확장본#1 - 젠킨스의 FreeStyle 로 만드는 개발-상용 환경 배포

조 훈(Hoon Jo)

심근우(Gnu Shim)

문성주(Martin Moon)

5.4.2 Freestyle로 만드는 개발-상용 환경의 연결된 배포

Freestyle을 통해서는 자유로운 형태의 프로젝트를 구현할 수 있습니다. 따라서 기존에 CI 2개 CD 2개의 방식과 같은 단순히 연결되는 구성이 아니라 전 단계의 검증이 완료되어 문제가 없다고 판단될 경우 배포를 진행하는 좀 더 지능적인 배포를 Freestyle로 구현해 보도록 하겠습니다.

일반적으로 현업에서는 애플리케이션을 배포하기 위해 **개발**(Development), **스테이징**(Staging), **상용**(Production)으로 분리하여 배포합니다. 즉 애플리케이션을 개발과 스테이징에서 충분히 검증하고 상용 환경으로 배포하는 것입니다. 이와 같은 과정을 우리는 Freestyle로 구현하려고 하는데 실습을 조금 더 쉽게 진행하기 위해 스테이징은 제외하고 개발과 상용으로만 구성하도록 하겠습니다. 참고로 스테이징은 개발 단계에서 기능 검증이 끝난 애플리케이션을 상용으로 이행하기 전의 중간 단계로 상용과 유사한 환경을 구성하여 안전한 배포가 가능한지 검증하는 단계입니다.

진행에 앞서 개발에서 상용 환경으로 이어지는 CI/CD를 어떻게 젠킨스에서 구현하는지 간략히 요약하겠습니다.

<번호스타일>

1. 깃허브에서 echo-ip를 빌드한 정보가 담긴 파일들을 내려(pull) 받습니다.

2. 받은 파일을 이용해서 컨테이너 이미지를 빌드합니다.

3. 빌드한 이미지를 레지스트리(192.168.1.10:8443)에 echo-ip라는 이름으로 저장(push)합니다.

4. 레지스트리에 저장된 echo-ip 이미지를 dev 네임스페이스에 배포합니다.

5. dev 네임스페이스에 배포한 echo-ip 애플리케이션 상태를 주기적으로 확인하기 위해 Execute Shell에 주기적으로 상태를 확인하는 코드를 입력합니다.

6. 만약 dev 네임스페이스에 배포한 echo-ip 애플리케이션이 정상적으로 동작한다면 해당 애플리케이션은 신뢰할 수 있다고 판단하여 빌드 된 echo-ip 컨테이너 이미지를 prod 네임스페이스에 배포합니다. 만약 정상적으로 동작하지 않는다면 prod 네임스페이스에 echo-ip를 배포하지 않습니다.

</번호스타일>

그러면 이제 실제로 개발단계에서 검증한 후에 상용환경으로 넘어가는 CI/CD 구성을 Freestyle 프로젝트로 만들어 진행해보겠습니다.

<번호스타일>

1. 실습에서 사용하는 격리 환경을 구성하기 위해서 사전에 구성해둔 야믈을 이용하여 dev, prod 네임스페이스(Namespace)를 생성합니다. 이렇게 생성된 네임스페이스를 통해서 **개발**(dev)과 **상용**(prod) 환경을 분리합니다.

<명령>

```
[root@m-k8s ~]# kubectl create ns dev
namespace/dev created
[root@m-k8s ~]# kubectl create ns prod
namespace/prod created
[root@m-k8s ~]# kubectl get namespaces
NAME              STATUS   AGE
default           Active   3d6h
dev               Active   7s
kube-node-lease   Active   3d6h
kube-public       Active   3d6h
kube-system       Active   3d6h
metallb-system    Active   2d21h
prod              Active   7s
```

