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

**K-Digital Training**



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 클라우드 컴퓨팅과 보안솔루션을 활용한  DC 엔지니어 양성 과정 | |  | | |
|  |  | | |  | | |
|  | 1차 프로젝트 기술 보고서 | | |  | | |
|  | - [K8S] 쿠버네티스를 활용한 3-Tier 웹서비스 구축 - | | |  | | |
|  |  | | |  | | |
|  | 2024.07.29.(월) | | |  | | |
|  |  | | |  | | |
|  | 구성원 : 이승환  유원종  김민지  이재혁  김홍섭 | | |  | | |
|  |  |  | | |

1. [프로젝트 개요 3](#_Toc168534865)

[2. 프로젝트 진행 순서 4](#_Toc168534866)

[3. 프로젝트 전체 구성도 4](#_Toc168534867)

4[. 프로젝트 구성 요소 4](#_Toc168534868)

[5. 프로젝트 배경 기술 5](#_Toc168534869)

6[. 프로젝트 추진 체계 5](#_Toc168534870)

[1) 프로젝트 참여 명단 6](#_Toc168534871)

[2) 프로젝트 역할 분담 6](#_Toc168534872)

7[. 프로젝트 시나리오 6](#_Toc168534873)

8. [On Premise 인프라 기본 설정 7](#_Toc168534877)

9. [index 페이지 8](#_Toc168534878)

10. [Docker File 10](#_Toc168534884)

11. [Yaml File 12](#_Toc168534886)

12. [MariaDB 이중화 23](#_Toc168534887)

13. [Ingress 27](#_Toc168534889)

14. [Tomcat Server 28](#_Toc168534890)

15. [WhaTap 29](#_Toc168534892)

16. [ArgoCD – GitHub 연동 31](#_Toc168534893)

17. [Jmeter 37](#_Toc168534897)

18. [Rolling Update & Roll Back 38](#_Toc168534898)

19. multi master node 환경을 통한 가용성 확보 ……………………………………………………………………………39

|  |  |
| --- | --- |
| 프로젝트 개요 | |
| **과정명 및 팀명** | |
| 과정명 | 클라우드 컴퓨팅과 보안솔루션을 활용한 DC 엔지니어 양성 과정 |
| 교육기관 | 한국정보교육원 |
| 팀 명 | 3조 |

**1) 프로젝트 소개**

- 배경: 온라인 신발 쇼핑몰 신규 오픈 행사를 대비한 서비스 점검

- 주제: kubernetes를 활용한 3Tier 웹 서비스 구축

- 목적

• On-Premise 환경에서 Nginx-Tomcat-MariaDB 연결 구성

• Dockerfile 이미지 경량화

• service, Auto Scaling(hpa) 등이 포함된 YAML 파일을 적용한 컨테이너 파드 생성

• Github-ArgoCD 연동을 통한 배포

**2) 네트워크 설계 및 관리 운용**

• 3Tier 아키텍처 설계 및 개발

• Dockerfile 이미지 설계 및 생성

• 파드 설계(service, ingress, autoscaling 등) 및 컨테이너 구축

• WEB-WAS-DB로 구현

• DB 이중화 구성

• Rolling Update 및 롤백 구성

• MetalLB 구성

**2. 프로젝트 진행 순서**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 주제 선정  및 분석 | ▶ | 관리대장 및  구성도 작성 | ▶ | 서버 구현 | ▶ | 서버 테스트 및 연동 | ▶ | 서버  운영 |

**3. 프로젝트 전체 구성도**

|  |
| --- |
|  |

**4. 프로젝트 구성 요소**

**< nginx 웹 서버 >**

• index.jsp 개발을 통한 main page 개발

• ingress service를 통해 여러 domain에 접근 가능

• 클라이언트가 입력한 데이터(id, password)를 tomcat으로 전송하고, tomcat에서 전달받은 데이터를 화면에 표시해서 클라이언트에게 제공

**< tomcat 서버 >**

• 클라이언트에게 동적인 페이지 제공

• 웹 서버에서 전달받은 데이터를 DB에게 전달하고, DB에서 전달받은 데이터를 웹 서버에게 전달

**< DB 서버 >**

• tomcat 서버에서 전달받은 데이터를 mariaDB를 이용해서 table 형태로 저장

**5. 프로젝트 배경 기술**

|  |  |  |
| --- | --- | --- |
| **연번** | **기술** | **내용** |
| 1 | Kubernetes | 컨테이너화된 워크로드와 서비스를 관리하기 위한 이식성이 있는 확장 가능한 오픈소스 플랫폼 |
| 2 | Docker | 컨테이너를 구축, 배포, 실행, 업데이트 및 관리할 수 있게 해주는 오픈소스 플랫폼 |
| 3 | Nginx web | 가벼우면서도 강력한 프로그램을 목표로 하는 오픈소스 웹 서버 소프트웨어이며 HTTP와 리버스 프록시, IMAP/POP3 등의 서버 구동 |
| 4 | Tomcat | Java 기반의 서블릿 컨테이너이자 웹서버 |
| 5 | MariaDB | 오픈소스 관계형 데이터베이스 관리 시스템 |
| 6 | ArgoCD | 쿠버네티스 환경에서의 애플리케이션 배포와 관리를 지원하는 도구 |
| 7 | Github | 빠른 속도와 분상형 저장소인 git들이 모여 있는 곳 |
| 8 | Jmeter | Java로 만들어진 웹 어플리케이션 성능 테스트 오픈소스 |
| 9 | Whatap | 쿠버네티스 모니터링 |

**6. 프로젝트 추진 체계**

1) 프로젝트 참여 명단:

|  |  |
| --- | --- |
| **팀장** | 이승환 |
| **기술이사** | 유원종 |
| **팀원** | 김민지, 이재혁, 김홍섭 |

2) 프로젝트 역할 분담

|  |  |
| --- | --- |
| **이름** | **역할** |
| 이승환 | 3-Tier 아키텍쳐 연동, Argocd를 통한 배포 |
| 유원종 | 백엔드 설계 및 구축, Dockerfile 생성, DB이중화 구현 |
| 김민지 | DB 설계 및 구축, 모니터링(와탭) , 이미지 경량화 설계 |
| 이재혁 | 토폴로지 설계 및 인프라 구축, Dockerfile 경량화, Yaml 파일 작성 |
| 김홍섭 | 프론트엔드 개발, 프론트엔드 연동, JMeter를 통한 부하 테스트 |

