **order.jsp**
* 로그인에 성공한 Client는 해당 jsp에서 이미지를 클릭하여 주문합니다.

****

* jsp와 DB 연동 성공 화면 (1)

****

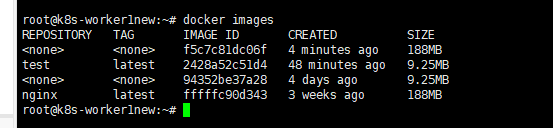
* jsp 파일과 DB 연동 성공 화면 (2)

****

1. **Dockerfile 작성 및 경량화**
2. **Nginx Dockerfile** **작성**

****

→ 저희는 nginx 도커 이미지를 생성할 때 alpine 버전으로 nginx 이미지 경량화를 진행했습니다.

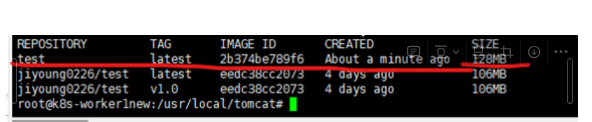
****

→ 그 결과, 188MB에서 9.25MB까지 경량화를 성공하였습니다.

1. **Tomcat Dockerfile 작****성**

****

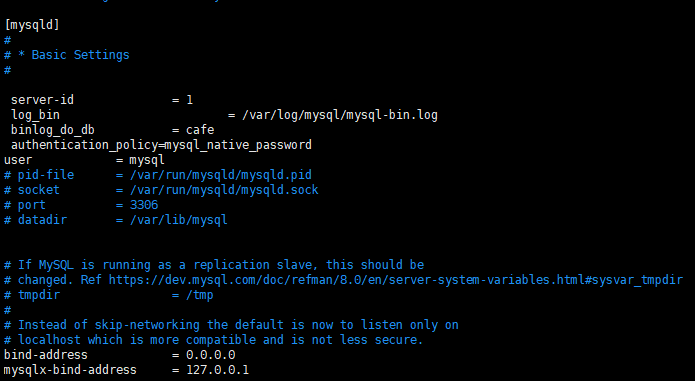
→ 저희는 Tomcat 기반 도커 이미지를 생성할 때, 멀티 스테이지 방식으로 빌드를 진행하였습니다.



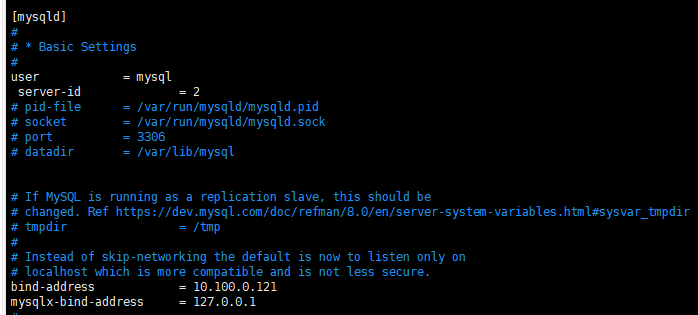
→ 그 결과, 128MB까지 경량화에 성공하였습니다. (단, 저희는 nginx 서버에 별도로 html 파일을 저장하지 않고 html 파일들과 jsp 파일들을 Tomcat Container에 배치하는 식으로 구성을 하였습니다.)

1. **DB 이중화**

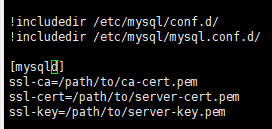
* **Master-Slave 관계 이중화**
* Client에 관한 정보가 저장되어 있는 DB에서 data 손실을 예방하기 위해 이중화를 설계하였습니다.
* Master DB 설정 (/etc/mysql/mysql.conf.d/mysqld.cnf)



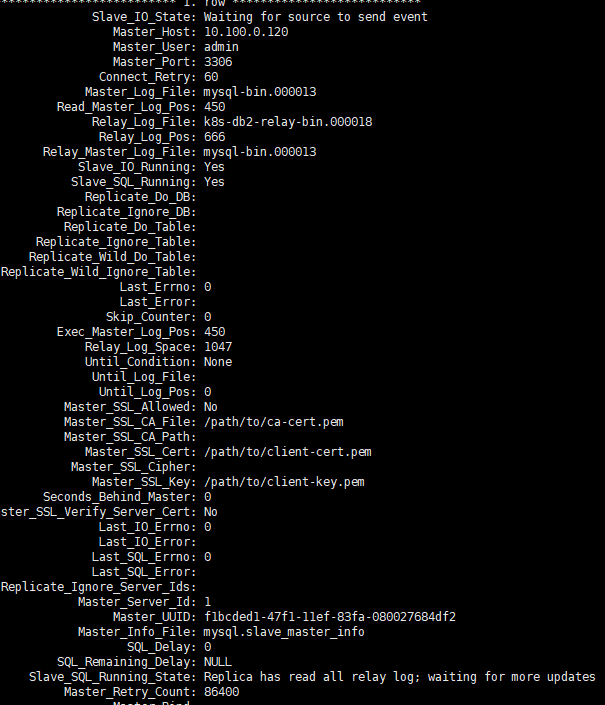
* Slave DB 설정

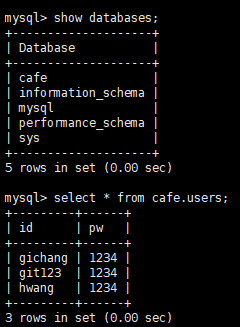
****

* 보안을 높이기 위한 인증 설정 역시 추가하였습니다.

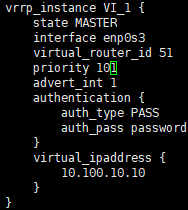
****

* **성공 화면**

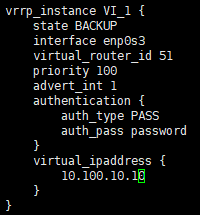
****

****

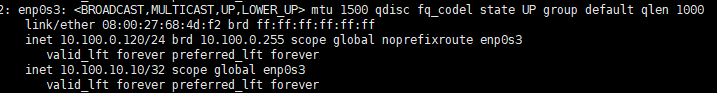
* **Active-Standby 관계의 keepalived 구현**
* 저희는 keepalived package를 설치 후, Virtual IP를 기반으로 Active DB가 종료됐을 때, Standby DB가 그 역할을 대신하는 것을 설계하여 고가용성을 살리는 것을 목표로 했습니다.
* **Active DB** (/etc/keepalived/keepalived.conf**)**

****

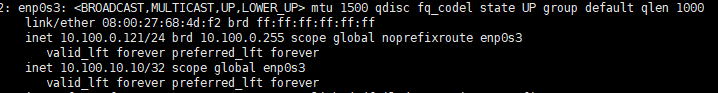
* **Standby DB (**/etc/keepalived/keepalived.conf**)**

