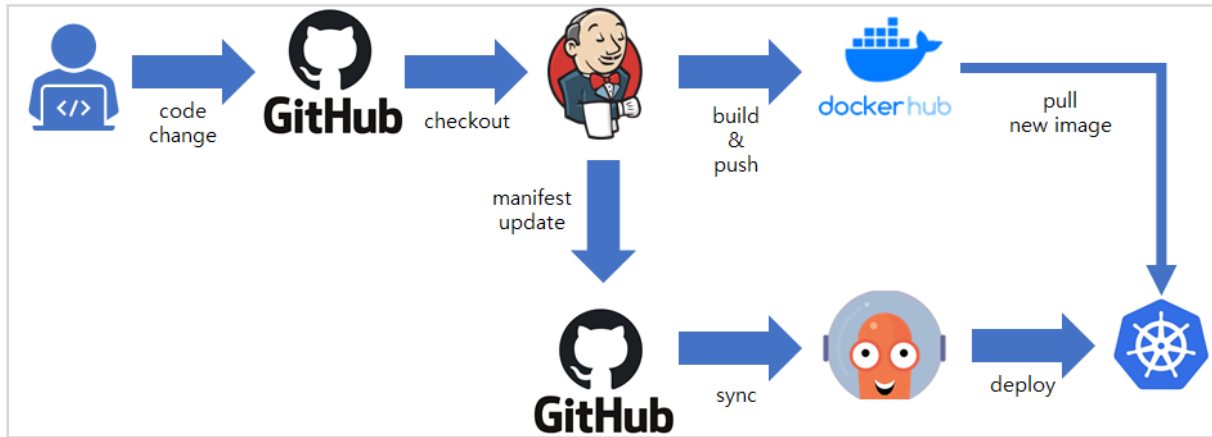


K8s Jenkins&Argo CI/CD Pipeline 구축

Kubernetes 위에 Jenkins 설치하기 [참고](#)

argoCD - [ArgoCD: Kubernetes에 GitOps 적용하기](#) [참고](#)

kustomizer - <https://cwal.tistory.com/22> [참고](#)



K8s에 Jenkins 배포

jenkins 서비스가 배포될 노드에 /jenkins-data 디렉토리 생성

```
$ mkdir /jenkins-data
$ chmod 777 /jenkins-data
```

ci namespace 생성

```
$ kubectl create namespace ci
```

yaml 생성 후 배포

jenkins-master.yaml

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: jenkins
spec:
  serviceName: jenkins
  replicas: 1
  selector:
    matchLabels:
      app: jenkins
  template:
    metadata:
      labels:
        app: jenkins
    spec:
      serviceAccountName: jenkins
      containers:
      - name: jenkins
        image: jenkins/jenkins:lts
        ports:
          - name: http-port
            containerPort: 8080
          - name: jnlp-port
            containerPort: 50000
        volumeMounts:
          - name: jenkins-vol
            mountPath: /var/jenkins_home
      volumes:
      - name: jenkins-vol
        hostPath:
          path: /jenkins-data
          type: Directory
---
apiVersion: v1
kind: Service
metadata:
  name: jenkins
spec:
  type: NodePort
```

```

    ports:
      - port: 8080
        targetPort: 8080
    selector:
      app: jenkins
---
apiVersion: v1
kind: Service
metadata:
  name: jenkins-jnlp
spec:
  type: ClusterIP
  ports:
    - port: 50000
      targetPort: 50000
  selector:
    app: jenkins

```

jenkins-rbac.yaml

```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: jenkins
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["create", "delete", "get", "list", "patch", "update", "watch"]
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create", "delete", "get", "list", "patch", "update", "watch"]
- apiGroups: [""]
  resources: ["pods/log"]

```

```

    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["get", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: jenkins
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: jenkins
subjects:
- kind: ServiceAccount
  name: jenkins

```

배포

```

$ kubectl apply -n ci -f jenkins-rbac.yaml
$ kubectl apply -n ci -f jenkins-master.yaml

```

Jenkins Dashboard

NodePort 확인

```
$ kubectl get svc -n ci
```

```
>>>
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
------	------	------------	-------------	---------	-----

jenkins	NodePort	10.105.178.161	<none>	8080:30146/TCP	67m
jenkins-jnlp	ClusterIP	10.111.24.33	<none>	50000/TCP	67m

웹 브라우저에서 {Master IP}:30146 으로 접속

Dashboard Token 확인

```
$ kubectl exec -it -n ci jenkins-0 -- cat /var/jenkins_home/secrets/initialAdminPassword
>>> 43b5f6b6be7d4bb3b325e8de6e79d17e
```

Install suggested plugin 클릭

자동으로 필요한 패키지가 설치됨 (5~10 분정도 소요)

Getting Started

Getting Started

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	⚙ Credentials Binding
⚙ Timestampers	⚙ Workspace Cleanup	⚙ Ant	⚙ Gradle
⚙ Pipeline	⚙ GitHub Branch Source	⚙ Pipeline: GitHub Groovy Libraries	⚙ Pipeline: Stage View
⚙ Git	⚙ SSH Build Agents	⚙ Matrix Authorization Strategy	⚙ PAM Authentication
○ LDAP	⚙ Email Extension	○ Mailer	

설치 이후 나오는 화면에서 admin 계정 설정

테스트 환경에서는 아래와 같이 생성함

- ID: admin
- PW: miruware!

Jenkins 관리 -> 플러그인관리 -> 설치가능 에서 kubernetes와 Docker Pipeline 플러그인 검색하여 설치

Kubernetes	✓ 성공
Loading plugin extensions	✓ Success
Docker Commons	✓ 성공
Docker Pipeline	✓ 성공
Loading plugin extensions	✓ Success

Jenkins 관리 -> 노드 설정 -> Configure Clouds 에서 Kubernetes 선택

Configure Clouds

Kubernetes

Name:

Kubernetes URL:

Use Jenkins Proxy: ☐

Kubernetes server certificate key:

Disable https certificate check: ☐

Kubernetes Namespace:

Credentials:

WebSocket: ☐

Direct Connection: ☐

Jenkins URL:

Jenkins tunnel:

Connection Timeout:

Read Timeout:

Concurrency Limit:

Pod Labels

Pod Label

Key:

Value:

- Name: 해당 클러스터를 구분할 수 있는 이름
 - kubernetes
- Kubernetes URL: Jenkins가 k8s 내부에서 실행중이므로, API서버의 In-cluster URL을 입력
 - <https://kubernetes.default.svc>
- Kubernetes Namespace: Remote Agent 관련 리소스가 생성될 Namespace를 의미한다.