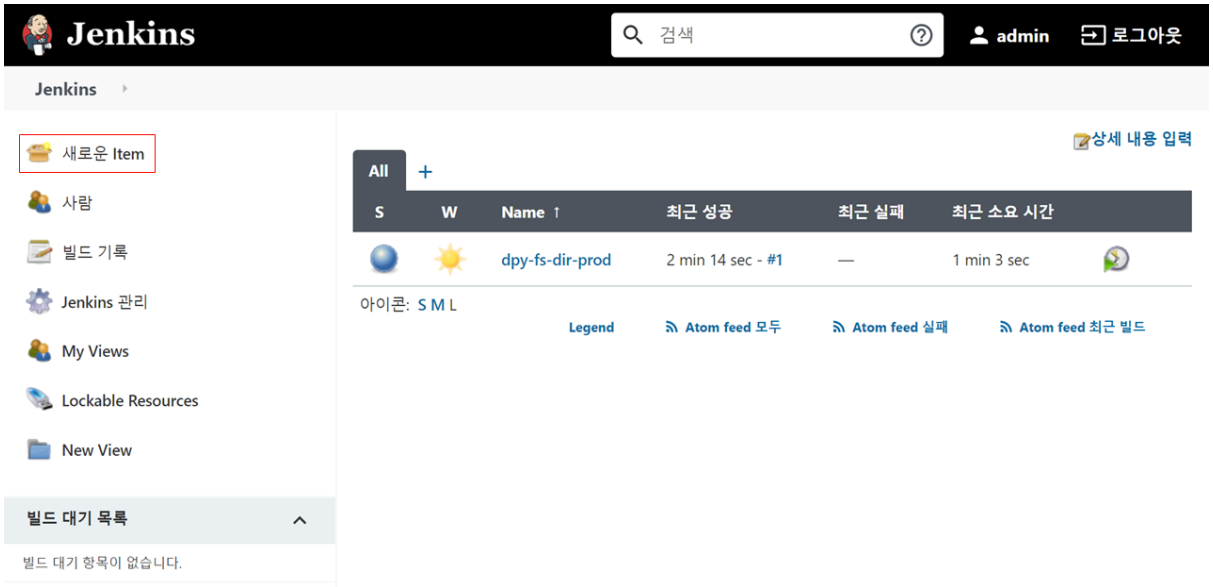
</명령>

<참고> **개발과 상용 환경을 분리하는 네임스페이스란?**

네임스페이스는 물리적으로 동일한 클러스터 내에서 가상으로 분리된 공간입니다. 네임스페이스는 다른 네임스페이스와 영향을 미치지 않고 동작할 수 있어 네임스페이스는 팀 혹은 프로젝트 단위로 분리하여 구동할 수 있는 환경을 제공하며 하나의 네임스페이스 내에서 사용 중인 오브젝트의 이름은 다른 네임스페이스와 동일하여도 관계가 없기 때문에 현재 실습과 같이 서로 간에 동일한 환경을 유지하면서 비교하며 구동해야 하는 개발 및 상용 환경에 효과적으로 사용할 수 있습니다. 하지만 실제로 서비스를 운영할 때는 이에 대해서 조금 주의가 필요합니다. 쿠버네티스 클러스터에서 네임스페이스는 자원을 공유하여 사용하기 때문에 특정 네임스페이스에서 CPU나 메모리 등을 과도하게 사용할 경우 다른 네임스페이스에서 사용할 수 있는 자원에 영향을 줄 수 있습니다. 또한 쿠버네티스 클러스터가 설치된 호스트 시스템의 문제는 호스트 위에 동작하는 모든 네임스페이스에 영향을 줄 수 있기 때문에 이에 대한 영향을 최소화하기 위해서 업무 중요도에 따라 쿠버네티스 클러스터를 여러 개 생성하여 분산 배치하는 것을 고려해야 합니다.