**7. 프로젝트 시나리오**

|  |
| --- |
| 온라인 신발 쇼핑몰 신규 오픈으로 인한 할인 행사를 진행하려 한다. 현재 신규 오픈까지 3일 정도를 남기고 있으며, 많은 고객들이 몰릴 것으로 예상하여 마지막 쇼핑몰 점검을 하려 한다.  우선 고객들이 접속할 웹 페이지, 네트워크 등 쇼핑몰 운영에 문제가 없는지 확인하기 위해 몇 가지 테스트를 진행한다.  1. WEB서버와 WAS서버, DB서버 연동 체크  WEB서버와 WAS서버의 연동이 잘 되어 있는지 확인하려면 원하는 신발 색상을 고르기 위해 Color에서 색상을 골랐을 때 요청한 색상의 신발들이 잘 표시가 되는지 확인한다.  WAS서버와 DB서버의 연동 확인은 고객들이 회원 가입하고 메인 페이지에 고객의 이름이 출력이 되는지 확인한다.  2. DB 이중화 체크  현재 쇼핑몰은 혹시 모를 문제를 대비해 DB 이중화를 해둔 상태이다. 이를 확인하기 위해 master DB의 작동을 강제로 멈추게 한다. master DB가 작동을 멈췄을 때, slave DB가 master DB의 역할을 잘 대체하여 작동하는지 확인한다.  3. 가용성 테스트  오픈 할인 행사로 인해 고객들이 얼마나 몰릴지 예상하지 못하는 상황이다. 대량의 통신량이 발생하게 되면 마스터 노드는 부하를 많이 받아 장애 발생 가능성이 커지게 되기 때문에 우리 쇼핑몰은 고가용성 클러스터를 구성했다. 현재 마스터 노드를 2개 더 준비하여 3개까지 준비 해놓은 상태이다.  테스트를 위해 현재 작동 중인 마스터 노드 1을 셧다운 시킨다. 셧다운 되고 나서 두 번째 마스터 노드가 마스터 노드1를 대신해 잘 작동하는지 확인한다.  4. 부하 테스트  많은 고객들이 쇼핑몰에 동시 접속할 경우에도 서버가 버틸 수 있어야 한다. 버티지 못 할 정도로 부하가 올 경우를 대비하여 어느 정도 부하가 왔을 때 자동으로 서버가 확장이 될 수 있도록 설정해두었다.  부하 테스트를 위해 JMeter 프로그램을 준비하여 직접 부하를 주면서 설정한 값에 따라 서버 확장이 잘 이루어 지는지 Whatap 프로그램으로 확인한다.  5. 롤링 업데이트, 자동화 배포 테스트  쇼핑몰 오픈을 하고 고객들이 쇼핑을 하다 보면 사이즈 및 색상, 재고 등이 품절될 수 있다. 만약 품절된 상품이 바로 업데이트가 되지 않고 품절된 상품이 재고가 있는 상태로 노출이 될 경우 엄청난 문제와 혼선이 생길 수 밖에 없다. 그렇기 때문에 롤링 업데이트와 자동화 배포가 잘 이루어 지는지 확인해야 한다.  Github, ArgoCD를 통해 롤링 업데이트, 자동화 배포가 동시에 잘 이루어지는지 확인한다.  이런 테스트를 통해 쇼핑몰이 오픈하기 전에 생길 수 있는 문제들을 미리 대비하고 방지해야 문제 발생 확률을 낮추고 발생하더라도 그 즉시 해결이 가능하다. 현재 테스트를 진행을 위한 준비까지 다 마친 상황이다. |

**8. On Premise 인프라 기본 설정**

**1) node 기본 정보**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **서버** | **IP** | **서브넷마스크** | **게이트웨이** | **VIP** | **Label** |
| K8s-masterMK1 | 10.100.0.131 | 255.255.255.0 | 10.100.0.1 | 10.100.0.120 | - |
| K8s-masterMK2 | 10.100.0.132 | 10.100.0.120 | - |
| K8s-masterMK3 | 10.100.0.133 | 10.100.0.120 | - |
| K8s-worker1 | 10.100.0.141 | - | type: web |
| K8s-worker2 | 10.100.0.142 | - | type: was |
| K8s-worker3r | 10.100.0.143 | 10.100.0.150 | - |
| K8s-worker4 | 10.100.0.144 | 10.100.0.150 | - |
| K8s-lb1 | 10.100.0.121 | - | - |
| K8s-lb2 | 10.100.0.122 | - | - |

**2) service 정보**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **이름** | **레이블** | **서비스** | **Cluster IP** | **포트** | **pvc-accessMode** |
| svc-web | app: web | NodePort | 10.100.120.1 | 80:30100 | ReadWriteMany |

**3) WEB 정보**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Deploy name** | **Label** | **Resource** | | | |
| **Request** | | **Limit** | |
| **Cpu** | **Memory** | **Cpu** | **Memory** |
| web | | 200m | 200Mi | 200m | 200Mi |

**4) ingress 정보**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Path** | **Service** | **Port** | **Ingress class** | **Host** |
| / | svc | 80 | nginx | yas24.com |

**9. Index 페이지**

1) signup.jsp: 회원가입 페이지

|  |  |
| --- | --- |
| **jsp 파일** |  |
| **회원가입 페이지** |  |

2) joinaction.jsp: 회원가입 정보 처리

→ 회원가입 페이지에서 받은 정보를 입력된 DB에 저장

|  |
| --- |
|  |

3) login.jsp: 로그인 페이지

|  |  |
| --- | --- |
| **jsp 파일** |  |
| **회원가입 페이지** |  |

4) loginaction.jsp: 로그인 정보 처리

|  |  |
| --- | --- |
| **jsp 파일** |  |
| **내용** | 가. 로그인 페이지는 정보를 DB에서 가지고 오기위해 loginaction.jsp 호출  나. loginaction이 입력된 DB 정보를 바탕으로 해당하는 정보 불러오기  다. DB에서 전달받은 정보와 클라이언트가 입력한 로그인 정보(id, password)가 일치하면 로그인 성공으로 처리하고, 입력된 ID를 메인 페이지로 보내서 ID 출력 |

5) index.jsp: 로그인 성공 시 ID 표시

|  |
| --- |
|  |

**10. Dockerfile**

1) Nginx Dockerfile

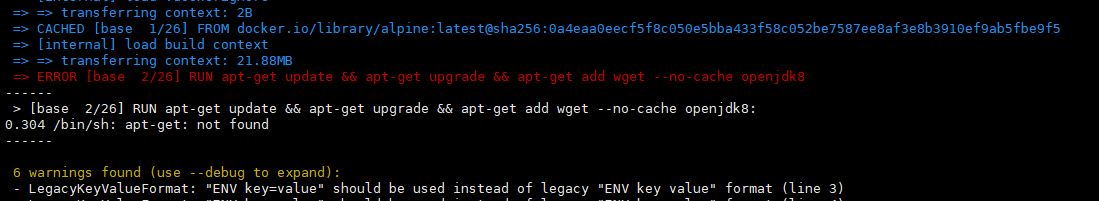
|  |  |
| --- | --- |
| **Docker**  **file** |  |
| **내용** | 가. 이미지  : Alpine base image를 사용해서 경량화 진행  나. tomcat과의 연동을 위해 nginx.conf 해당하는 위치로 복사 |

3) Tomcat Dockerfile

|  |  |
| --- | --- |
| **Docker**  **file** |  |
| **내용** | 가. DB와 연동을 위해 DB 정보를 입력해둔 context.xml, server.xml 파일을 해당 경로로 복사  나. DB와 연동을 위해 미리 받아 뒀던 mariadb-java-client 파일을 해당 경로로 복사  다. Alpine base image를 사용해서 기본 이미지의 사이즈를 경감하고, multi-stage 빌드를 통해서 이미지 경량화 추가 진행 |

4) Trouble Shooting

가. tomcat dockerfile이 빌드가 되지 않는 문제



|  |  |
| --- | --- |
| **원인** | **해결방법** |
| 기존에 사용하던 ubuntu 기반의 명령어가 alpine OS에서는 작동하지 않음 | 기존에 ubuntu 기반 운영체제에서 사용하던 명령어 대신 alpine에서 사용하는 명령어인 apk upgrade, apk add –no-cache등을 사용해서 문제를 해결함 |