RBAC 설정시 'ci' Namespace에만 유효하도록 정의하였으므로 ci를 입력

- ci
- Jenkins URL: In-cluster의 Jenkins URL로 HTTP 관련 Service 이름과 Port를 명시
 - `http://jenkins:8080`
- Jenkins tunnel: Remote Agent가 접근할 주소, jnlp 관련 Service 이름과 Port를 입력
 - `jenkins-jnlp:50000`
- Pod Labels 추가
 - Key: jenkins
 - Value: slave

나머지 항목은 입력하지 않고 비워두거나 기본값으로 설정.

Test Connection 버튼을 눌러서 k8s 연결에 문제가 없는지 확인 후 Save 버튼을 눌러서 위 설정을 적용

Connected to Kubernetes v1.21.14

Test Connection

Jenkins와 Github 연동

Jenkins Master 서버의 Container 안으로 들어가 SSH Key 생성

```
$ kubectl exec -ti -n ci jenkins-0 -- bash
```

```
(jenkins-0)$ ssh-keygen
```

<Git Hub>

SSH key(id_rsa.pub) 확인

```
(jenkins-0)$ cat /var/jenkins_home/.ssh/id_rsa.pub
```

```
>>>
```

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGDHGL2W6nfl043IBJXCdoIhpeBtb/  
zPppBjZYFnlpXAVgsPKdK146KuYpAt04RbXFSC+uE9X3J7uVP6aG7rlQNErjBgX0qLYgWdxyXh3gdSy  
f0L4IVS122Xrc3HINDjX+xee8J43i9AQe9+kjD3imQ8CqcdwW03owmbEUs/wxt9xgbyQPwNGG/Yv/  
SPZwgVjvTeie9YUD0BAFJthkv03xEdKeGAU2bdUg6XpKVoY7tGgx8pHS2fcKPf1kPK25q9gBwQvPkds
```

```
I7RQ+/0/nJ+xEQ10Y0U4r6pl2gEvU8Ycob3QxKcw30T//  
8HbYdPsk9gxbVwJDfNN243WdVfHSduVeTeKP+Pi++eX5/  
KC2FxNAKkar9IOkr5lHNjWuaNL0PaY7VzbPehWA1NTNjZ105Vc+fP47Yj08SKfQumAmXLB4D0J0feXZ  
cTXjA9oXNu0+7Af/  
0ZKypqeGpIjpFNGRCSG3F6cFfyBBKbBrNd7yV36lLmNKtMtDF3t0vDG3JxyT8IZ00=  
jenkins@jenkins-0
```

Git Hub 홈페이지 Setting -> SSH and GPG keys 탭에서 SSH key 등록

SSH keys / Add new

Title

jenkins-ssh-public-key

Key

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQgQDHL2W6nflo43IBJXCdohpeBtb/zPppBjZyFnlpxAVgsPKdK146KuYpATo4
RbXFSC+uE9X3J7uVP6aG7rIQNErjBgXOqLYgWdxyXh3gdSyfOL4IVSI22Xrc3HINDjX+xee8J43i9AQe9+kjD3imQ8Cq
cdwW03owmbEUs/wxt9xgbyQPwNGG/Yv/SPZwgVjvTeie9YUDOBafJthkv03xEdKeGAU2bdUg6XpKVoY7tGgx8pHS
2fcKPflkPK25q9gBwQvPkdSI7RQ+/0/nJ+xEQ1OY0U4r6pl2gEvU8Ycob3QxKcW3OT//8HbYdPsk9gxbVwJDfNN243W
dVfHSduVeTeKP+Pi++eX5/KC2FxNAKkar9IOkr5IHnjWuaNL0PaY7VzbPehWA1NTNjZ105Vc+fP47YjO8SKfQumAmX
LB4DOJOfexXZcTXJA9oXNuO+7Af/0ZKypqeGpljpfNGRCSG3F6cFfyBBKbBrNd7yV36lLmNKtMtDF3t0vDG3JxyT8lZO
0= jenkins@jenkins-0

Add SSH key

<Jenkins>

SSH Key(id_rsa) 확인

```
(jenkins-0)$ cat /var/jenkins_home/.ssh/id_rsa

>>>

-----BEGIN OPENSSH PRIVATE KEY-----

#%#%#%#%#%#%#@#

#%!@#^#%#@#%#@#%

...


!@#$$$%#@%#%#@#


!@#$==

-----END OPENSSH PRIVATE KEY-----
```


Jenkins Dashboard 관리 -> Manage Credentials 로 이동 후 Credentials 등록

Stores scoped to Jenkins

P	Store ↑	Domains
	Jenkins	(global)

 Add credentials

Update credentials

Scope ?

Global (Jenkins, nodes, items, all child items, etc) ▼

ID ?

jenkins-ssh-private

Description ?

Username


Yangsuseong

☐ Treat username as secret ?

Private Key

☒ Enter directly

Key

 Enter New Secret Below

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3B1bnNzaC1rZXktdjEAAAABG5vbmUAAAABbm9uZQAAAAAAAAABAAAblwAAAAAdzc2gtcn
NhAAAAAwEAAQAAAEAxix9lup35aONyASVwnaCIaXqbW/8z6aOY2WBZ5acOFYLDynSteOi
```

- Credential ID 와 Private Key 값만 입력하고 나머지는 공백
 - ID : Git hub ID
 - Key: 위에서 확인한 id_rsa 값 붙여넣기


Jenkins 관리 -> Configure Global Security 로 이동

Jenkins 2.346.2 버전 기준 최 하단의
Git Host Key Verification Configuration 을
Accept first connection
으로 변경

<Docker Hub>

Jenkins Dashboard 관리 -> Manage Credentials 로 이동 후 Credentials 등록

Stores scoped to Jenkins

P	Store ↑	Domains
	Jenkins	(global) Add credentials

Update credentials

Scope ?


Global (Jenkins, nodes, items, all child items, etc) ▼

Username ?

tntjd5596

☐ Treat username as secret ?

Password ?

 Concealed [Change Password](#)

ID ?

Yangsuseong-docker

Description ?