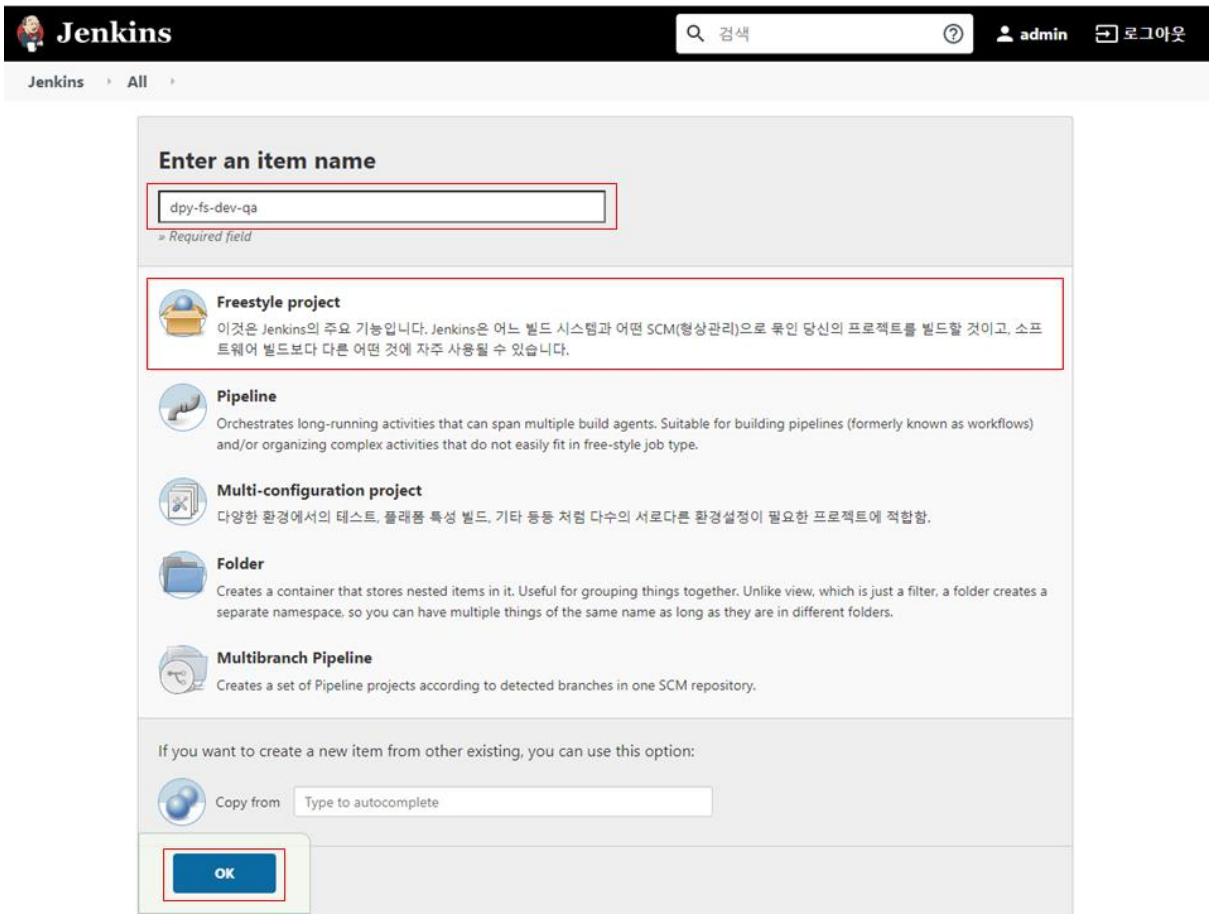
</참고>

2. 젠킨스 홈 화면으로 이동한 후에 새로운 Item 버튼을 눌러서 개발과 상용을 위한 새로운 아이템을 생성하는 화면으로 넘어갑니다.



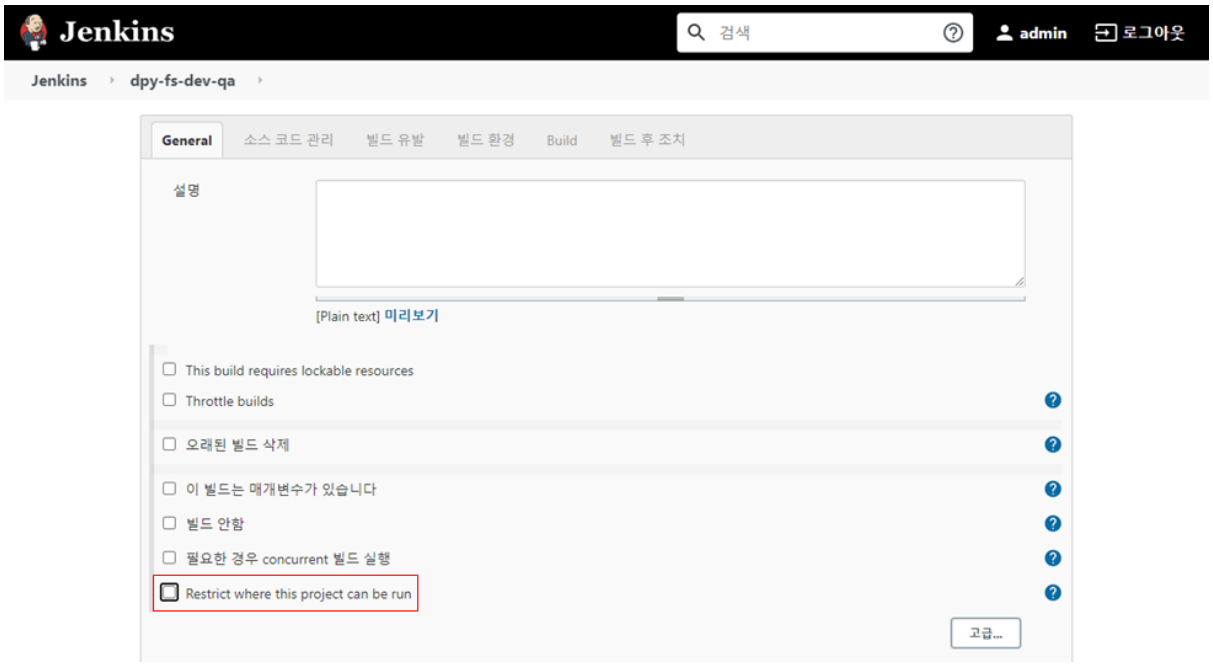
3. 개발에서 상용으로 이어지는 CI/CD 파이프라인 중에서 개발 품질 검증 단계를 위한 아이템을 만들기 위해서 `Freestyle project`를 선택하고 이름은 `dpy-fs-dev-qa`로 입력한 후에 `OK` 버튼을 누릅니다.