**11. Yaml File**

**<WEB>**

1) Web-deploy.yaml

|  |  |  |
| --- | --- | --- |
| **Yaml 파일** | **구분** | **내용** |
|  | apiVersion | Deployment의 api버전인 apps/v1을 입력 |
| kind | 배포를 위해 Deployment로 생성 |
| annotations | version history를 확인하기 위해Kubernetes.io  /change-cause 값 입력 |
| replicas | 초기 desired 개수 설정 |
| progressDeadline  Seconds | 정해진 시간동안 rolling update가 되지 않으면 rollback |
| revisionHistory  Limit | change-cause 값으로 입력해둔 history 저장 개수 지정 |
| rollingUpdate | rolling update 시 한번에 추가 및 제거할 deployment의 수 지정 |
| Selector.  matchLabels | service와 연동을 위해 label을 app: web으로 설정 |
| Template.spec.  nodeSelector | worker node의 부하를 막기 위해 affinity를 설정  \* Nginx deployment는 type: web label 값을 가진 worker node 1번에만 생성 |
| Image | container image 설정 |
| Port | nginx container에서 사용할 포트를 지정  (http의 기본 포트인 80으로 지정) |
|  | resources | deployment가 사용할 자원 지정  (Deployment 하나가 사용할 cpu는 250m, memory는 250Mi. request와 limits 값을 동일하게 줘서 guaranteed 클래스로 지정) |
| • initialDelaySeconds: pod 생성 후 15초 후부터 livenessprobe 적용  • periodSeconds: 5초마다 pod에게 ping을 보내게 설정  • timeoutSeconds: 2초 안에 응답이 있으면 성공으로 취급  • successThreshold: 1번만 응답이 있으면 성공으로 취급하게 설정  • failureThreshold: 응답이 없을 때 2번 반복해서 ping을 보내게 설정 |
| volumeMounts.  mountPath | mount 시켜줄 경로 지정 |
| volumes.  name | volume mount 해줄 volume 이름 설정 |
| volumes.  persistentVolumeClaim | volume mount 할 때 사용할 pvc 설정 |

2) web-service.yaml

|  |  |  |
| --- | --- | --- |
| **Yaml 파일** | **구분** | **내용** |
|  | apiVersion | service의 apiVersion인 v1으로 설정 |
| kind | nginx deployment에 단일진입점을 제공하기 위해 service 생성 |
| spec.type | 외부에서의 접근을 허용하기 위해 NodePort 사용 |
| clusterIP | 단일진입점 IP 값 설정 |
| port | NodePort의 port에서 nginx 서버의 port로 port forwarding 할 수 있도록 설정 |
| selector | app: web label을 가진 deployment들에게 단일진입점을 제공하도록 설정 |

3) web-hpa.yaml

|  |  |  |
| --- | --- | --- |
| **Yaml 파일** | **구분** | **내용** |
|  | maxReplicas | 최대로 생성되는 pod 개수 설정 |
| minReplicas | 최소로 생성되는 pod 개수 설정 |
| scaleTargetRef | 적용할 deploy 정보 기재 |
| targetCPUUtilizationPercentage | CPU 사용률 목표 설정 |

4) web-pv.yaml / web-pvc.yaml

|  |  |
| --- | --- |
|  |  |
| **pv.yaml** | **pvc.yaml** |

**<WAS>**

1) was-deploy.yaml

|  |  |  |
| --- | --- | --- |
| **Yaml 파일** | **구분** | **내용** |
|  | apiVersion | Deployment의 api버전인 apps/v1을 입력 |
| kind | 배포를 위해 Deployment로 생성 |
| annotations | version history를 확인하기 위해Kubernetes.io  /change-cause 값 입력 |
| replicas | 초기 desired 개수 설정 |
| progressDeadline  Seconds | 정해진 시간동안 rolling update가 되지 않으면 rollback |
| revisionHistory  Limit | change-cause 값으로 입력해둔 history 저장 개수 지정 |
| rollingUpdate | rolling update 시 한번에 추가 및 제거할 deployment의 수 지정 |
| Selector.  matchLabels | service와 연동을 위해 label을 app: was로 설정 |
| Template.spec.  nodeSelector | worker node의 부하를 막기 위해 affinity를 설정  \* was deployment는 type: was label 값을 가진 worker node 2번에만 생성 |
| Image | container image 설정 |
| Port | tomcat container에서 사용할 포트를 지정 |
|  | resources | deployment가 사용할 자원 지정  (Deployment 하나가 사용할 cpu는 300m, memory는 300Mi. request와 limits 값을 동일하게 줘서 guaranteed 클래스로 지정) |
| • initialDelaySeconds: pod 생성 후 15초 후부터 livenessprobe 적용  • periodSeconds: 5초마다 pod에게 ping을 보내게 설정  • timeoutSeconds: 1초 안에 응답이 있으면 성공으로 취급  • successThreshold: 1번만 응답이 있으면 성공으로 취급하게 설정  • failureThreshold: 응답이 없을 때 2번 반복해서 ping을 보내게 설정 |
| volumeMounts.  mountPath | mount 시켜줄 경로 지정 |
| volumes.  name | volume mount 해줄 volume 이름 설정 |
| volumes.  persistentVolumeClaim | volume mount 할 때 사용할 pvc 설정 |

2) was-service.yaml

|  |  |  |
| --- | --- | --- |
| **Yaml 파일** | **구분** | **내용** |
|  | apiVersion | service의 apiVersion인 v1으로 설정 |
| kind | was deployment에 단일진입점을 제공하기 위해 service 생성 |
| clusterIP | 단일진입점 IP 값 설정 |
| port | NodePort의 port에서 nginx 서버의 port로 port forwarding 할 수 있도록 설정 |
| selector | app: web label을 가진 deployment들에게 단일진입점을 제공하도록 설정 |

3) was-hpa.yaml

|  |  |  |
| --- | --- | --- |
| **Yaml 파일** | **구분** | **내용** |
|  | maxReplicas | 최대로 생성되는 pod 개수 설정 |
| minReplicas | 최소로 생성되는 pod 개수 설정 |
| scaleTargetRef | 적용할 deploy 정보 기재 |
| targetCPUUtilizationPercentage | CPU 사용률 목표 설정 |

4) was-pv.yaml / was-pvc.yaml

|  |  |
| --- | --- |
|  |  |
| **pv.yaml** | **pvc.yaml** |

**<DB>**

1) db-deploy.yaml

|  |  |  |
| --- | --- | --- |
| **Yaml 파일** | **구분** | **내용** |
|  | apiVersion | Pod의 api버전인 apps/v1을 입력 |
| kind | Pod명을 보장하기 위해 StatefulSet으로 생성 |
| replicas | 초기 desired 개수 설정 |
| Selector.  matchLabels | service와 연동을 위해 label을 app: mariadb로 설정 |
| Template.spec.  nodeSelector | worker node의 부하를 막기 위해 affinity를 설정  \* mariadb deployment는 type: db label 값을 가진 worker node 3번에만 생성 |
| Image | container image 설정 |
| env | 환경변수 설정  (데이터베이스 접근 시 비밀번호 설정 및 데이터베이스 자동 생성) |
| Port | db container에서 사용할 포트를 지정 |
|  | resources | deployment가 사용할 자원 지정  (Deployment 하나가 사용할 cpu는 200m, memory는 200Mi. request와 limits 값을 동일하게 줘서 guaranteed 클래스로 지정) |
| livenessProbe | Podrk 살아있는지 확인하는 방법 지정 |
| • initialDelaySeconds: pod 생성 후 15초 후부터 livenessprobe 적용  • periodSeconds: 15초마다 pod에게 ping을 보내게 설정  • timeoutSeconds: 1초 안에 응답이 있으면 성공으로 취급  • successThreshold: 1번만 응답이 있으면 성공으로 취급하게 설정  • failureThreshold: 응답이 없을 때 3번 반복해서 ping을 보내게 설정 |
| volumeMounts.  mountPath | mount 시켜줄 경로 지정 |
| volumes.  name | volume mount 해줄 volume 이름 설정 |
| volumes.  persistentVolumeClaim | volume mount 할 때 사용할 pvc 설정 |

2) db-service.yaml

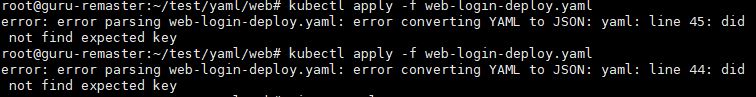
|  |  |  |
| --- | --- | --- |
| **Yaml 파일** | **구분** | **내용** |
|  | apiVersion | service의 apiVersion인 v1으로 설정 |
| kind | was deployment에 단일진입점을 제공하기 위해 service 생성 |
| clusterIP | 단일진입점 IP 값 설정 |
| port | localhost 서버에서 db컨테이너 port로 port forwarding 할 수 있도록 설정 |
| selector | app: mariadb label을 가진 deployment들에게 단일진입점을 제공하도록 설정 |