[Save](#)

- Docker hub Username/Password 입력
- ID : Jenkins에서 변수로 사용할 ID

나머지는 빈칸으로 둔다.

Git Repository 추가

GitHub - [learnk8s/docker-hello-world: Node.js "Hello world"](#)

위 링크에 있는 코드를 Fork 하여 테스트를 진행한다.

Git clone 후 Jenkinsfile 을 생성하여 Commit&Push 한다.

```
$ git clone https://github.com/Yangsuseong/docker-hello-world.git
```

```
$ vim Jenkinsfile
```

Jenkinsfile

```
podTemplate(label: 'docker-build',  
  containers: [  
    containerTemplate(  
      name: 'git',  
      image: 'alpine/git',  
      command: 'cat',  
      ttyEnabled: true  
    ),  
    containerTemplate(  
      name: 'docker',  
      image: 'docker',  
      command: 'cat',  
      ttyEnabled: true  
    ),  
  ],  
)
```

```

volumes: [
  hostPathVolume(mountPath: '/var/run/docker.sock', hostPath: '/var/run/
docker.sock'),
]
) {
  node('docker-build') {
    def dockerHubCred = <your_dockerhub_cred>
    def appImage

    stage('Checkout'){
      container('git'){
        checkout scm
      }
    }

    stage('Build'){
      container('docker'){
        script {
          appImage = docker.build("<your-dockerhub-id>/node-hello-
world")
        }
      }
    }

    stage('Test'){
      container('docker'){
        script {
          appImage.inside {
            sh 'npm install'
            sh 'npm test'
          }
        }
      }
    }

    stage('Push'){
      container('docker'){
        script {
          docker.withRegistry('https://registry.hub.docker.com',

```

```
'<Your Dockerhub Cred>'){
    appImage.push("${env.BUILD_NUMBER}")
    appImage.push("latest")
}
}
}
}
}
}
}
```

commit & push 한다

```
$ git add Jenkinsfile
$ git commit -a
$ git push
```

Jenkins Pipeline Job 추가


Jenkins Dashboard -> 새로운 Item 에서 Job 추가

Job 이름 및 종류 선택


Enter an item name

node-hello-world-docker-build

» Required field


Freestyle project

이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.


Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Job 구성

☐ Do not allow concurrent builds

☐ Do not allow the pipeline to resume if the controller restarts

☒ GitHub project

Project url ?

☐ Pipeline speed/durability override ?

☐ Preserve stashes from completed builds ?

☐ Throttle builds ?

☐ 오래된 빌드 삭제 ?

☐ 이 빌드는 매개변수가 있습니다 ?

Build Triggers

☐ Build after other projects are built ?

☐ Build periodically ?

☒ GitHub hook trigger for GITScm polling ?

☐ Poll SCM ?

☐ 빌드 안함 ?

☐ Quiet period ?

☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?

- GitHub project: 빌드할 소스코드와 Jenkinsfile이 위치한 GitHub 프로젝트 URL(.git 생략)
- Github hook trigger for GITScm polling 체크
 - git push 가 이루어질때마다 자동으로 빌드 진행하도록 하는 trigger

(notice)

Poll SCM을 체크하고 'H/5 * * * *' 을 입력 시 5분마다 빌드가 진행되도록 설정할 수 있음.

Pipeline 설정

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/Yangsuseong/docker-hello-world.git

Credentials ?

Git

+ Add

고급...

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/master

Add Branch

Repository browser ?

(자동) ▼

Additional Behaviours

Add ▼

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

[Pipeline Syntax](#)

저장 Apply

- 체크아웃할 Repository에 이미 Pipeline Script이 존재하므로 'Pipeline script from SCM' 항목을 선택
- Repository URL은 해당 Repository SSH URL(.git 포함)을 입력
- Credential은 미리 생성한 GitHub Credential을 사용
- 빌드할 대상 브랜치는 master

Github Repository 의 web hook 설정

Github -> Repository -> Setting -> Webhooks

- [Add Webhook] 클릭하여 web hook 생성

Webhooks / Manage webhook

Settings

Recent Deliveries

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

Payload URL *

`http://miruware-a.iptime.org:30146/github-webhook/`

Content type

`application/json`

Secret

Which events would you like to trigger this webhook?

- ☒ Just the push event.
- ☐ Send me **everything**.
- ☐ Let me select individual events.

☒ **Active**

We will deliver event details when this hook is triggered.

Update webhook

Delete webhook

- Payload URL
 - `http:{Jenkins 를 외부에서 접속할 수 있는 Domain}github-webhook`
- Content type
 - `application/json`