그림 1 개발 품질 검증 아이템 생성



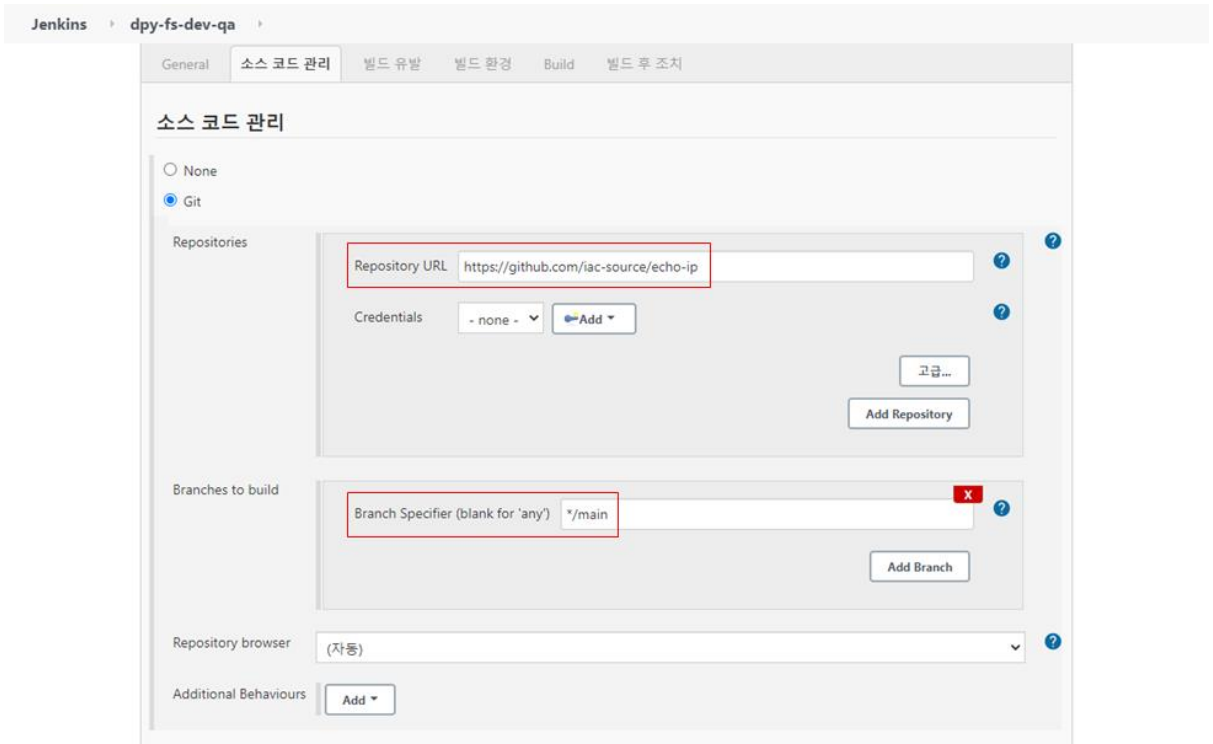
4. Pod가 배포되는 위치를 제한하는 `Restrict where this project can be run` 옵션을 해제합니다.

그림 2 Restrict where this project can be run 체크 해제



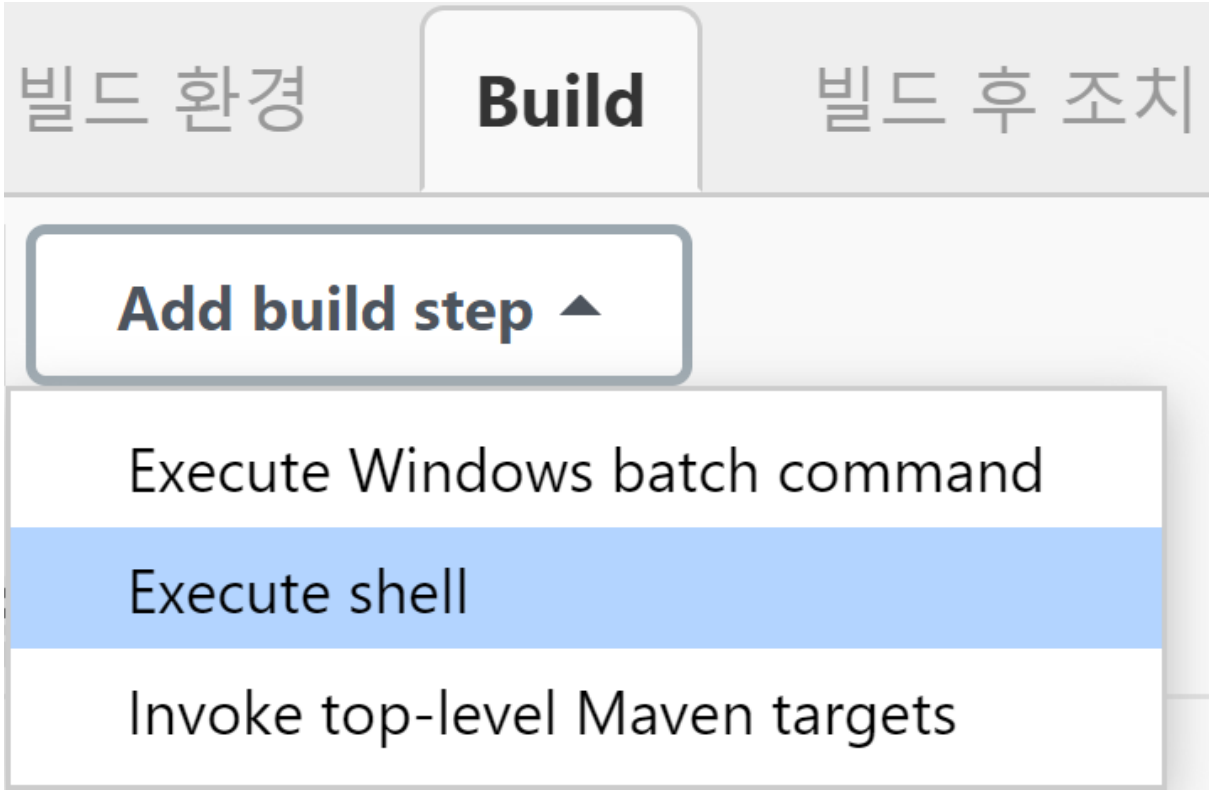
5. **소스 코드 관리**에서 `Git`을 선택하고 나타난 화면에서 빌드를 위한 소스코드를 불러오기 위해 아래와 같이 **Repository URL**은 `https://github.com/iac-source/echo-ip`로 설정하고 **Branch Specifier**를 `*/main`으로 설정합니다.