3) db-pv.yaml / db-pvc.yaml

|  |  |
| --- | --- |
|  |  |
| **pv.yaml** | **pvc.yaml** |

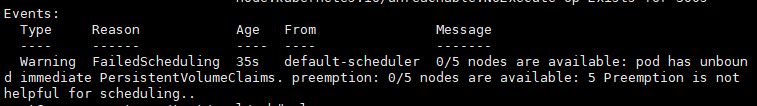
**<Trouble Shooting>**

1) yaml 파일 실행 시 알 수 없는 이유로 실행이 되지 않는 문제



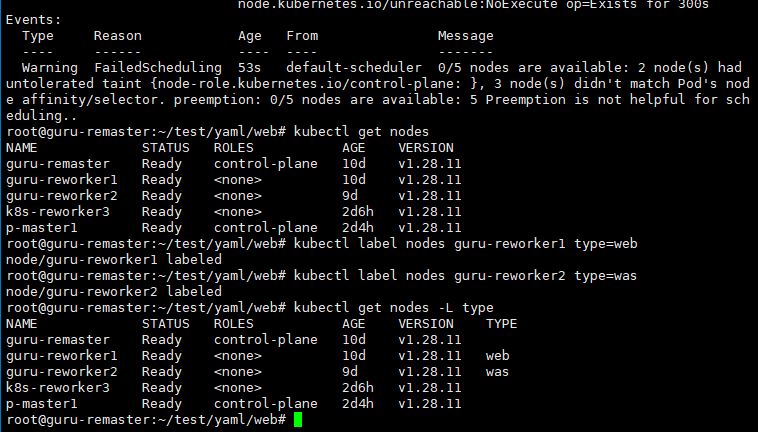
|  |  |
| --- | --- |
| **원인** | **해결방법** |
| Yaml 파일에 보이지 않는 빈칸이 입력되어 있음 | 보이지 않는 빈칸을 모두 삭제 후 다시 입력하여 해결함 |

2) 생성한 deploy가 계속 pending이 뜨는 현상



|  |  |
| --- | --- |
| **원인** | **해결방법** |
| Deploy에 해당된 pvc가 작동하지 않아 scheduling이 되지 않음  → 해당하는 pvc도 pending 상태  (project 초기에는 하나의 커다란 pv에 여러 개의 pvc를 할당하려고 디자인함) | 1. 여러 개의 작은 pv를 구성  2. 각각 하나의 pvc로 할당하도록 토폴로지를 전체적으로 수정  3. 결과적으로 pvc가 pending이 뜨던 문제도 해결되고 deploy도 정상적으로 생성 |

3) 2)번에서의 문제를 해결한 후에 deploy가 또 다시 pending이 뜨는 현상



|  |  |
| --- | --- |
| **원인** | **해결방법** |
| worker node의 부하를 분산시키려고 추가한 nodeselector 설정이 node의 label 설정과 맞지 않아 scheduling에 실패함 | 1. worker node의 알맞은 label을 추가  2. deploy의 affinity 설정과 일치하게 설정 3. Label을 추가하자 마자 deploy가 알맞은 node에서 생성되는 것을 확인 가능. |

4) 2)번과 3)번에서 문제를 해결했음에도 일부 deploy는 running이 잘 되지만 일부 deploy는 pending이 뜨던 문제



|  |  |
| --- | --- |
| **원인** | **해결방법** |
| deploy의 resource 값에 너무 과한 limit 값과 request값을 설정해 scheduling 된 worker node에 deploy가 생성되지 않음 | worker node의 CPU와 Memory가 감당할 수 있는 적절한 resource 값으로 deploy의 resource 재설정 |

**12. MariaDB 이중화**

: 두 DB에 정보가 같도록 동기화

1) Master DB – Slave DB 연동

**<master DB>**

가. /etc/mysql/mariadb.conf.d/50-server.출 파일 수정

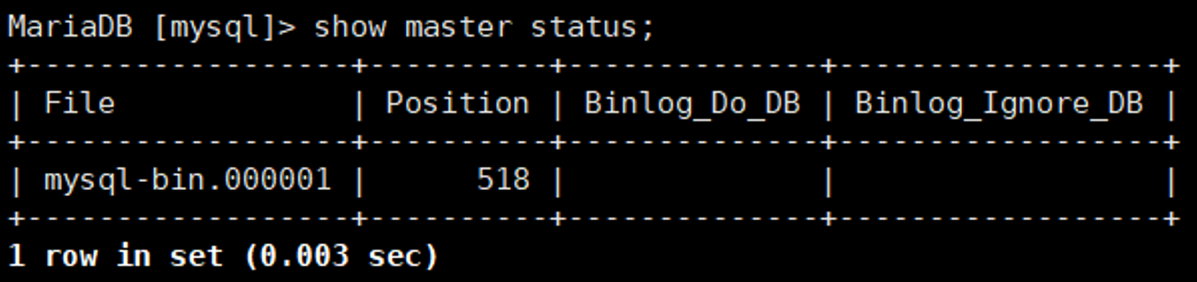
|  |  |
| --- | --- |
|  | - log-bin: 마스터에서 저장할 log 파일 이름  - server-id: master와 slave를 구분하기 위한 id number  - expire\_logs\_days: log파일 보관 기간 |

나. master db에서 mariadb 실행

- 복제 슬레이브 관한 권한 부여

- mysql db 이동

- 상태확인



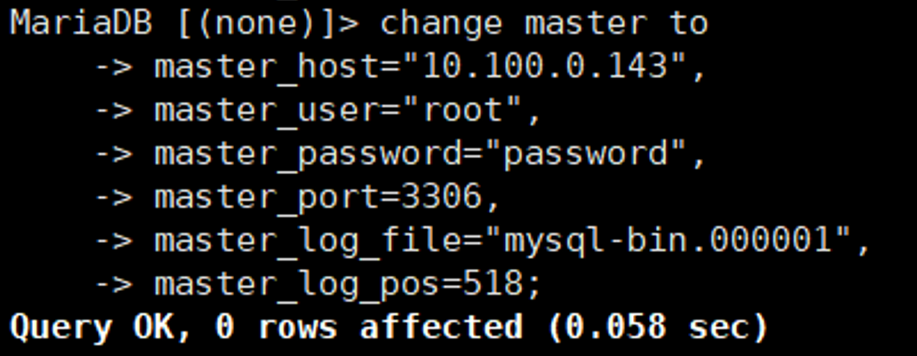
**<slave DB>**

가. /etc/mysql/mariadb.conf.d/50-server.cnf 파일 수정

|  |  |
| --- | --- |
|  | - log-bin: 마스터에서 저장할 log 파일 이름  - server-id: master와 slave를 구분하기 위한 id number  - expire\_logs\_days: log파일 보관 기간 |