Web hook 정상작동 확인

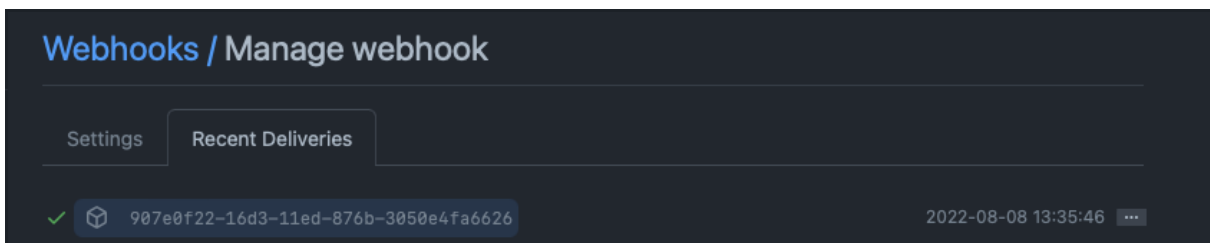
Jenkins와 연동되어 있는 Repository에 코드를 수정하여 Push

```

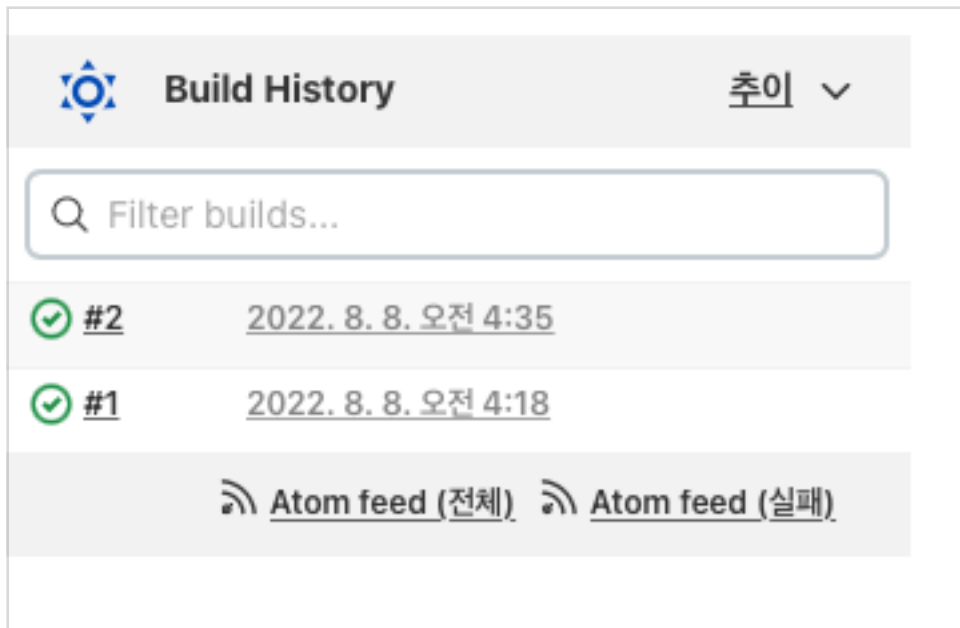
mercury ~/Documents/git/docker-hello-world master
> vim index.js
mercury ~/Documents/git/docker-hello-world master ±
> git add index.js
mercury ~/Documents/git/docker-hello-world master +
> git commit
[master c8d0653]      modified:   index.js
1 file changed, 1 insertion(+), 1 deletion(-)
mercury ~/Documents/git/docker-hello-world master
> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 299 bytes | 299.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/Yangsuseong/docker-hello-world.git
3496b08..c8d0653  master -> master
mercury ~/Documents/git/docker-hello-world master

```

Github 홈페이지 → Repository → Setting → Webhooks → 생성한 web hook 에서 Recent Deliveries 클릭하여 확인

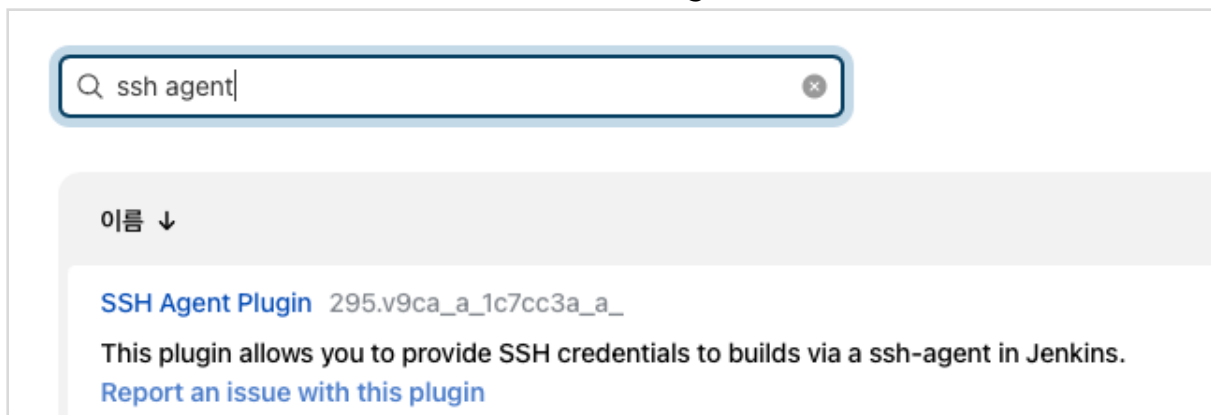


Jenkins 작업에서 Build가 이루어졌는지 확인



Jenkins Dashboard 에서 ssh agent 설치

Jenkins 관리 -> 플러그인 관리 -> 설치가능 -> ssh agent



ssh Agent 설치 진행

ArgoCD 배포

Argo CD 설치

Argo CD 배포용 namespace 생성

```
$ kubectl create namespace argocd
```

Argo CD Manifest 파일 다운로드

```
$ curl https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml -o argo-cd.yaml
```

Manifest 파일 수정

현재 테스트 환경의 k8s는 Desktop의 VM 위에 구축되었기 때문에, Load Balancer 대신 NodePort를 사용한다.

argocd-server 서비스 부분을 찾아 type: NodePort 를 추가한다.

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/component: server
    app.kubernetes.io/name: argocd-server
    app.kubernetes.io/part-of: argocd
  name: argocd-server
spec:
  type: NodePort
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 8080
    - name: https
      port: 443
      protocol: TCP
      targetPort: 8080
  selector:
    app.kubernetes.io/name: argocd-server
```

Argo CD CLI Tool 다운로드 후 환경변수 추가

```
$ VERSION=$(curl --silent "https://api.github.com/repos/argoproj/argo-cd/releases/latest" | grep 'tag_name' | sed -E 's/.*"([^\"]+)"\.*/\1/')

$ curl -sSL -o /usr/local/bin/argocd https://github.com/argoproj/argo-cd/releases/download/$VERSION/argocd-linux-amd64

$ chmod +x /usr/local/bin/argocd
```

Argo CD 배포

```
$ kubectl apply -n argocd -f argo-cd.yaml
```

서비스 포트 확인

```
$ kubectl get svc -n argocd argocd-server

NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)
argocd-server                       NodePort            10.101.149.21   <none>           80:30817/TCP
TCP,443:30499/TCP                    5s
```

Argo CD Dashboard

admin 계정 Default Password 확인

계정 : admin

```
$ kubectl -n argocd get secret argocd-initial-admin-secret -o
jsonpath="{.data.password}" | base64 -d