그림 3 소스코드 저장소 설정



6. 소스 코드를 불러온 후 빌드 작업을 작성하기 위해 Build 탭을 눌러 **Build 단계**로 이동하겠습니다. Build 단계에서 Add build step을 누르고 Execute shell을 누르면 검증 작업에 필요한 셸 스크립트를 입력할 수 있는 공간이 나타납니다.

그림 4 빌드 중에 Execute shell 단계를 추가



7. 사전에 구성해둔 빌드 스크립트를 cat 명령으로 출력한 후 복사해서 추가한 **Execute shell**에 붙여 넣습니다. (주의: #! /usr/bin/env bash 또한 붙여 넣어야 합니다.)

<명령>

```
[root@m-k8s ~]# cat ~/_Book_k8sInfra/ch5/5.4.2/deploy-dev-qa-101.freestyle
#!/usr/bin/env bash
docker build -t 192.168.1.10:8443/echo-ip .
[출력 생략]
```

</명령>

그림 5 배포할 코드를 Execute shell에 입력

복사하여 웹 브라우저 화면에 붙여 넣는 코드는 다음과 같습니다.

<코드> **deploy-dev-qa-101.freestyle**

```
1 #!/usr/bin/env bash
2 docker build -t 192.168.1.10:8443/echo-ip .
3 docker push 192.168.1.10:8443/echo-ip
4 kubectl apply -f https://raw.githubusercontent.com/IaC-Source/dev-prod/main/echo-ip-dev.yaml
5 for try in {1..30}
6   do
7     export ready=$(kubectl get deployment --selector=app=fs-echo-ip-dev -n dev -o jsonpath --template="{.items[0].status.readyReplicas}")
8     echo "trying $try: ready $ready";
9     if [ "$ready" == "1" ]; then
10      exit 0
11    fi
12    sleep 1
13  done
14 exit 1
```

</코드>

배포를 위해 복사 후 붙여지는 코드는 다음과 같은 작업을 수행합니다.

<점스타일>

- **2 ~ 3 번 줄** 소스 코드 저장소에서 받아온 소스 코드를 도커를 이용해 컨테이너 이미지로 빌드하고 저장소로 푸시합니다.

- **4 번 줄** 사전에 구성된 배포 야믈 파일을 사용하여 dev 네임스페이스에 echo-ip 애플리케이션을 배포하고 이를 서비스로 노출합니다.
- **5 ~ 14 번 줄** 30번을 반복하면서 현재 배포한 echo-ip Deployment 상태가 Ready라면 배포를 정상종료(`exit 0`)합니다. 그렇지 않다면 1초를 대기한 후 다시 echo-ip Deployment를 확인합니다. 만약 30번의 체크 이후로 echo-ip Deployment 상태가 Ready가 아니면 배포를 실패처리(`exit 1`) 시켜 상용 배포를 수행하지 않도록 설정합니다.

</점스타일>

deploy-dev-qa-101.freestyle에서 호출하는 **echo-ip-dev.yaml**(4번 줄)에는 기존에 echo-ip와 다르게 품질 테스트를 위해서 `readinessProbe` 를 추가하였습니다.

<코드> **echo-ip-dev.yaml**

[중략]

```
17    spec:
18      containers:
19      - image: 192.168.1.10:8443/echo-ip
20        name: fs-echo-ip-dev
21        readinessProbe:
22          failureThreshold: 3
23          httpGet:
24            path: /
25            port: 80
26            scheme: HTTP
27          periodSeconds: 10 # 체크하는 주기
28          successThreshold: 1 # 요구하는 성공 횟수
29          timeoutSeconds: 1 # 응답 요구 시간
```

[생략]

</코드>

`readinessProbe`는 컨테이너가 정상적으로 사용자의 요청을 처리할 수 있는 상태인지 진단할 수 있는 기능으로 HTTP 요청을 통한 확인, 특정 파일의 존재 유무를 통한 확인 등 여러가지 진단 방법이 있습니다. 개발된 애플리케이션을 검증하는 단계에서는 dev 네임스페이스에 배포되는 echo-ip 애플리케이션이 정상적으로 실행되었는지 진단하기 위해 사용합니다. 이와 유사한 livenessProbe라는 기능이 있는데 `readinessProbe`와 다르게 컨테이너 자체의 실행 유무만을 확인합니다. 따라서 현재 실습에서는 정상적으로 HTTP 요청을 처리하는 것을 확인할 필요가 있기 때

문에 readinessProbe로 컨테이너의 정상 동작을 확인하였습니다.

<참고> **echo-ip-dev.yaml와 echo-ip-prod.yaml의 소스 위치**

각 소스는 https://github.com/IaC-Source/dev-prod에 위치해 있습니다. 경로 수정이 꼭 필요하기 때문에 반드시 참고하시기 바랍니다.

</참고>

<점스타일>

- **17 ~ 20번 줄** 빌드한 echo-ip 이미지를 이용해서 테스트를 위한 애플리케이션을 배포하고 이름은 fs-echo-ip-dev로 설정합니다.
- **21 ~ 29번 줄** readinessProbe의 상태 확인 기능을 이용해서 HTTP 요청을 컨테이너의 '/' 경로로 HTTP 요청을 보냈을 때 응답을 한다면 Pod가 정상적인 상태라고 판단합니다. 상태 확인을 위한 세부 조건으로 3가지 항목(periodSeconds, successThreshold, timeoutSeconds)을 모두 만족해야 합니다.