나. mariadb 실행

- 복제 설정 변경



- slave 시작

- 상태확인

**<연동 확인>**

가. master db에서 table에 data 입력



나. slave db에 같은 data 확인



2) Keepalived 적용

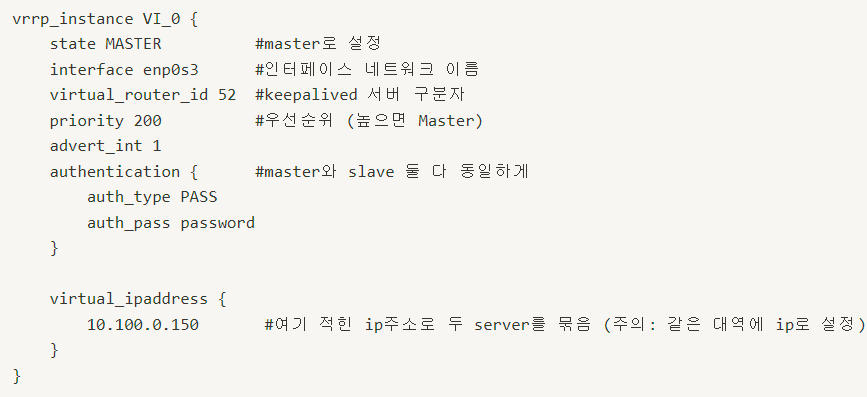
가. 네트워크 설정 변경

**<master db, slave db>**

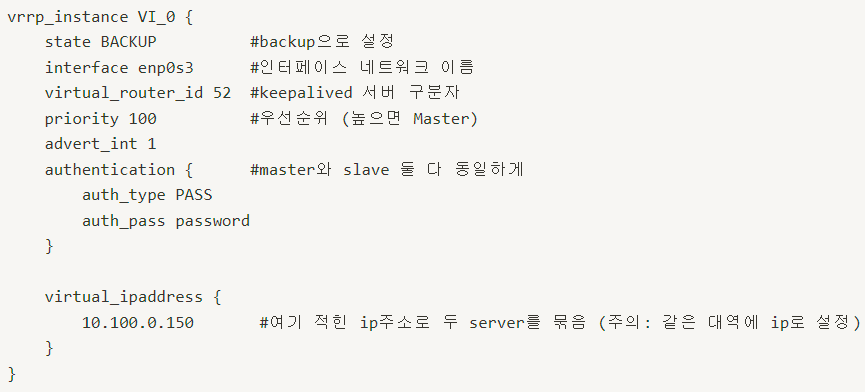
****

나. /etc/keepalived/keepalive.conf 파일 수정 (없으면 생성)

**<master db>**

****

**<slave db>**

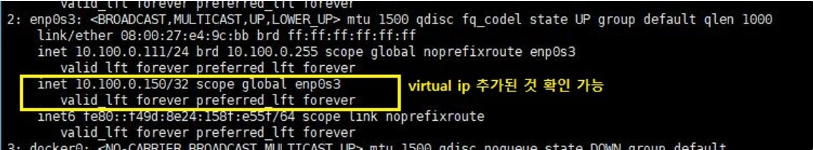


⇒ master db와 slave db와 구분하여 연결

다. 설정파일 적용

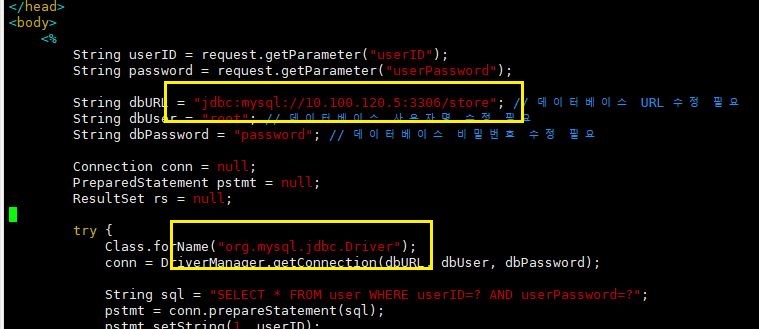
라. 적용 확인

: #ip a



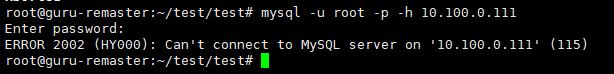
3) Trouble Shooting

가. tomcat에서 입력한 내용이 DB에 반영되지 않던 문제



|  |  |
| --- | --- |
| **원인** | **해결방법** |
| index.jsp에 부정확한 DB 정보가 입력되어 있어서 발생한 문제 | 1. DB 정보와 DB driver 부분에서 mysql 대신 mariadb를 입력해서 정확한 드라이버 정보와 DB 주소를 입력  2. keepalived를 사용하면서 설정한 virtual IP 값을 DB 서버와 같은 대역인 10.100.0.150으로 수정해서 DB와 연동 |

나. 외부에서 DB로 접속이 되지 않던 문제



|  |  |
| --- | --- |
| **원인** | **해결방법** |
| 외부에서 mariadb 서버로  접속이 불가능하던 문제 | /etc/mysql/mariadb.conf.d/50-server.cnf 파일에서 bind-address를 127.0.0.1에서 0.0.0.0으로 수정해서 해결함 |

**13. Ingress**

|  |  |  |
| --- | --- | --- |
| **Yaml 파일** | **구분** | **내용** |
|  | apiVersion | Kubernetes API 버전 |
| kind | Ingress 생성 |
| metadata | name: web인 서버와 연결 |
| ingressClassName | ingress 클래스 지정 |
| paths: pathType | profix URL 경로 지정 |
| service | name: svc-web인 서비스 참조 |
| port | 포트 번호 지정하여 해당 포트로 서비스 접근 |

**<Troble Shooting>**

1) Ingress가 연동이 되지 않던 문제

|  |  |
| --- | --- |
| **원인** | **해결방법** |
| Ingress 설정을 완료 했음에도 Ingress IP path 설정된 web으로 연동이 되지 않는 문제 | 작업 환경을 defualt가 아닌 다른 namespace에서 default namespace로 변경해서 다시 실행해서 해결 |