>>> 5ENdc9r24v83djXH
```

admin 계정의 비밀번호를 변경

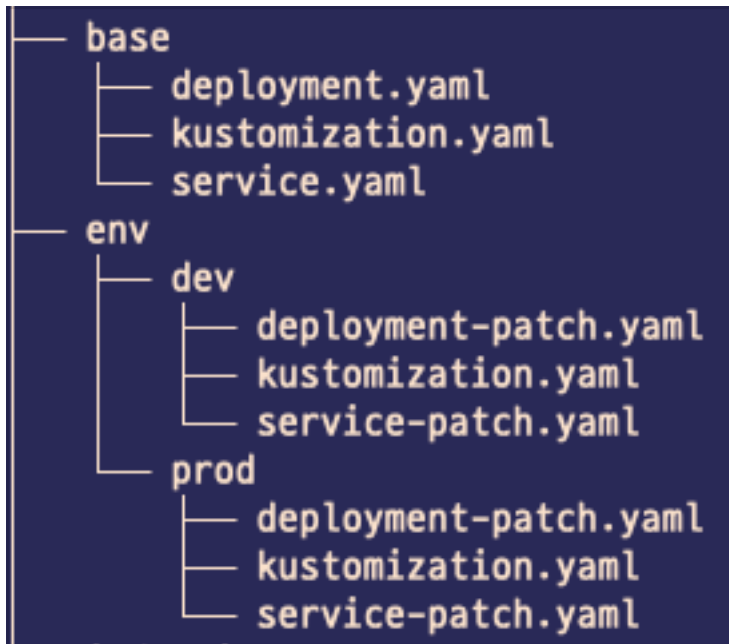
```
$ argocd login <ARGOCD_SERVER>
#ex) argocd login localhost:30817

$ argocd account update-password
>>>
*** Enter password of currently logged in user (admin):
*** Enter new password for user admin:
*** Confirm new password for user admin:
Password updated
Context 'localhost:30953' updated
```

Kustomizer 설치 및 사용 준비

- 테스트용 Hello world 디렉토리와 kustomizer 디렉토리는 다른 repository 에서 관리되어야 한다.
 - 만약 같은 디렉토리에 있으면?
 - Tag 변경을 위해 지속적으로 Git push 가 이루어지므로 이미지 빌드가 무한 루프가 돈다.
- 이번 테스트에서는 같은 Git Repository에 관리를 진행하여 무한 루프가 도는 것을 확인할 수 있다.

kustomizer 기본 디렉토리 구조



Manifest 작성

`.basedeployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deployment
  labels:
    app: hello
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
        - name: hello
          image: <Docker Hub 계정>/node-hello-world:latest
          ports:
            - containerPort: 8080
```

.baseservice.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: hello
spec:
  type: NodePort
  selector:
    app: hello
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

.basekustomization.yaml

```
resources:
- deployment.yaml
- service.yaml
```

.envdev/deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deployment
  labels:
    env: dev
spec:
  replicas: 4
```

.envdev/service.yaml


```
apiVersion: v1
kind: Service
```

.envdev/kustomization.yaml

```
patchesStrategicMerge:
- deployment-patch.yaml
- service-patch.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- ../../base
images:
- name: <Docker Hub 계정>/node-hello-world
  newTag: "278"
```

prod 디렉토리는 이번 테스트에서는 없어도 됨.

Jenkinsfile 수정

.envdev/kustomization.yaml 파일의 newTag 값이 자동으로 변경될 수 있도록
Git Repository 내의 Jenkinsfile을 수정한다.

Jenkinsfile

```
podTemplate(label: 'docker-build',
  containers: [
    containerTemplate(
      name: 'docker',
      image: 'docker',
      command: 'cat',
      ttyEnabled: true
    ),
    containerTemplate(
      name: 'argo',
      image: 'argoproj/argo-cd-ci-builder:latest',
      command: 'cat',
```

```

        ttyEnabled: true
    ),
],
volumes: [
    hostPathVolume(mountPath: '/var/run/docker.sock', hostPath: '/var/run/
docker.sock'),
]
) {
    node('docker-build') {
        def dockerHubCred = "dockerhub_cred"
        def appImage

        stage('Checkout'){
            container('argo'){
                checkout scm
            }
        }

        stage('Build'){
            container('docker'){
                script {
                    appImage = docker.build("<Docker Hub 계정>/node-hello-
world")
                }
            }
        }

        stage('Test'){
            container('docker'){
                script {
                    appImage.inside {
                        sh 'npm install'
                        sh 'npm test'
                    }
                }
            }
        }