<점스타일>

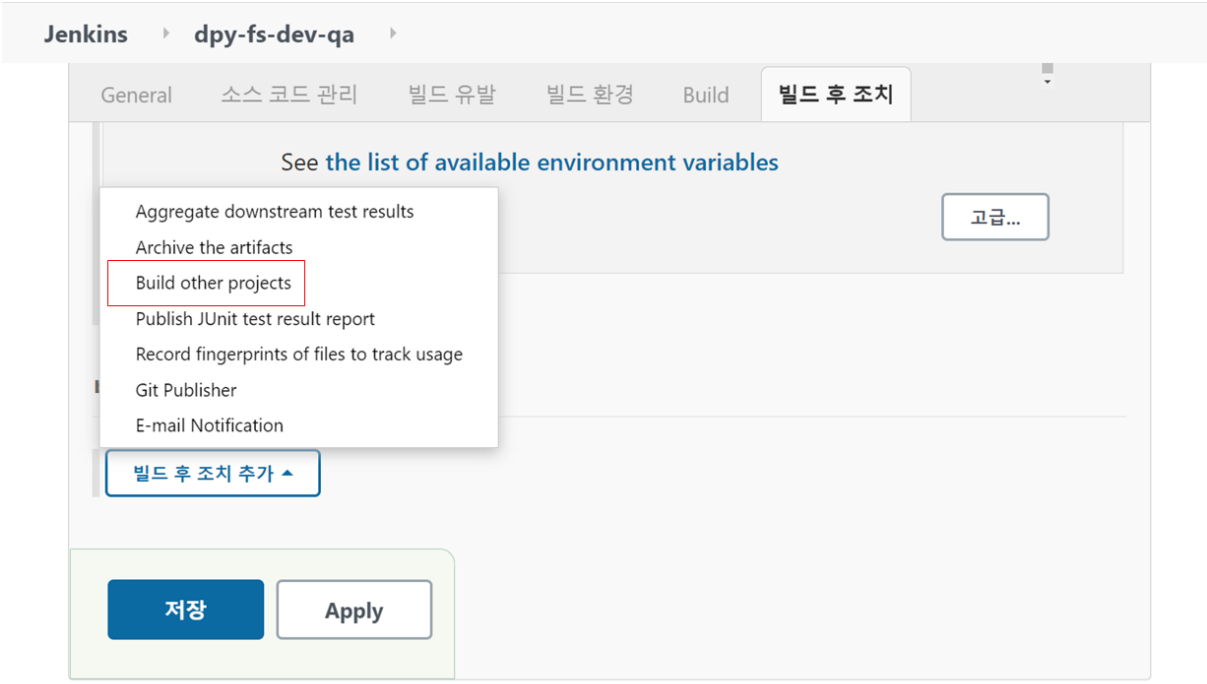
<참고> **젠킨스의 freestyle project에서 #!/usr/bin/env bash(셔뱅, shebang)이 필요한 이유**

젠킨스는 freestyle project를 통해 스크립트를 수행할 때 실행하는 셸 스크립트에 대해서 셸 스크립트를 해석하는 인터프리터가 설정되어 있지 않습니다. 셸 스크립트 인터프리터가 지정되지 않으면 젠킨스 freestyle project 빌드 작업의 단계에서 if, for과 같은 여러 셸 스크립트가 문법을 인식하지 못해 사용할 수 없습니다. 이런 문제를 해결하기 위해 셔뱅을 지정하여서 셸 스크립트의 문법을 해석할 수 있도록 설정을 한 것입니다.