**14.Tomcat Server**

|  |
| --- |
|  |
| context.xml |
|  |
| server.xml |

**15. WhaTap**

1) 와탭 회원가입

무료로 시작하기 → 정보 입력 → 계정 생성

2) 프로젝트 생성

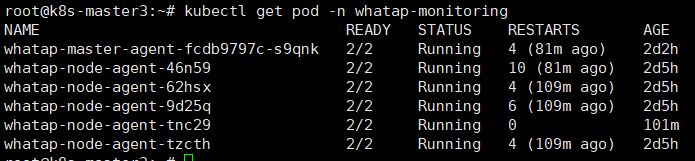


3) 쿠버네티스 모니터링 에이전트 설치

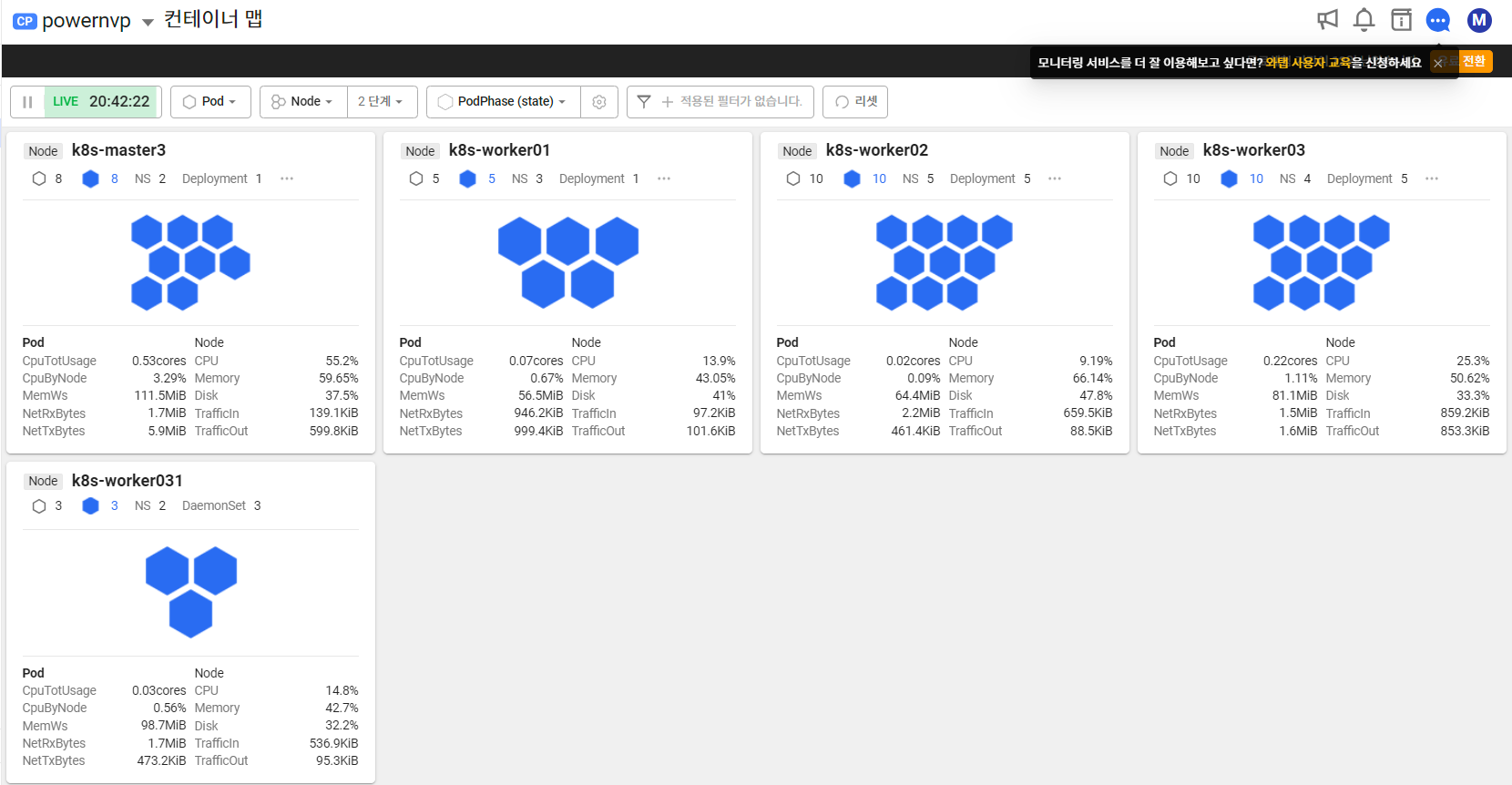
|  |  |
| --- | --- |
|  | - 설치 방식: yaml 파일 선택  → yaml 파일 다운로드 |

4) master server에서 yaml 파일 실행

5) 설치 확인



6) 모니터링



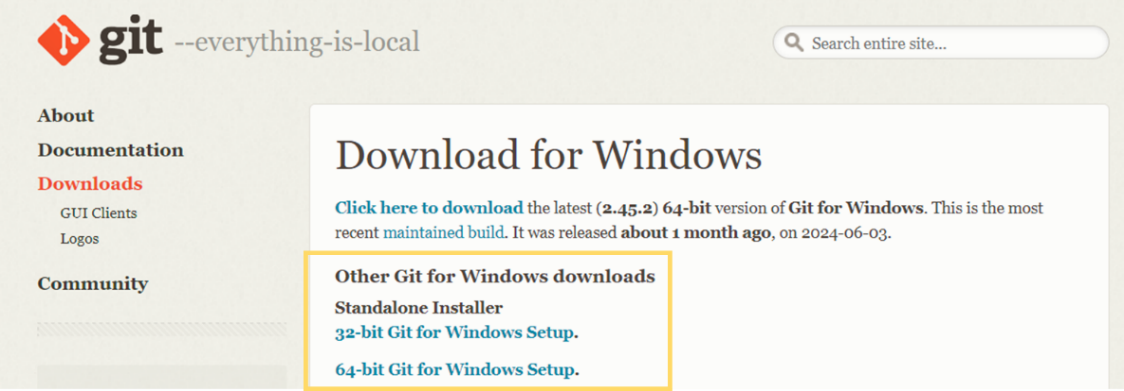
→ 와탭 홈페이지에서 확인 가능

**16. ArgoCD – GitHub 연동**

1) Git & Github

가. git 설치

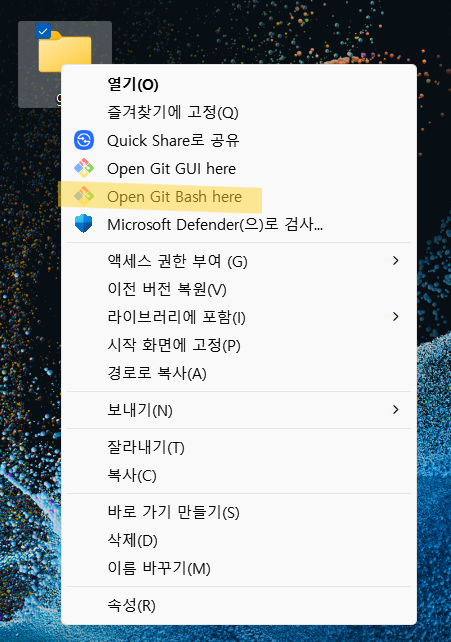
- [Git - Downloads (git-scm.com)](https://git-scm.com/downloads) → bit에 맞게 윈도우 다운로드