        stage('Push'){

```

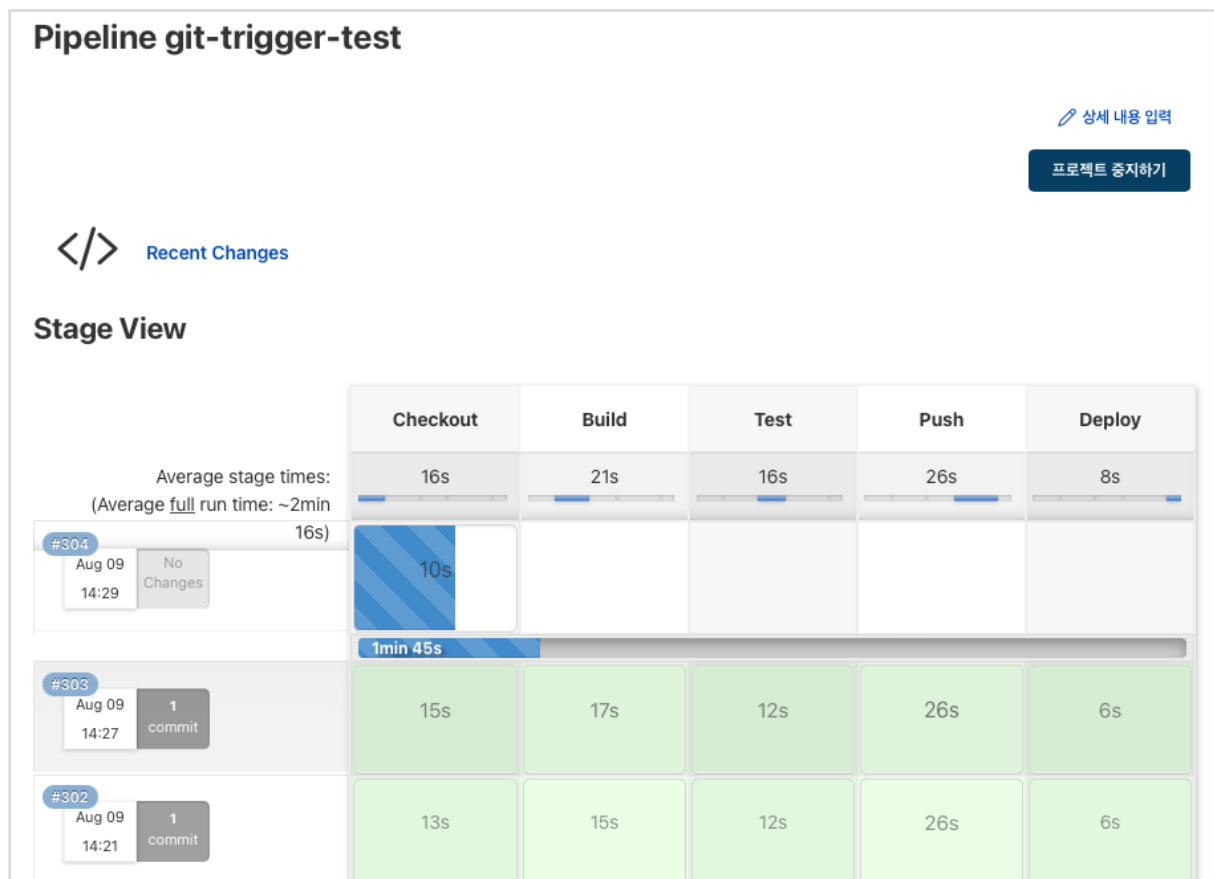
```

        container('docker'){
            script {
                docker.withRegistry('https://registry.hub.docker.com',
'<Jenkins Docker 인증서 이름>'){
                    appImage.push("${env.BUILD_NUMBER}")
                    appImage.push("latest")
                }
            }
        }
    }

stage('Deploy'){
    container('argo'){
        checkout([class: 'GitSCM',
            branches: [[name: '*/master' ]],
            extensions: scm.extensions,
            userRemoteConfigs: [[
                url: '<Git Hub 주소>',
                credentialsId: '<Jenkins Git Hub 인증서 이름>',
            ]]
        ])
        sshagent(credentials: ['<Git Hub ssh 인증서 이름>']){
            sh("""
                #!/usr/bin/env bash
                set +x
                export GIT_SSH_COMMAND="ssh -oStrictHostKeyChecking=no"
                git config --global user.email "<Git Hub ID>"
                git checkout master
                cd env/dev && kustomize edit set image tntjd5596/node-
hello-world-test:${BUILD_NUMBER}
                git commit -a -m "updated the image tag"
                git push
            """)
        }
    }
}
}
}
}
}


```

Jenkins 이미지 빌드 정상 작동 확인



빌드 완료 후 Git Hub Repository의 .envdev/kustomization.yaml 확인

🔑 master ▾ docker-hello-world / env / dev / kustomization.yaml

 Yangsuseong updated the image tag

👤 1 contributor

10 lines (10 sloc) | 220 Bytes