****

* **성공 화면**
* **Active DB**

****

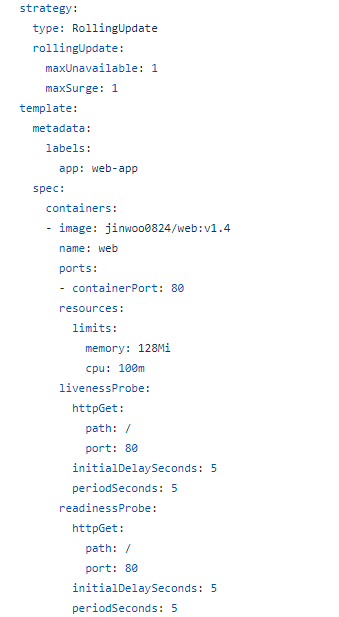
* **Standby DB (Active DB 종료 후)**

****

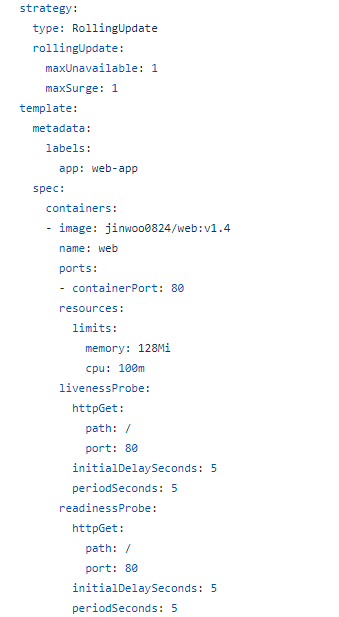
1. **YAML 파일**
2. **Deployment**

* **nginx Deployment**
* 저희는 nginx deployment 배포 전략을 RollingUpdate로 설정하여 무중단 배포 환경을 구성하였습니다.
* 그리고, 각 Deployment에 livenessProbe와 readinessProbe를 적용하여 주기적으로 / 경로로 HTTP GET 요청을 보내 컨테이너의 상태를 확인하는 것을 구현하였습니다.
* 또한 Volume 역시 마운트하여 컨테이너 상에서 갑작스럽게 파드가 종료되더라도, Data의 손실을 예방하려 하였습니다.

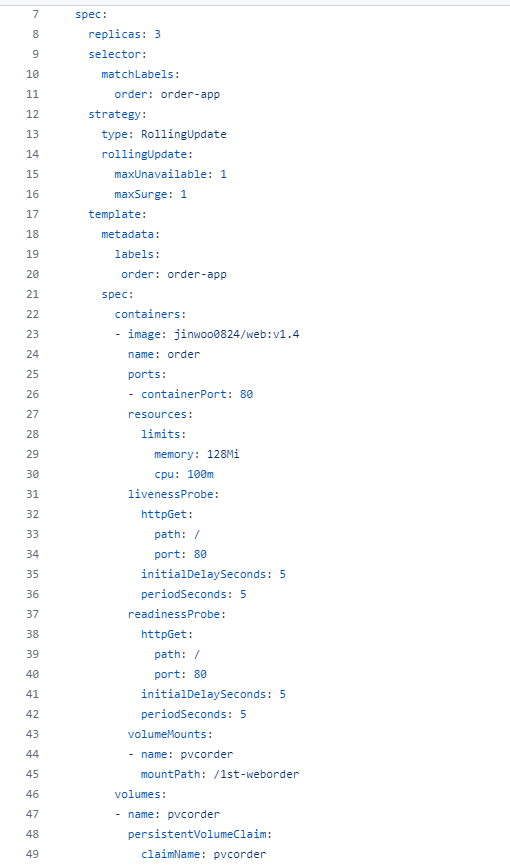
**ex) nginx-main.yaml**

****

**ex) nginx-sign.yaml**



**ex) nginx-order.yaml**

****

* **Tomcat Deployment**
* Deployment에 livenessProbe와 readinessProbe를 적용하여 주기적으로 / 경로로 HTTP GET 요청을 보내 컨테이너의 상태를 확인하는 것을 구현하였습니다.
* 또한 Volume 역시 마운트하여 컨테이너 상에서 갑작스럽게 파드가 종료되더라도, Data의 손실을 예방하려 하였습니다



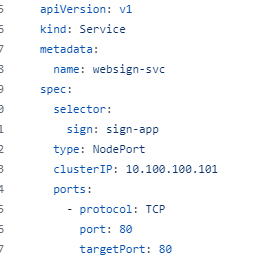
1. **Service**

* **nginx Service**
* nginx Deployment 3개의 Service type은 NodePort로 지정하여 Client가 내부 Deployment로 access가 되도록 하였습니다.

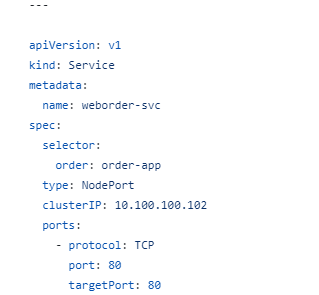
**ex) nginx-main\_svc.yaml**

****

**ex) nginx-sign\_svc.yaml**

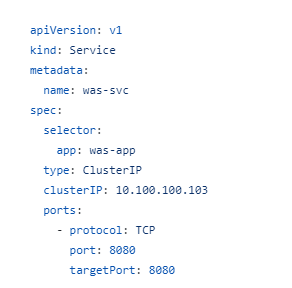
****

**ex) nginx-order\_svc.yaml**

****

* **Tomcat Service**
* nginx Deployment의 Service type은 ClusterIP로 지정하여 client가 무단으로 DB에 접속하는 것을 예방하고자 하였습니다.

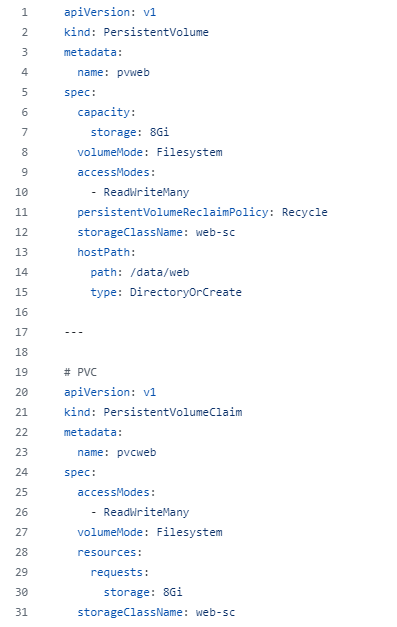
**ex) tomcat-svc.yaml**

****

1. **Persistent Volume & Persistent Volume Claim**

* **nginx, Tomcat pv, pvc 설정**
* 각 deployment에 volume을 설정을 pvc.yaml 파일에서 정의하였습니다.