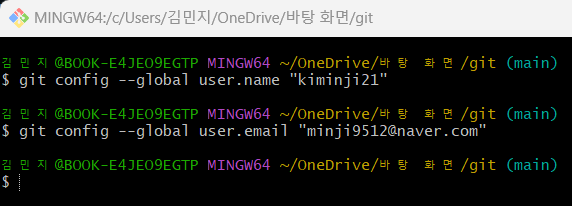
→ Setup 시 기본 선택으로 진행

나. git 설정

* Git 저장소로 사용할 폴더 생성 및 지정 후 우클릭 → Open Git Bash here 클릭



* 본인 Github 연결



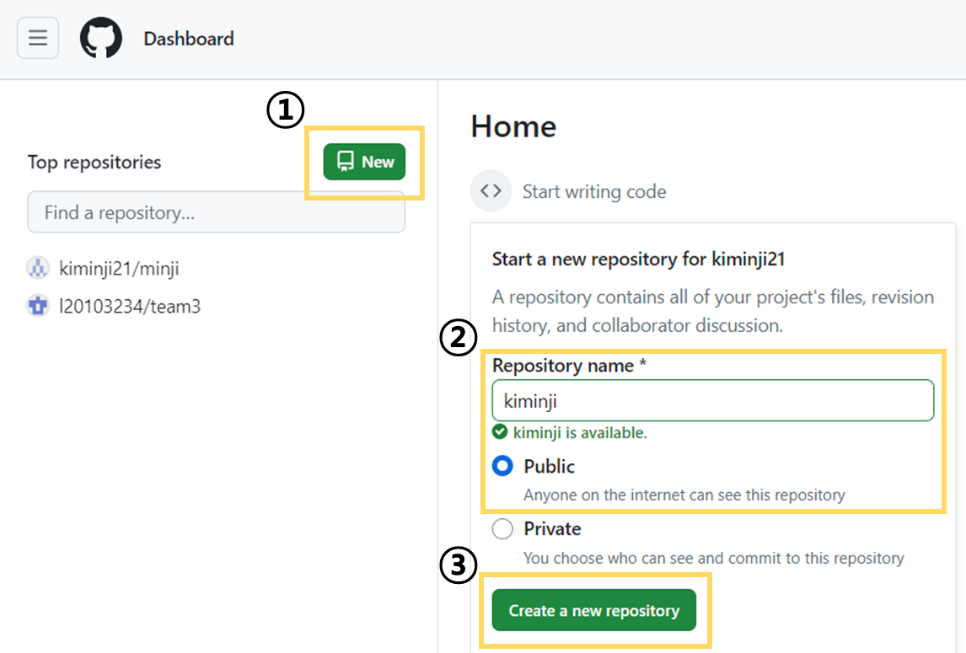
* 로컬 저장소와 Git Repository 연결

① Git Repository 생성

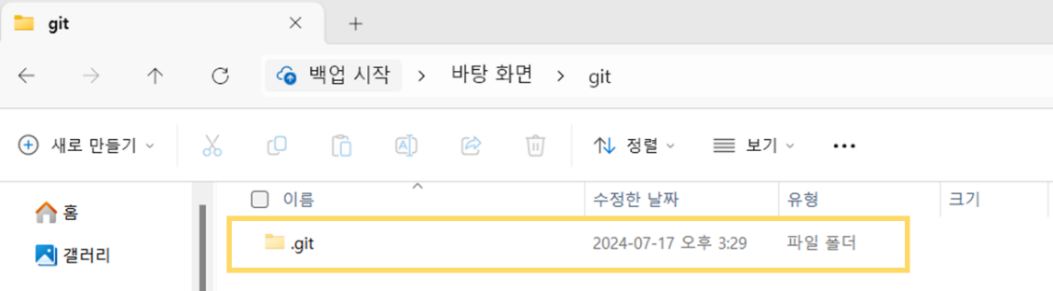
- Github 접속

- 좌측 상단 ‘New’ 클릭

- Repository name 입력 후 생성



→ 아래 사진처럼 숨김파일로 .git 폴더 생성된 것을 확인 가능



② 로컬 저장소에 Github 추가

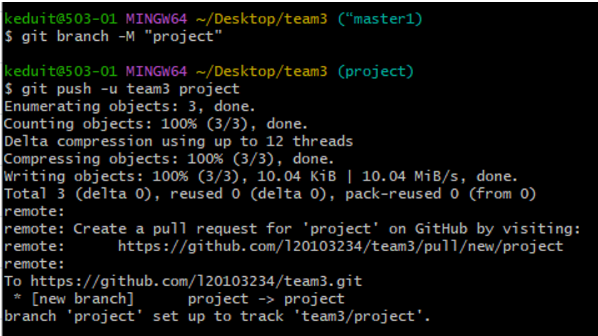
- $git remote add <remote-name> <git address>



③ 로컬 저장소에 Git Repository 연결

- 브랜치 생성: $ git branch -M “branch 이름”

- github 브랜치와 본인 브랜치 연결: $ git push -u <Repository name> <branch 이름>

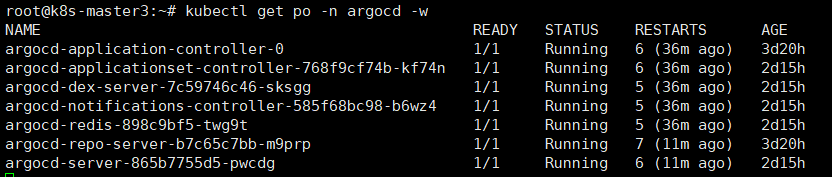


2) ArgoCD 설치

|  |
| --- |
| - $kubectl create namespace argocd  - $kubectl apply -n argocd -f <https://raw.githubusercontent.com/argoproj/argo-cd/v2.12.0-rc3/manifests/install.yaml> |

3) ArgoCD 설치 확인

: #kubectl get po -n argocd -w → STATUS: Running 확인



4) ArgoCD 접속

가. argocd password 찾기

: kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath={.data.password}” | base64 -d&&echo

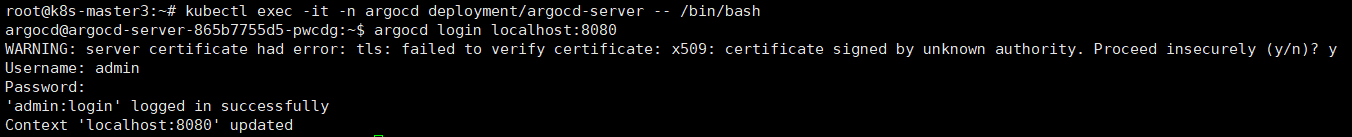


나. **argocd-server pod 접근**

: kubectl exec -it -n argocd deployment/argocd-server -- /bin/bash

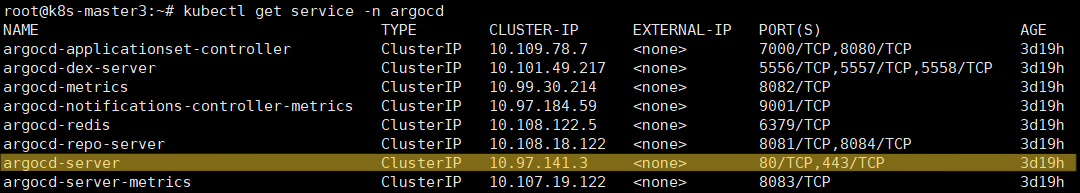
: argocd login [localhost](http://localhost):8080

\* 아래 참고 ( Username : admin , password : 위에서 찾은 비밀번호 )



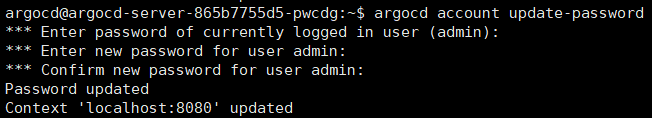
다. **argocd password 변경**

: argocd 파드에 진입 하여 $argocd account update-password



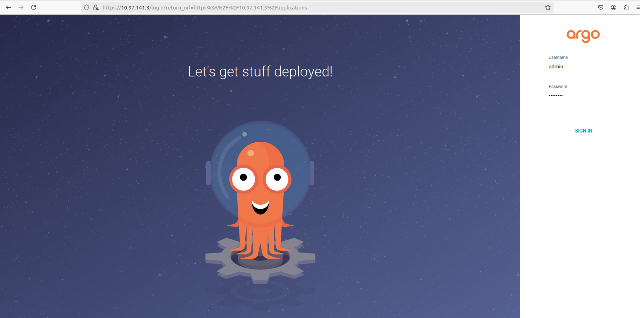
**라. argocd-server IP 확인**

: kubectl get service -n argocd



마. argocd 접속

: 우분투 Firefox에 argocd-server IP 입력 후 접속 → 로그인 확인



5) Github-ArgoCD 연동

가. 레포지토리 연결

|  |  |
| --- | --- |
|  |  |
|  | |

\* 설정 방법

- method VIA SSH —> VIA HTTPS로 변경

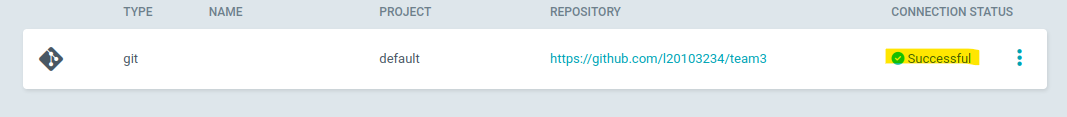
- TYPE 자동으로 git 설정

- Project 클릭하면 default 선택할 수 있음

- Repository URL 은 본인 Github 주소 입력

- 좌측 상단 CONNECT

나. github-argocd 연동 확인



→ successful 확인 ⇒ 연동 완료

\* unsuccessful: 입력값 확인 필요

6) Github-ArgoCD 배포

가. 배포

|  |  |
| --- | --- |
|  |  |
|  |  |

\* 설정 방법

- Application Name : 본인이 원하는 이름

- Project : ArgoCD에서 애플리케이션을 구분하고 관리하기 위한 그룹 **( Default 선택 )**

- SYNC POLICY : Manual 과 Automatic 중에 선택, Git의 환경과 비교하는 것을 수동으로 할지 자동으로 할지 선택 **( Automatic 선택 )**

- Prune Resources : 변경 사항에 따라 Resource를 업데이트할 때, 기존의 Resource를 삭제하고 새로운 Resource를 생성 **( 체크 )**

- Self Heal : ArgoCD가 지속적으로 Git Repository의 설정 값과 Sync를 맞추게 설정할 건지 선택 **( 체크 )**

- Repository URL : 사용할 Repository URL , 이전에 연동했던 Repository가 등록될 것이다. **( 본인 Github URL주소 )**