```
1 patchesStrategicMerge:
2   - deployment-patch.yaml
3   - service-patch.yaml
4 apiVersion: kustomize.config.k8s.io/v1beta1
5 kind: Kustomization
6 resources:
7   - ../../base
8   images:
9     - name: tntjd5596/node-hello-world-test
10       newTag: "304"
```

newTag 가 최신 빌드 버전(304)으로 변경된 것을 확인할 수 있다.

Argo Application 배포

Argo Dashboard 에서 [+ NEW APP] 버튼 클릭

[GENERAL]

CREATE

CANCEL

×

GENERAL

EDIT AS YAML

Application Name

hello-world-test

Project Name

default

SYNC POLICY

Automatic

PRUNE RESOURCES

SELF HEAL

SET DELETION FINALIZER

SYNC OPTIONS

SKIP SCHEMA VALIDATION

PRUNE LAST

RESPECT IGNORE DIFFERENCES

AUTO-CREATE NAMESPACE

APPLY OUT OF SYNC ONLY

PRUNE PROPAGATION POLICY: foreground

REPLACE

RETRY

- Application Name
 - <생성할 Application 이름>
 - Project Name
 - default
 - SYNC POLICY
 - 자동으로 지속적인 배포가 필요하므로 Automatic 선택
 - ☒ SELF HEAL
 - ☒ AUTO-CREATE NAMESPACE
- 체크

[SOURCE]

SOURCE

Repository URL

https://github.com/Yangsuseong/docker-hello-world.git

GIT ▼

Revision

HEAD

Branches ▼

?

Path

env/dev

- Repository URL
 - Source 가 있는 Git Hub Url
- Revision
 - HEAD
- Path
 - .devdev 디렉토리의 kustomization.yaml 을 사용할 예정이므로 dev/dev 선택

[DESTINATION]

DESTINATION

Cluster URL

https://kubernetes.default.svc

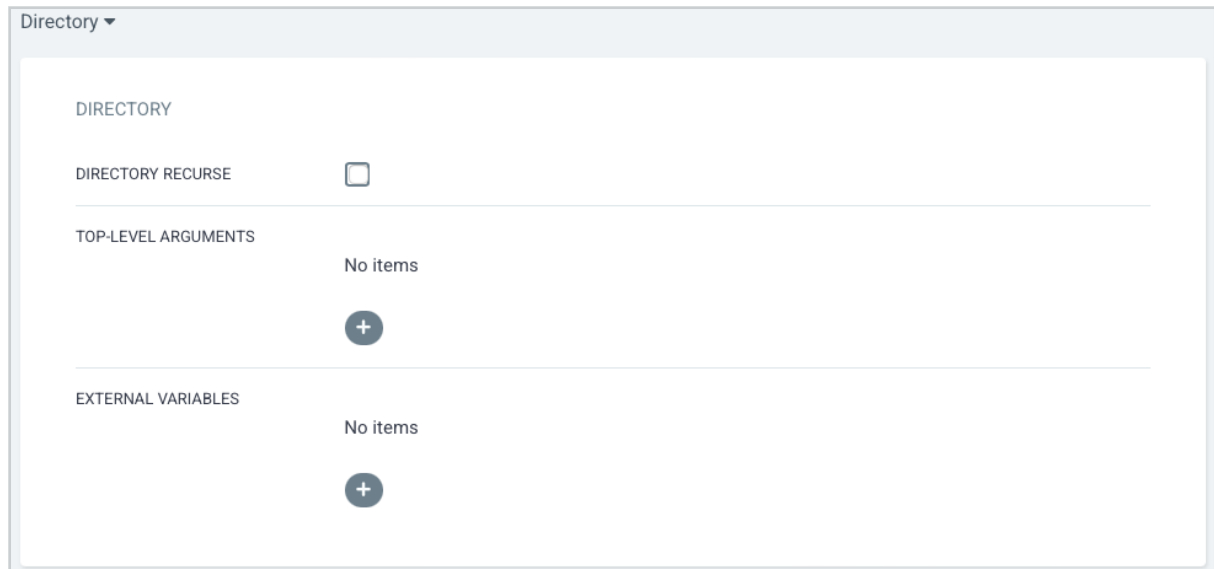
URL ▼

Namespace

dev

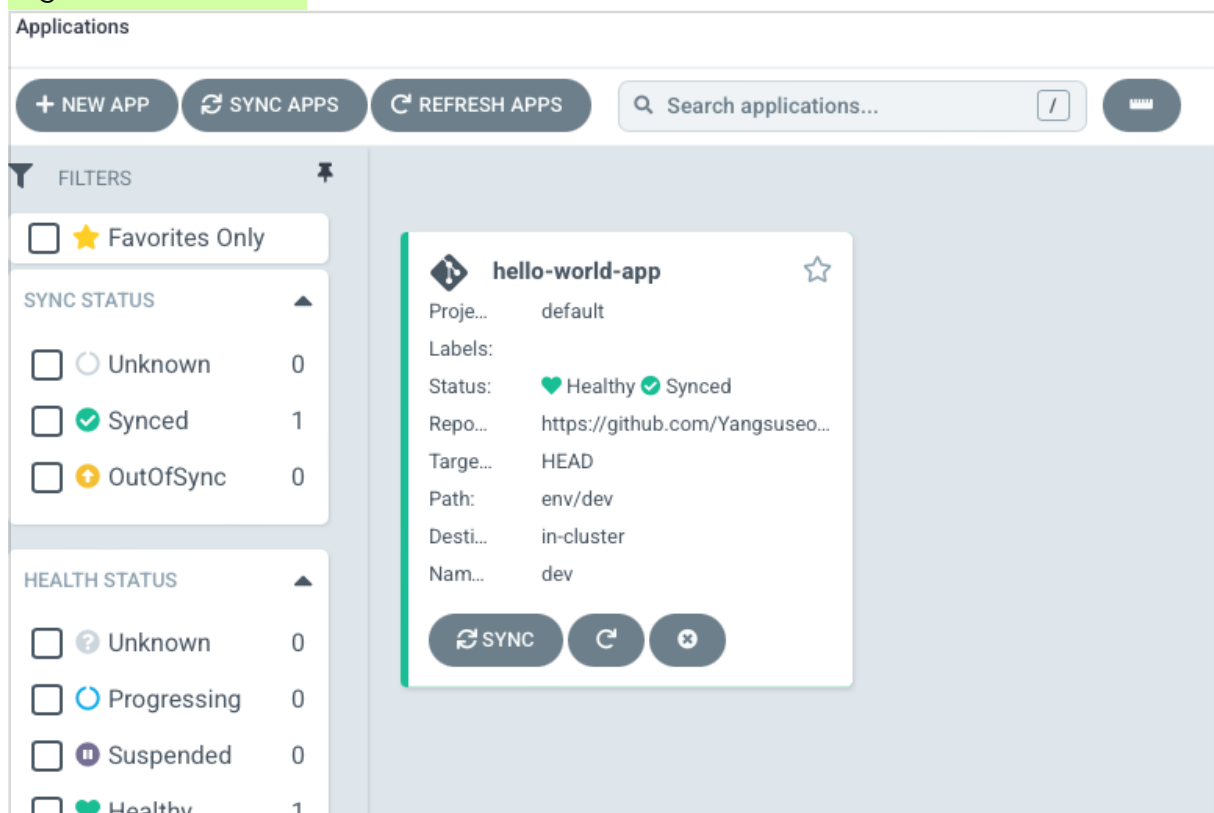
- Cluster URL
 - <https://kubernetes.default.svc>
- Namespace
 - APP을 실행할 namespace 이름 (이번 테스트에서는 dev 사용)

[Kustomize] 를 클릭하여 [Diretory] 로 변경

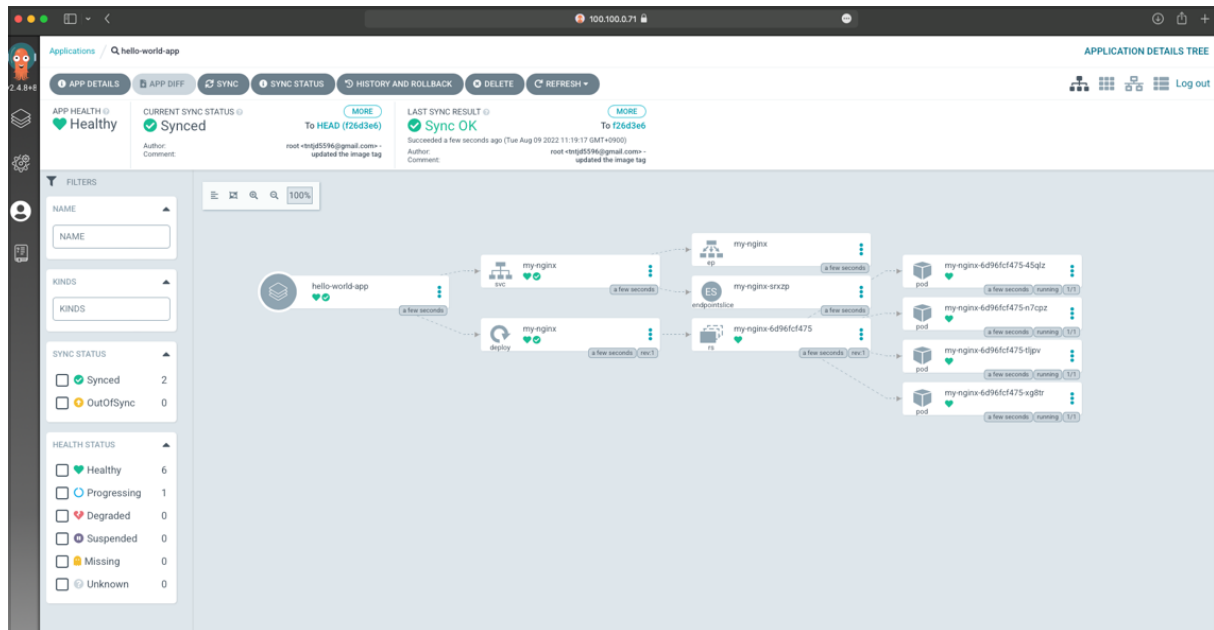


Argo App 상태 확인

Argo Dashboard Main



Hello-world-app 클릭하여 상세정보로 이동



Kubernetes 터미널에서 배포중인 Hello Service에 Curl 요청