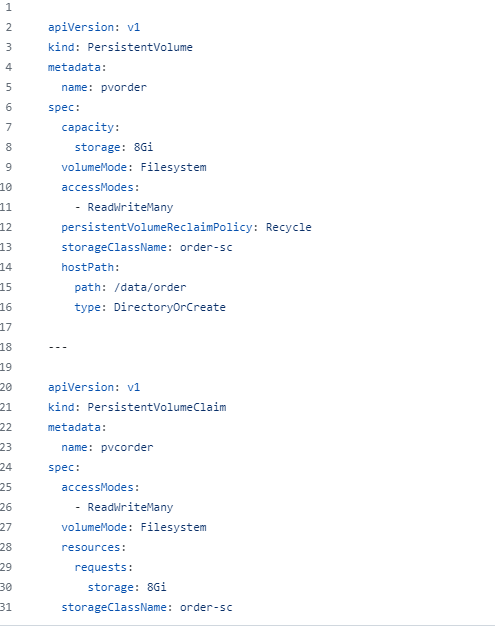
**ex) nginx-main\_pvc.yaml**

****

**ex) nginx-sign\_pvc.yaml**

****

**ex) nginx-order\_pvc.yaml**

****

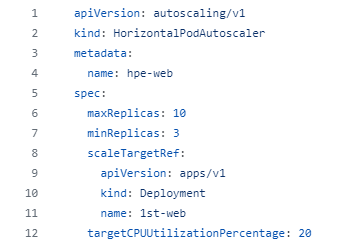
**ex) tomcat\_pvc.yaml**

****

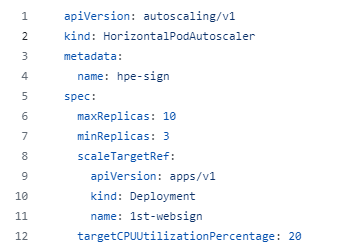
1. **Horizontal Pod Autoscaler**

* Autoscaling 설정을 하여 CPU에 과부하가 발생했을 때 pod의 개수를 늘리면서 이를 예방하고자 설정하였습니다.

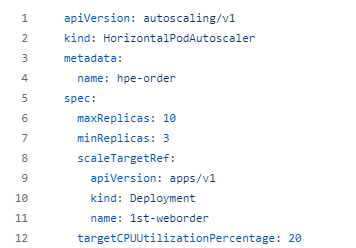
**ex) nginx\_main.hpa.yaml**

****

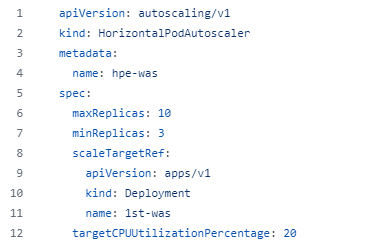
**ex) nginx\_sign\_hpa.yaml**

****

**ex) nginx-order\_hpa.yaml**

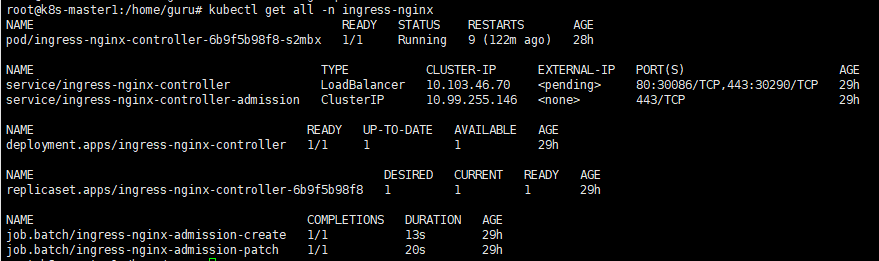
****

**ex) tomcat\_hpa.yaml**

****

1. **Ingress**

* 저희는 ingress controller를 설치하여 HTTP 및 HTTPS 트래픽 라우팅을 구현했습니다.
* 사용 파일: https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.6.4/deploy/static/provider/cloud/deploy.yaml
* Ingress Controller 설치 성공 화면



**ex) ingress.yaml**

ingress.yaml 파일에서 client는 각 pod에 라우팅할 때, / 경로를 거치게 설계했습니다.



1. **Node scheduling 구****현**

* 저희는 각 노드별 스케줄링을 설정하여 특정 노드에 과부하가 발생하는 현상을 예방하고자 했습니다.
* 아래의 yaml 파일에서는 nginx-main 서비스와 nginx-sign 서비스는 k8s-worker1 노드에, tomcat 서비스는 k8s-worker2 노드에, nginx-order 서비스는 k8s-worker3 노드에 예약했습니다.