- Revision : Git의 어떤 Revision (HEAD , Master branch 등)을 바라볼지 결정

**( 기본 값 HEAD )**

- Path : Git에서 어떤 저장소의 어떤 디렉토리를 바라볼지 결정

**( Github에 따로 Directory를 만들어 놓지 않았다면 ./ )**

- Cluster URL : Kubernetes의 어느 클러스터에 배포할지 결정

\*\*( 기본 값. 클릭하면 선택할 수 있음 )\*\*

- Namespace : Kubernetes 클러스터의 어느 Namespace에 배포할지 결정

**( 사용한 Namespace가 있는게 아니라면 default로 설정 )**

- 좌측 상단 CREATE

**⇒ Health, Synced 확인 → 배포(자동화) 성공**

**17. Jmeter**

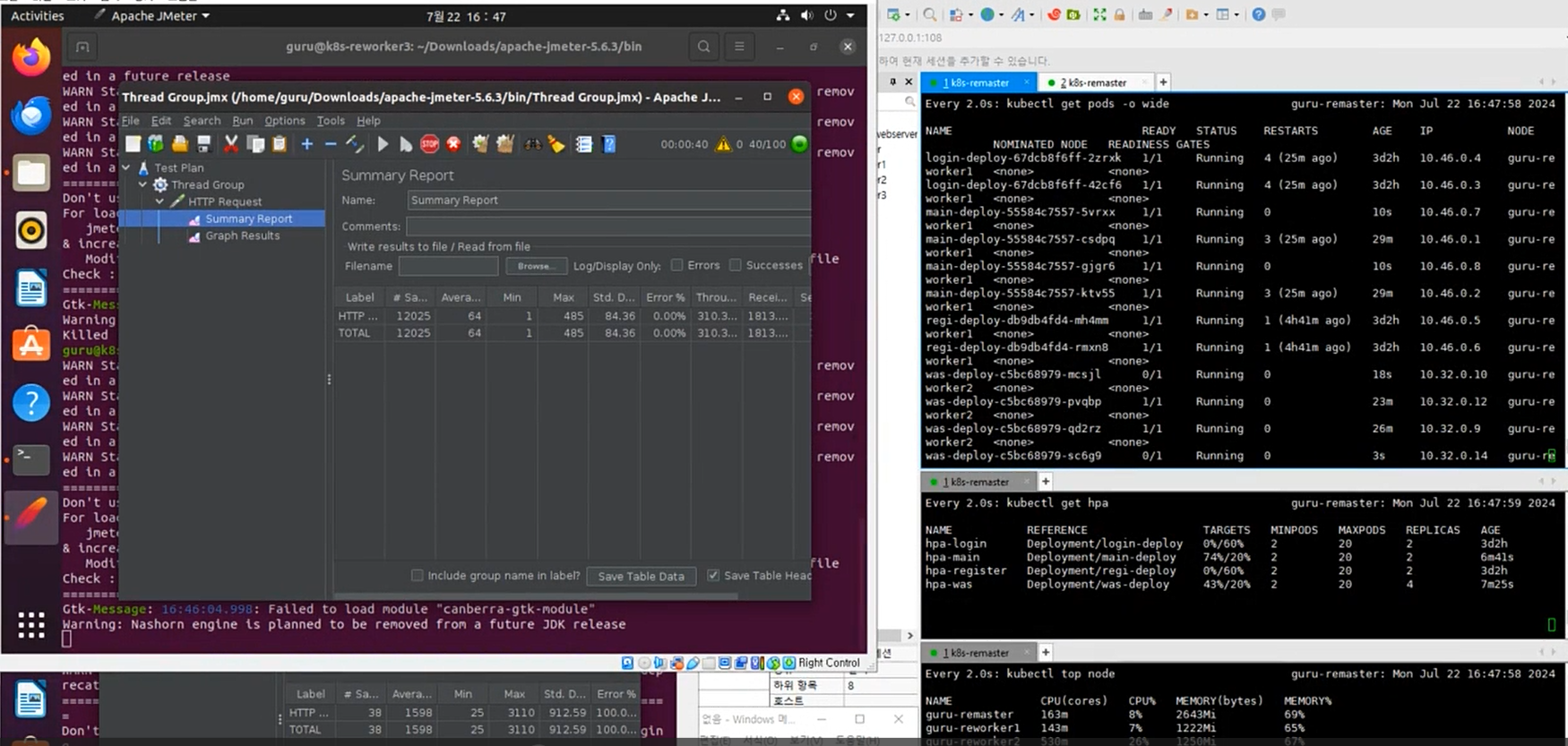
1) 자바 설치

: apt-get install -y openjdk-11\*

2) jmeter 설치(Ubuntu)

: jmeter 홈페이지 설치 후 실행

3) 부하테스트



**18. Rolling Update & Roll Back**

1) 이미지 변경

|  |  |
| --- | --- |
| **기존** | **변경 후** |
|  |  |

2) argocd SYNCHRONIZE



3) 업데이트 확인

|  |  |
| --- | --- |
| **업데이트 전** | **업데이트 후** |
|  |  |

**19. multi master node 환경을 통한 가용성 확보**

1) Master node의 keepalived를 설치하기 전, loadbalancer를 구성 및 HAproxy 설정

|  |
| --- |
|  |

2) 2개의 loadbalancer에 keepalived 설치

3) master node(main) keepalived 설정

|  |  |
| --- | --- |
| **파일** | **내용** |
|  | **/etc/keepalived/keepalived.conf 파일 수정**  - state MASTER와 BACKUP 차별화  - Interface는 현재 사용 중인 네트워크의 enp0s3 사용(환경에 따라 다름)  - Virtual id는 51로 master node 모두 동일  - priority: master가 높음(우선순위 지정)  - virtual ip: HAproxy와 master node 동일하게 지정 |

4) virtual IP 주소 확인

|  |
| --- |
|  |

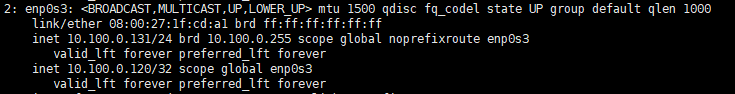
5) master node(sub) keepalived 설치 및 설정

|  |  |
| --- | --- |
| **파일** | **내용** |
|  | **/etc/keepalived/keepalived.conf 파일 수정**  - state MASTER와 BACKUP 차별화  - Interface는 현재 사용 중인 네트워크의 enp0s3 사용(환경에 따라 다름)  - Virtual id는 51로 master node 모두 동일  - priority: master가 높음(우선순위 지정)  - virtual ip: HAproxy와 master node 동일하게 지정 |

6) 설정 확인



<keepalived 적용 전>



<keepalived 적용 후에 virtual IP를 확인 가능>

**<trouble shooting>**

1. Endpoint error

|  |  |
| --- | --- |
| **문제** | **해결방법** |
| Multi master node를 join 할 때 end point를 찾을 수 없다는 error가 발생 | 설정 파일에서 end point 값을  main master node IP로 수정해주면 해결 |

1. Etcd error

|  |  |
| --- | --- |
| **문제** | **해결방법** |
| Main master node와 join 이후에도  main master node의 문제가 생기면  slave master node들이 etcd와 연결이 되지 않아 정상 작동 불가 | **/etc/kubernetes/manifests/kube-apiserver.yaml을 수정**  : advertise-address를 VIP(10.100.0.120)로 수정 및 --etcd-servers 내용 추가 |

3) couldn’t get API server error

|  |  |
| --- | --- |
| **문제** | **해결방법** |
| etcd 문제를 해결한 후에도  여전히 main master node에  문제가 생기면 slave master node가  정상 작동 불가 | **명령어로 API server 경로 재지정**  [명령어]  #mkdir -p $HOME/.kube  #sudo cp -i /etc/kubernetes/kubelet.conf #$HOME/.kube/config  #sudo chown $(id -u):$(id -g) #$HOME/.kube/config |