</참고>

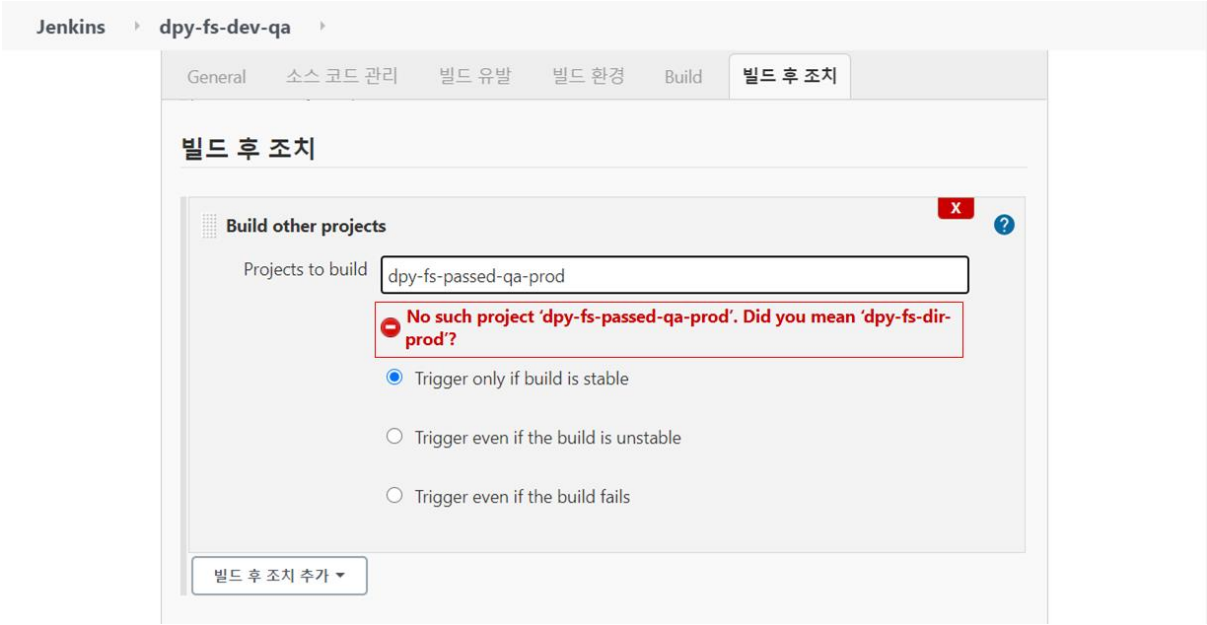
8. 다음으로 정상적으로 품질 검사가 완료될 때 prod 네임스페이스로 echo-ip 애플리케이션을 배포하도록 하기 위해 **빌드 후 조치** 탭에서 빌드 후 조치 추가 메뉴를 선택한 다음 build other projects 작업을 지정하겠습니다.

그림 6 빌드 이후에 다른 프로젝트를 추가 빌드하도록 선택



9. 연관 프로젝트 빌드하기(build other projects)를 선택한 후에 **Projects to build**의 이름을 dpy-fs-passed-qa-prod로 입력하면 해당 프로젝트가 존재하지 않는 다는 메시지를 확인할 수 있습니다.

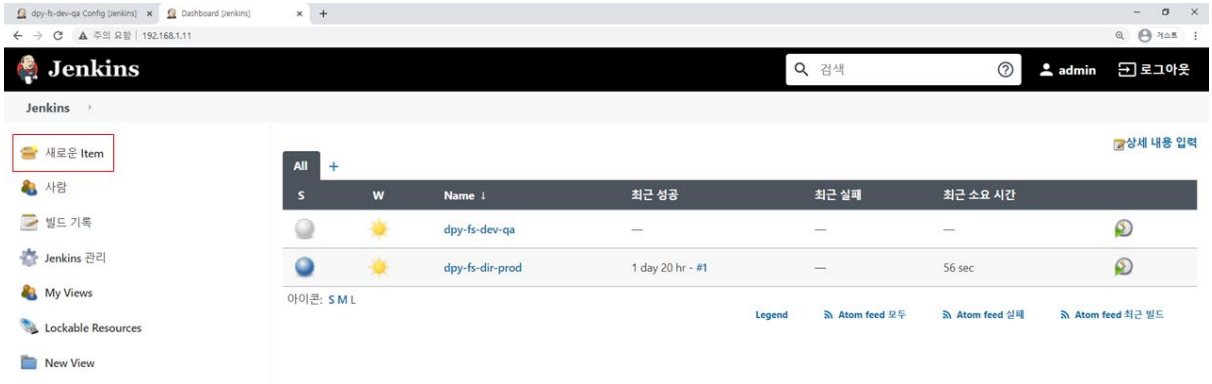
그림 7 추가한 프로젝트가 존재하지 않는다는 메시지



10. 이 문제를 해결하기 위해서 추가 브라우저 탭을 열고 192.168.1.11를 입력하여 젠킨스 홈 화면으로 이동합니다. 그리고 새로운 탭에서 품질 검증이 끝난 후에 배포를 위한 아이템을 생성하