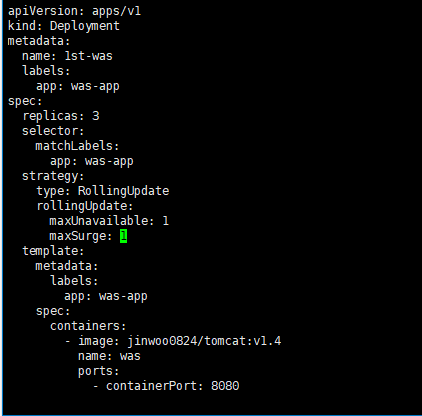
**ex) node-labels.yaml**

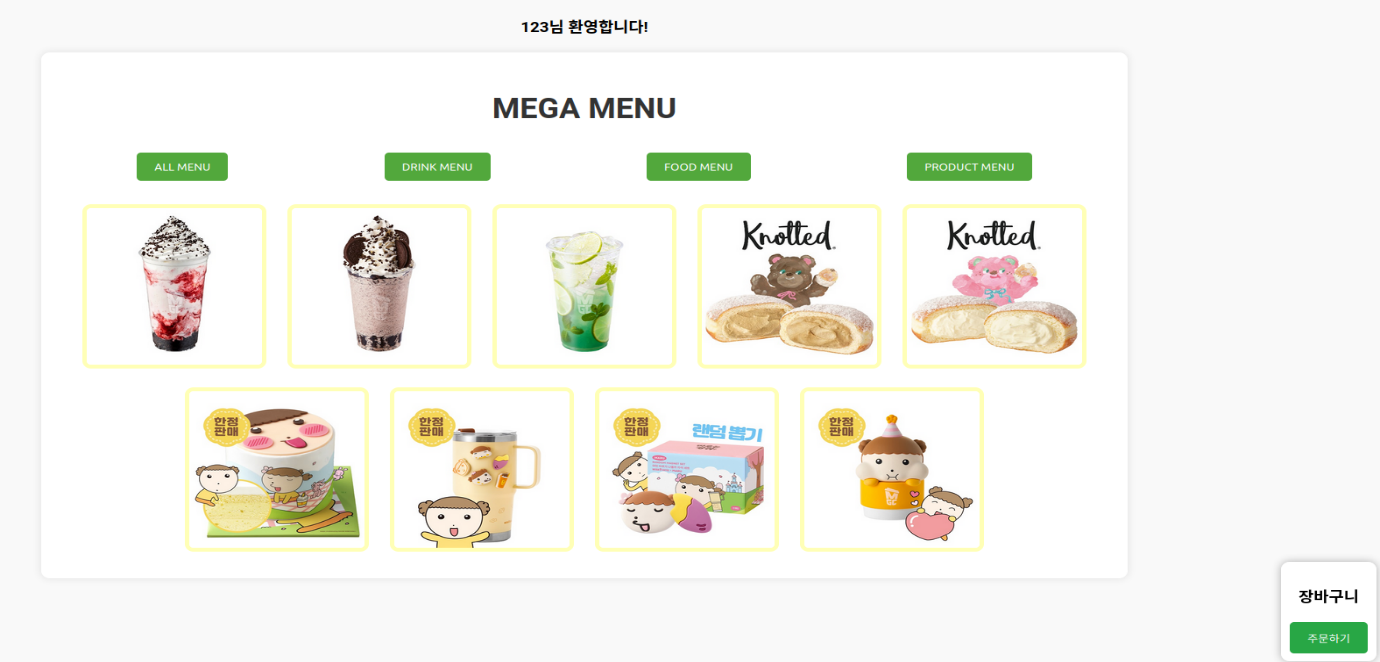


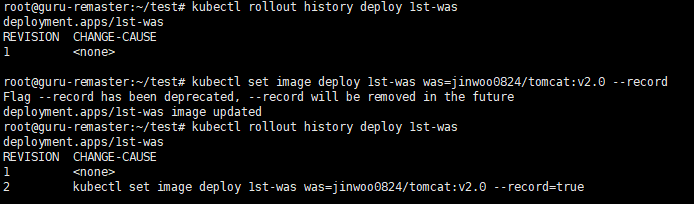
1. **Rolling Update & RollBack**

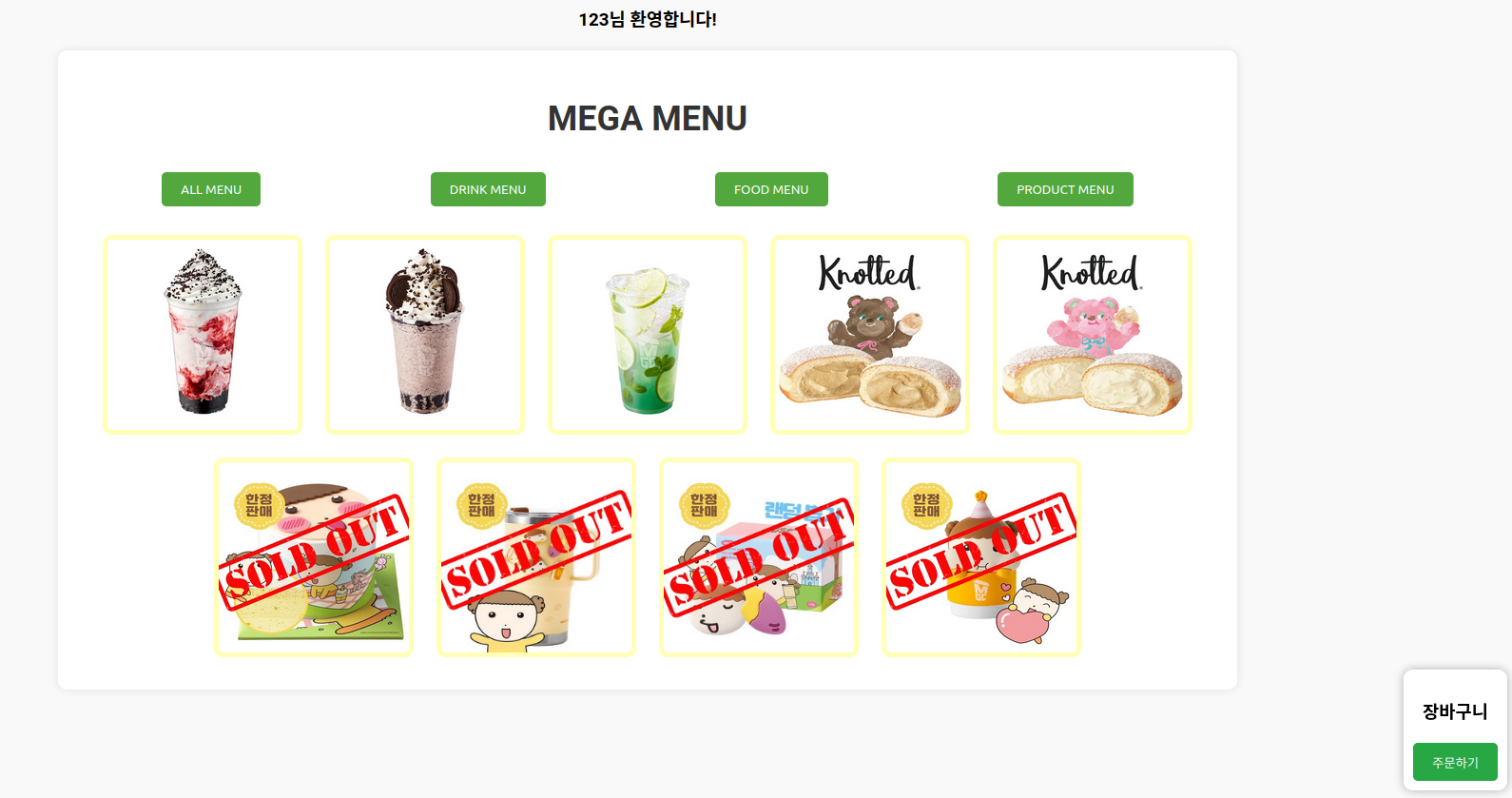
* **Rolling Update 구현**
* 초기 이미지 버전: jinwoo0824/tomcat:v1.4