```
$ kubectl get svc -n dev
```

```
>>>
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hello	NodePort	10.102.234.143	<none>	80:30577/TCP	68m

```
$ curl localhost:30577
```

```
>>> Hello World!!
```

Jenkins와 Argo 의 CI/CD Pipeline 작동 확인

Git Repository의 index.js 파일 내용 변경

```
const express = require('express')
const app = express()
const port = process.env.PORT || 8080;

app.get('/', (req, res) => {
```

```

    res.send('Hello World!! This Is Trigger Test') # 내용 추가
  })

  app.get('/version', (req, res) => {
    res.send(process.env.VERSION || 'No version')
  })

  app.listen(port, () => {
    console.log(`Example app listening on port ${port}!`)
  })

```

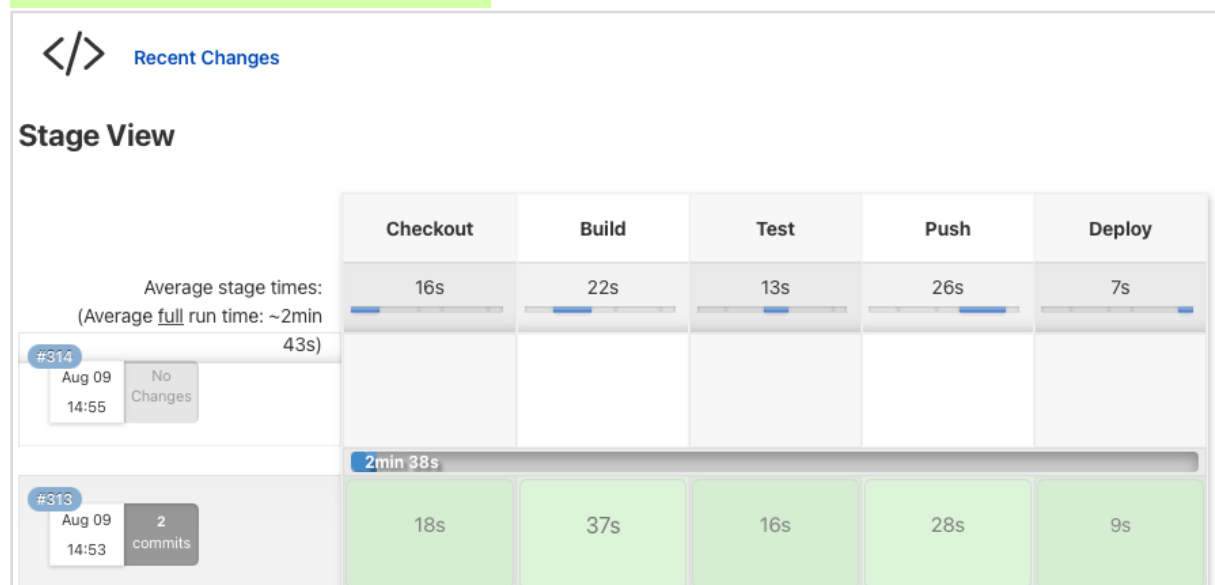
Git Push

```

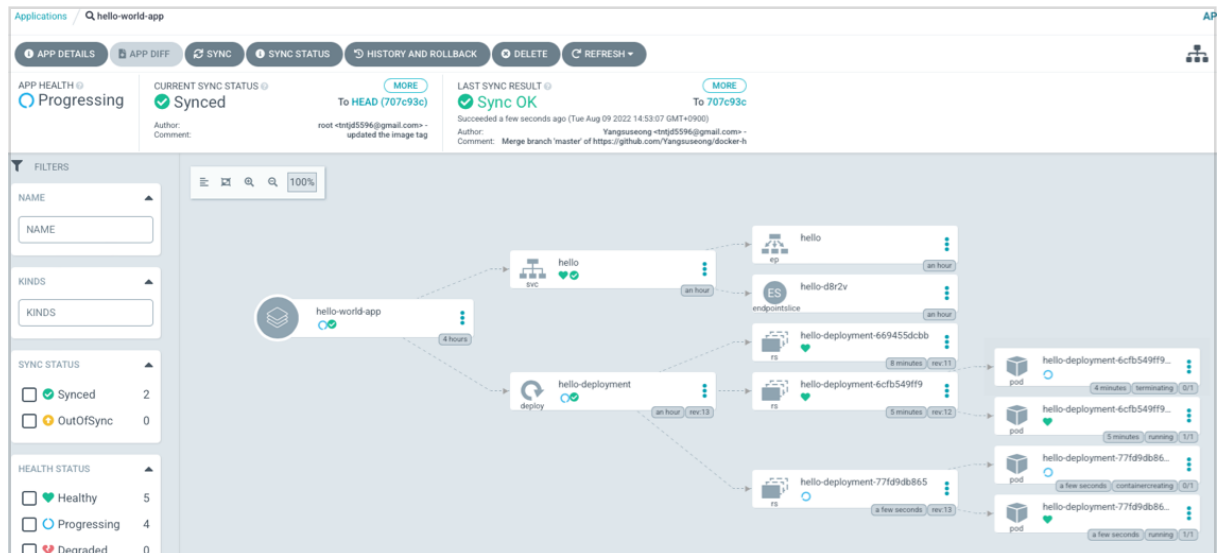
$ git pull
$ git add index.js
$ git commit
$ git push

```

Jenkins 대시보드에서 이미지 Build 확인



빌드 완료 후 Argo Dashboard 에서 Application 상태 확인



- 자동으로 변경된 내용으로 배포가 진행되는 것을 확인

Kubernetes 터미널에서 배포중인 Hello Service에 Curl 요청

```
$ kubectl get svc -n dev
```

```
>>>
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hello	NodePort	10.102.234.143	<none>	80:30577/TCP	68m

```
$ curl localhost:30577
```

```
>>> Hello World!! This Is Trigger Test
```