→ tomcat:v2.0으로 Rolling Update

****

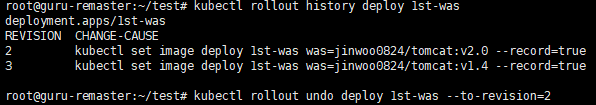
****

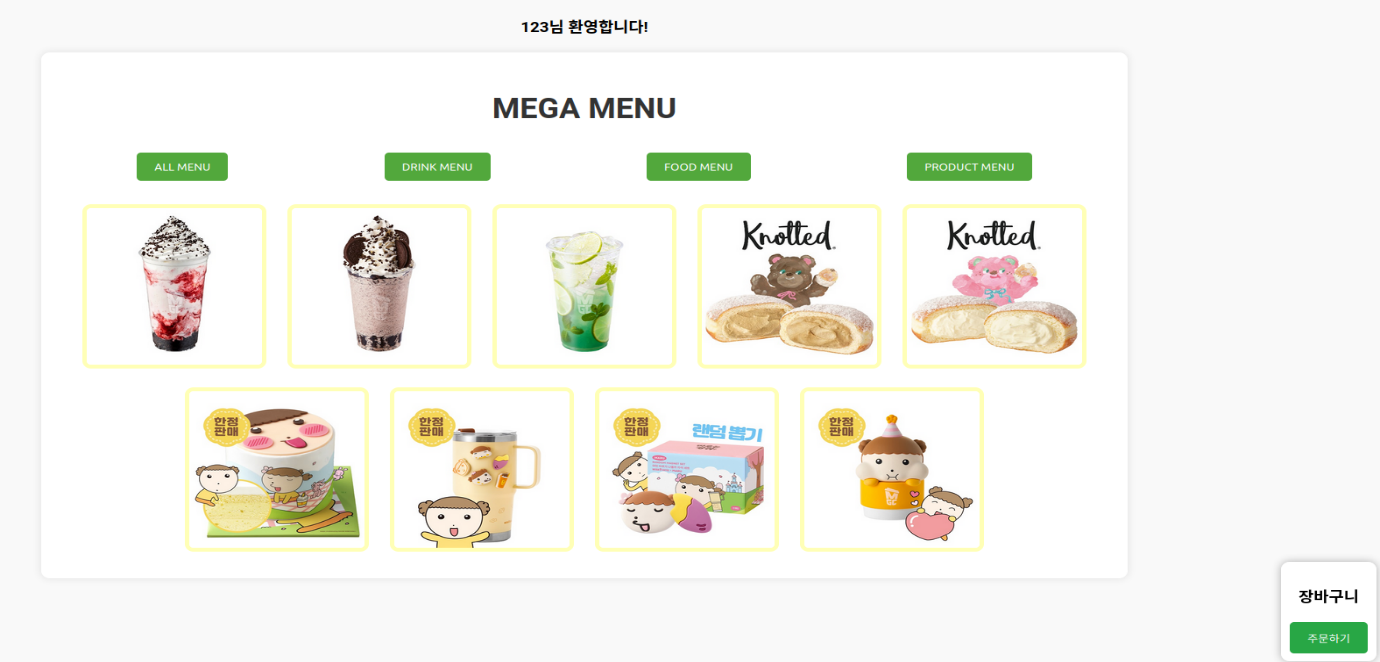
****

****

* **Rollback 구현**
* **현재 이미지 버전:** jinwoo0824/tomcat:v2.0

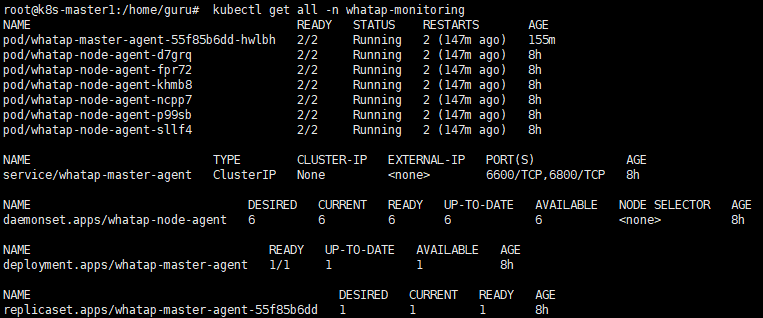
→ tomcat:v1.4 버전으로 RollBack을 진행하였습니다**.**

****

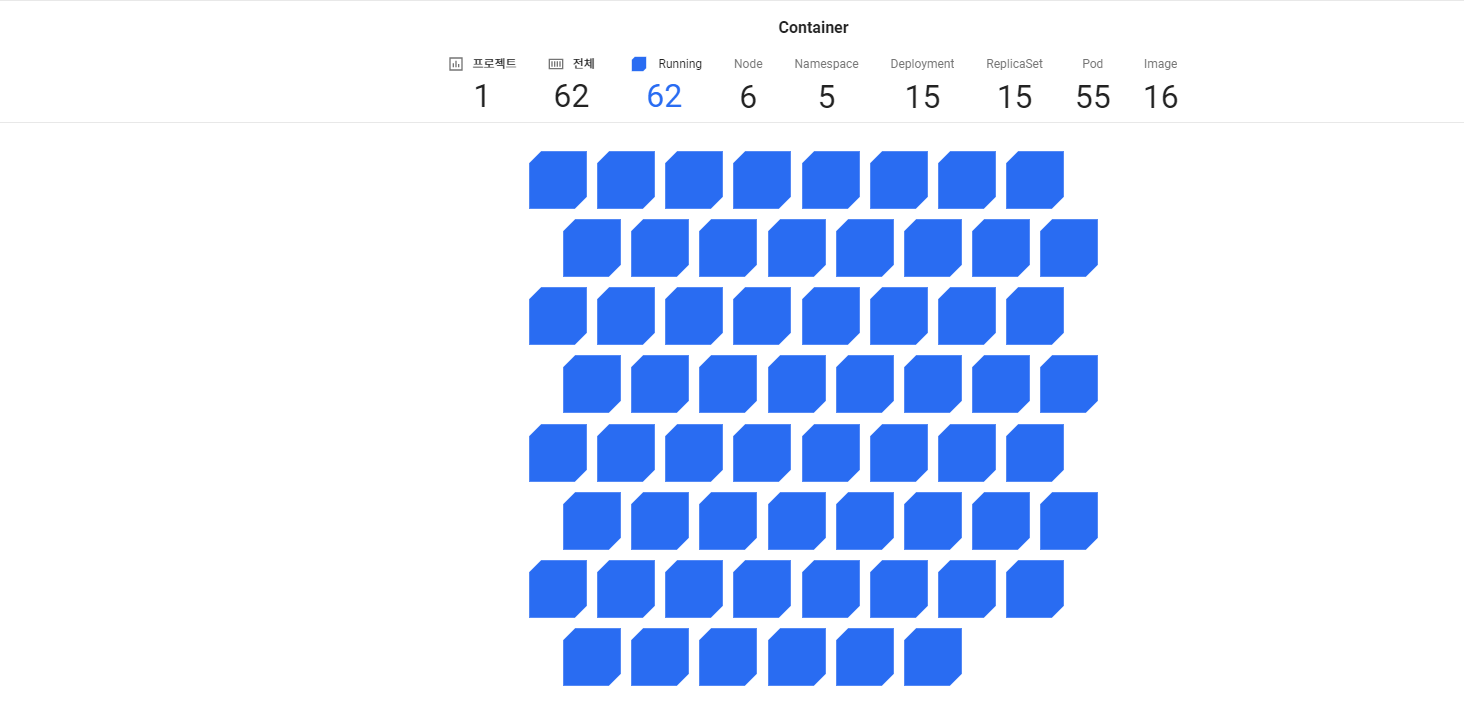
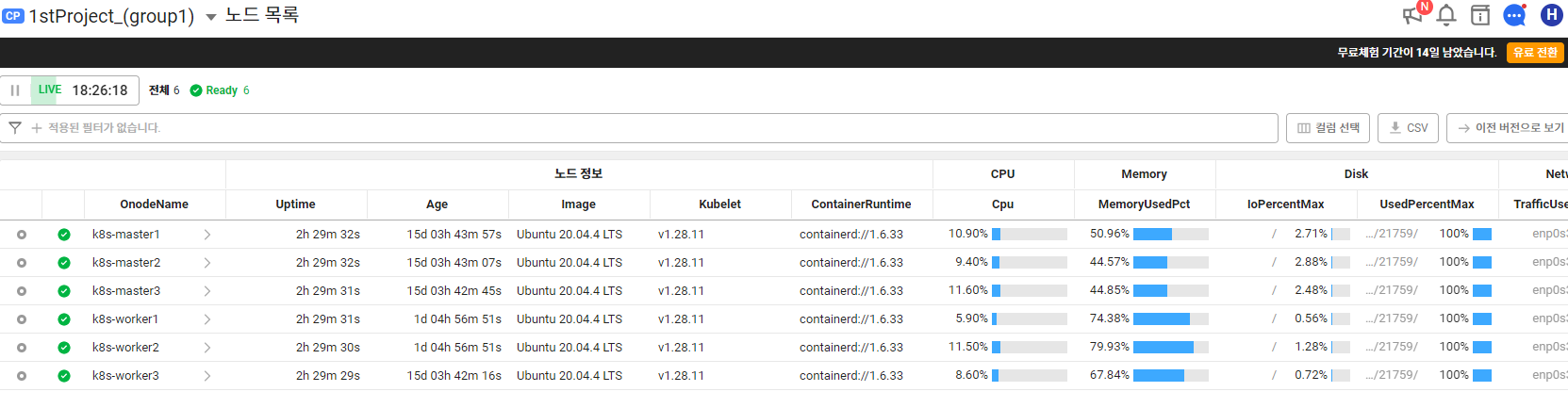
****

1. **WhaTap**

* **저희는 WhaTap monitoring tool을 설치하여 보다 파드들의 상태를 자세히 파악하고자 하였습니다.**

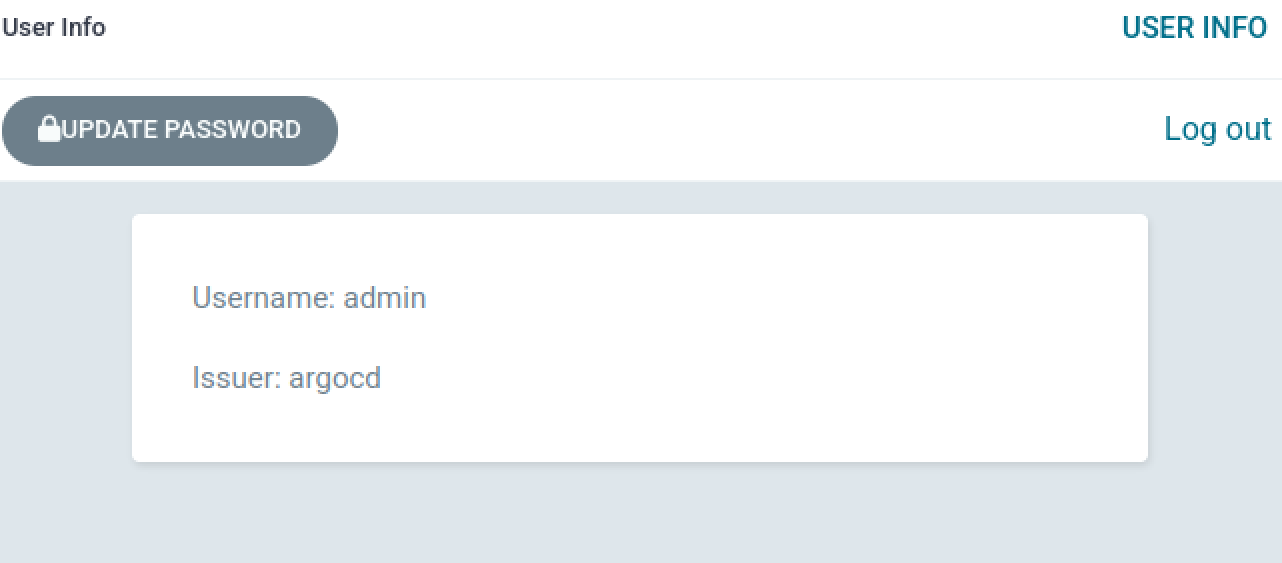
****

* **Node, ContainerMap monitoring**

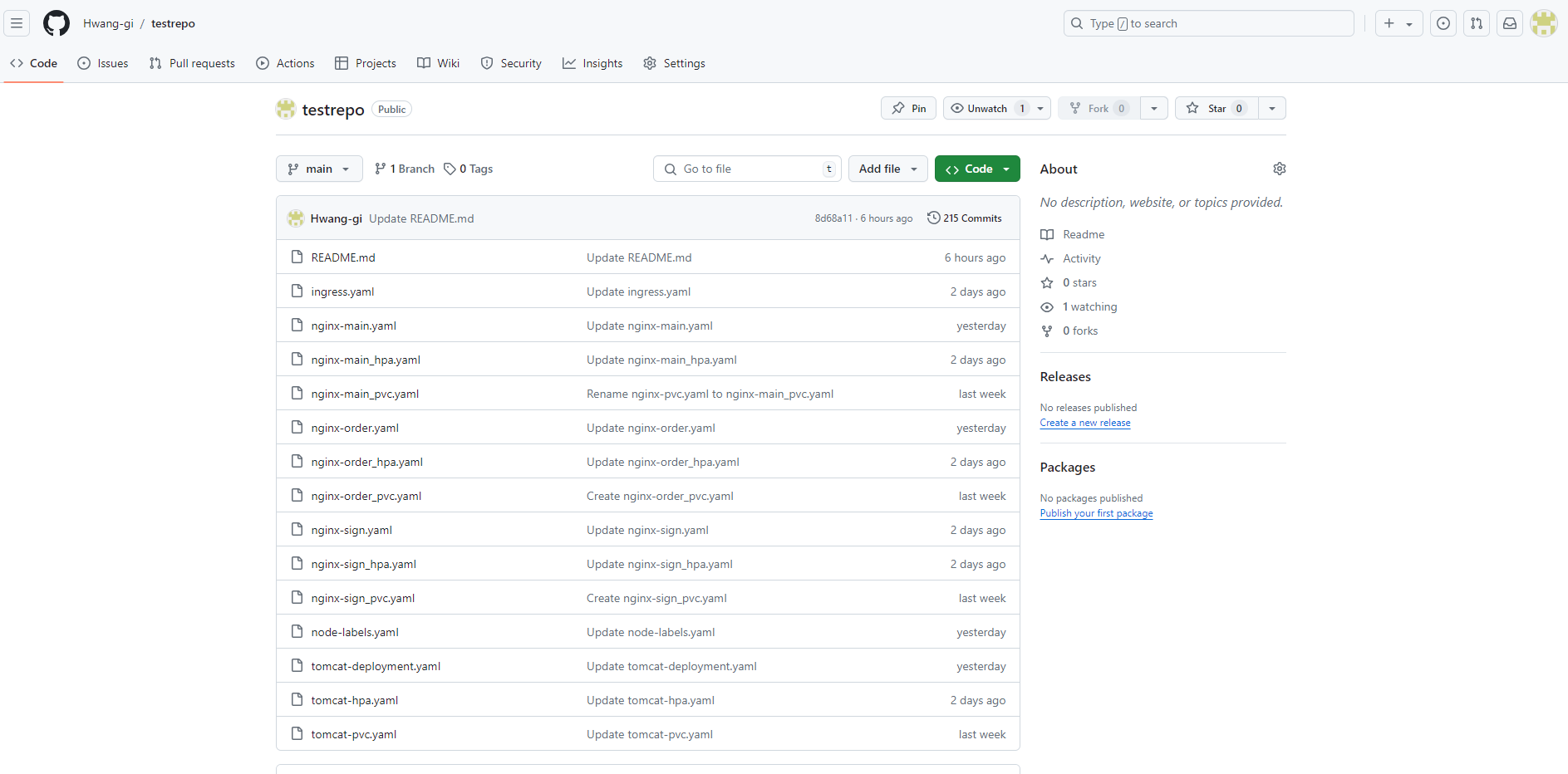
****

1. **Argocd와 Github 연동**

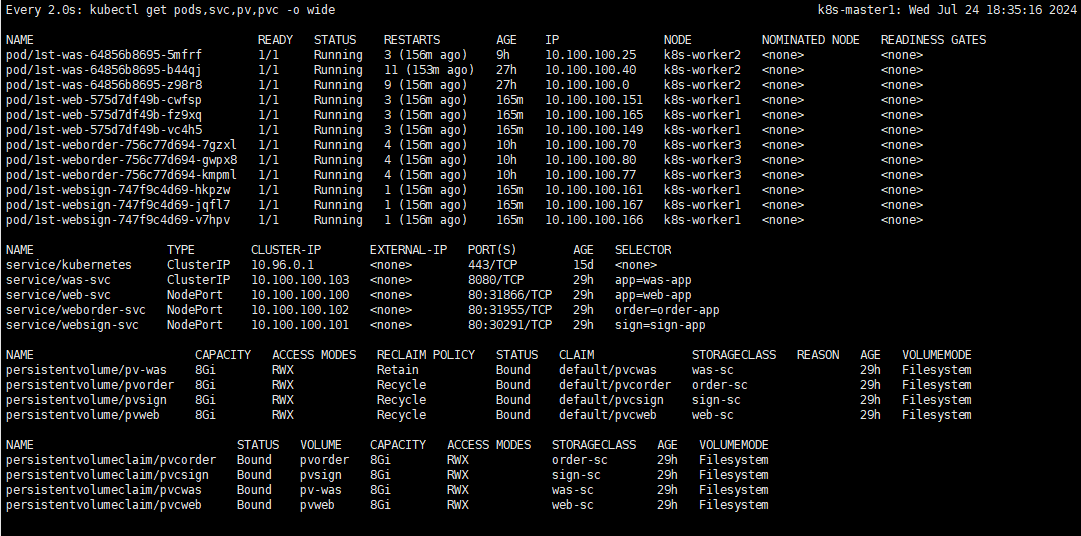
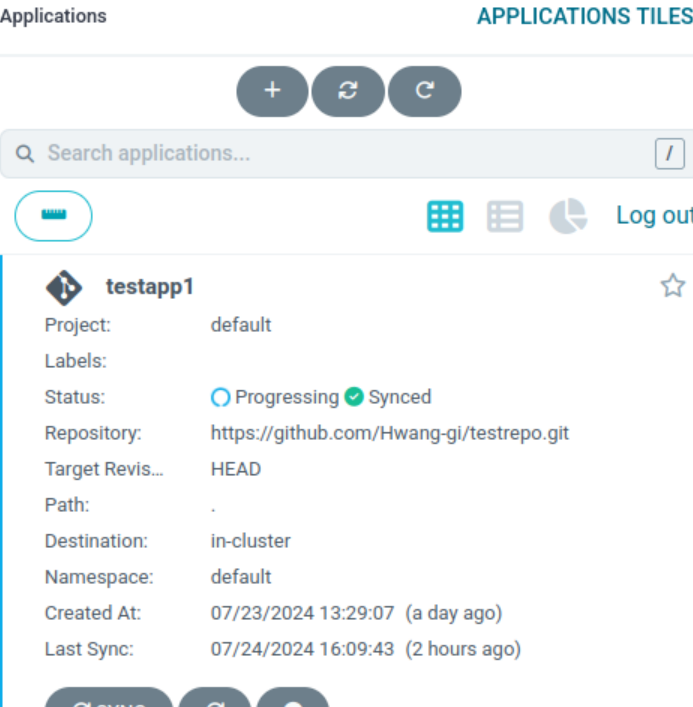
* **Argocd 설치**

****

* **Github 저장소 생성**

****

* **ArgoCD와 Github 연동 성공**

****

1. **Metric Server** **구성**

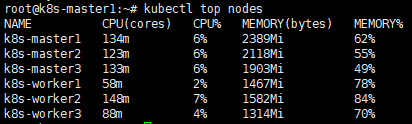
* **역할:** 실시간으로 쿠버네티스 리소스를 모니터링합니다.
* **사용 가능한 명령어:** top 명령어
* **필요 파일**

<https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml>

* metric-server 설치 성공 화면



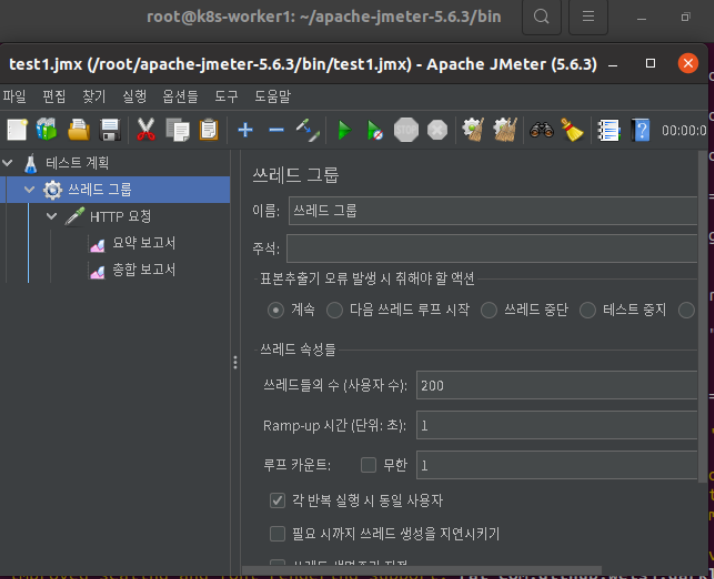
* kubectl top nodes 실행 후 나타나는 노드별 CPU/메모리 사용량

****

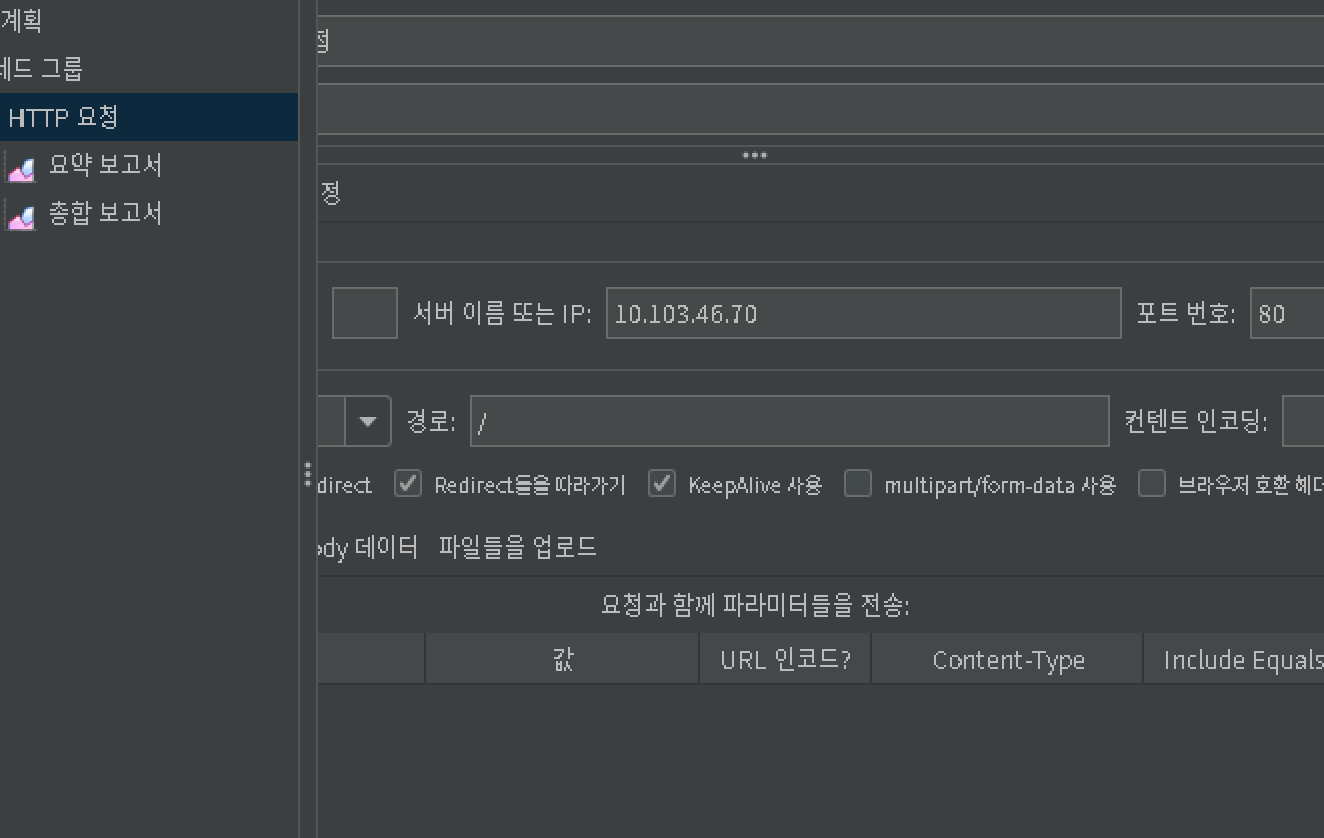
* unknown에서 특정 퍼센트로 바뀌는 hpa 수치

1. **Jmeter 부하 테스트**

* **Jmter 부하 설정 (각 워커 노드별)**
* **부하 횟수 설정**

****

* **Ingress 컨트롤러 생성 후 할당된 로드밸런서 IP:80으로 테스트 진행**

****

**jmeter 부하 성공 화면**

****

### 결론 및 제언(Trouble Shooting)

|  |  |
| --- | --- |
| **구분** | **내용** |
| **결론** | 이번 프로젝트는 3-Tier 웹서비스 구축을 통해 오토스케일링 및 경량화를 달성하고자 했습니다. Kubernetes를 활용하여 Nginx, Tomcat, MySQL을 포함한 아키텍처를 설계 및 구현하였으며, 이를 통해 안정적이고 효율적인 서비스 제공을 목표로 했습니다. 또한, MySQL 데이터베이스의 이중화를 통해 데이터의 가용성과 무결성을 보장하였습니다. 프로젝트 진행 과정에서 다음과 같은 주요 성과를 달성하였습니다: |
| 1. **3-Tier 아키텍처 구현**: 프론트엔드(Nginx), 백엔드(Tomcat), 데이터베이스(MySQL)를 포함한 아키텍처를 성공적으로 설계하고 구현하였습니다. |
| 2. **오토스케일링**: Kubernetes를 활용하여 애플리케이션의 자동 확장성을 구현하였습니다. |
| 3. **데이터베이스 이중화**: MySQL의 이중화를 통해 데이터의 가용성과 무결성을 보장하였습니다. |
| 4. **CI/CD 파이프라인 구축**: ArgoCD와 Github를 연동하여 자동 배포 파이프라인을 설정하였습니다. |
| 5. **성능 최적화 및 모니터링**: WhaTap을 활용한 모니터링 및 JMeter를 이용한 부하 테스트를 통해 시스템 성능을 최적화하였습니다. |
| **제언** | 이번 프로젝트를 통해 여러 가지 중요한 교훈을 얻을 수 있었습니다. 이를 바탕으로 향후 프로젝트 또는 시스템 개선을 위한 몇 가지 제언을 제시합니다: |
| 1. **확장성 향상**: 애플리케이션의 트래픽 증가에 대비하여 더 정교한 오토스케일링 전략을 수립할 필요가 있습니다. 예를 들어, 애플리케이션의 특성에 맞춘 커스텀 매트릭스를 활용한 오토스케일링 설정을 고려해볼 수 있습니다. |
| 2. **보안 강화**: 데이터베이스의 이중화와 함께 보안 측면에서도 강력한 접근 제어 및 데이터 암호화 방안을 추가적으로 구현해야 합니다. |
| 3. **DevOps 문화 정착**: CI/CD 파이프라인의 안정성을 더욱 강화하고, 지속적인 통합 및 배포 프로세스를 팀 전체에 확산시켜 DevOps 문화를 정착시키는 것이 중요합니다. |
| 4. **사용자 경험 개선**: 사용자 피드백을 적극 반영하여 웹페이지의 UI/UX를 개선하고, 더 나은 사용자 경험을 제공하기 위해 지속적으로 업데이트 해야 합니다. |
| 5. **지속적 모니터링**: WhaTap과 같은 모니터링 도구를 지속적으로 활용하여 시스템 상태를 실시간으로 파악하고, 문제 발생 시 신속하게 대응할 수 있는 체계를 마련해야 합니다. |
| **결론** | 이번 프로젝트는 성공적으로 마무리되었지만, 제언된 사항들을 반영하여 지속적으로 개선해 나간다면 더욱 안정적이고 효율적인 시스템을 구축할 수 있을 것입니다. |