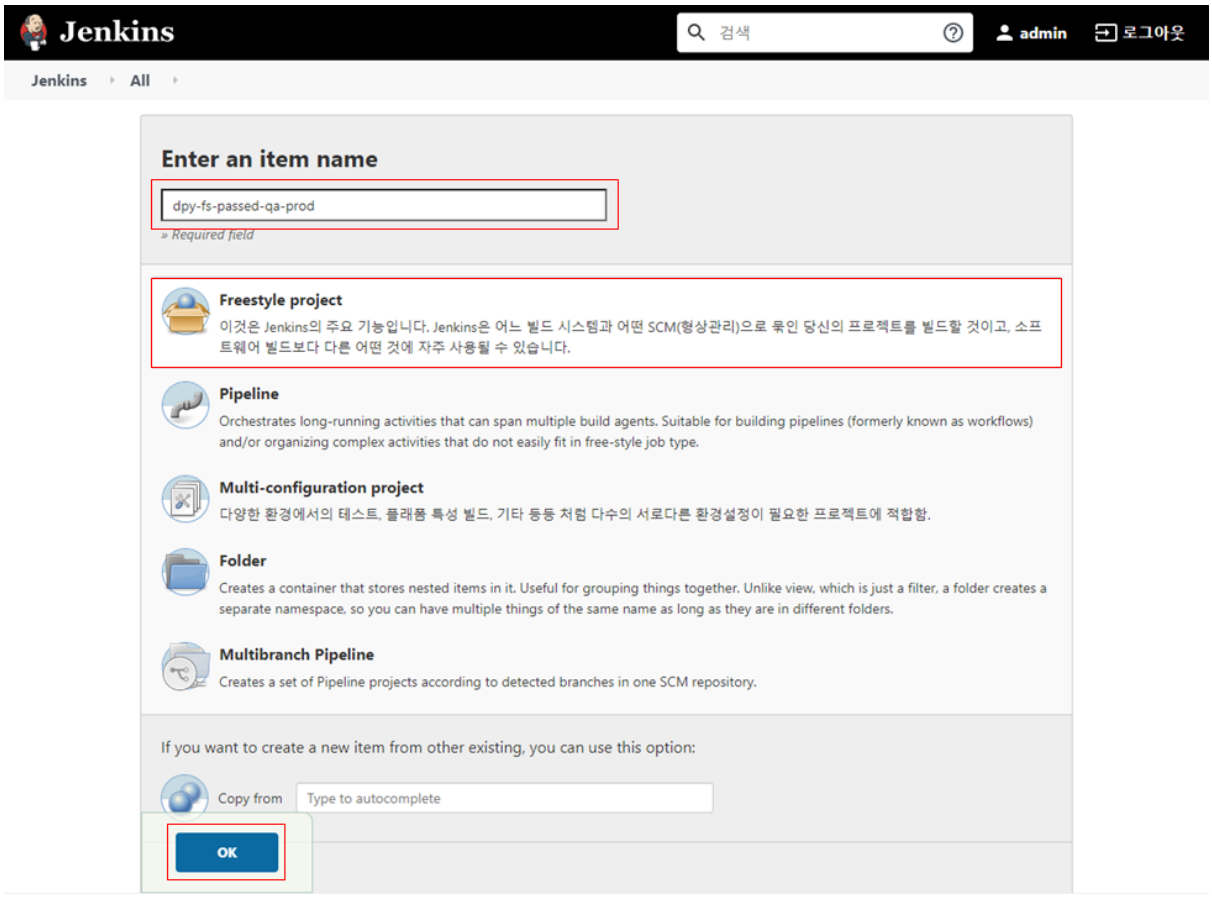
기 위해서 새로운 Item을 클릭합니다.

그림 8 품질 검증이 끝난 후 배포를 위해서 2번째 탭에서 새로운 Item을 클릭



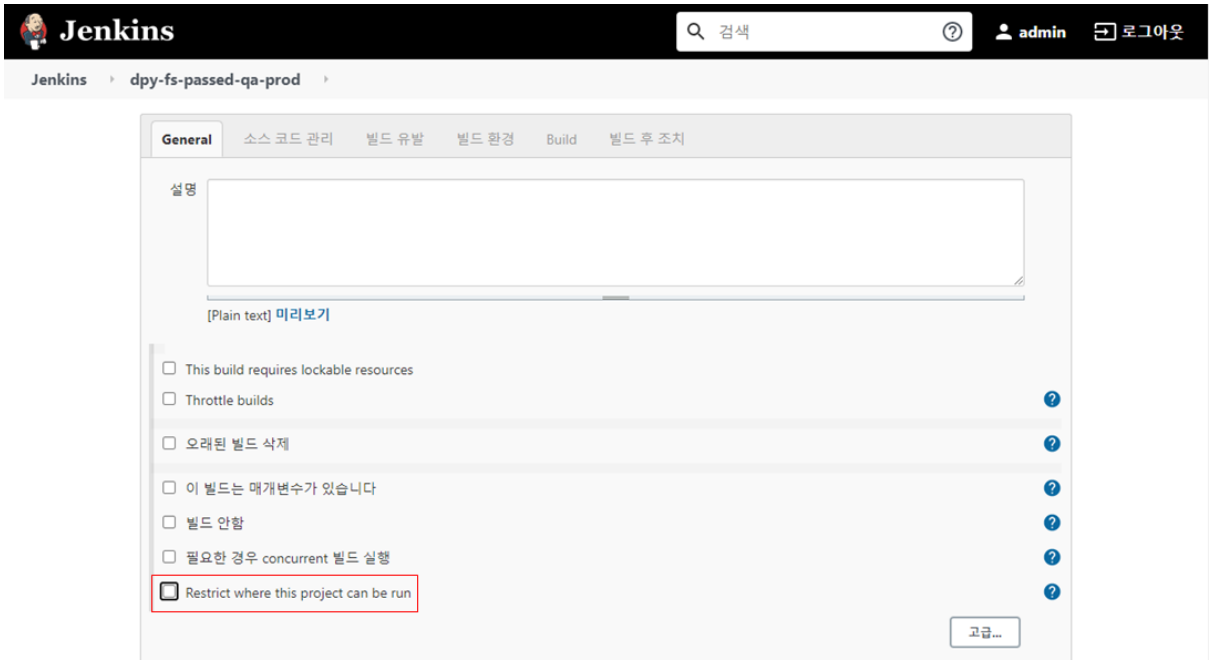
11. 품질 검증이 끝난 애플리케이션을 배포하기 위해 Freestyle project를 선택하고 dpy-fs-passed-qa-prod로 이름을 입력한 후에 OK 버튼을 누릅니다.

그림 9 품질 검증이 끝난 애플리케이션을 배포하기 위한 아이템 설정



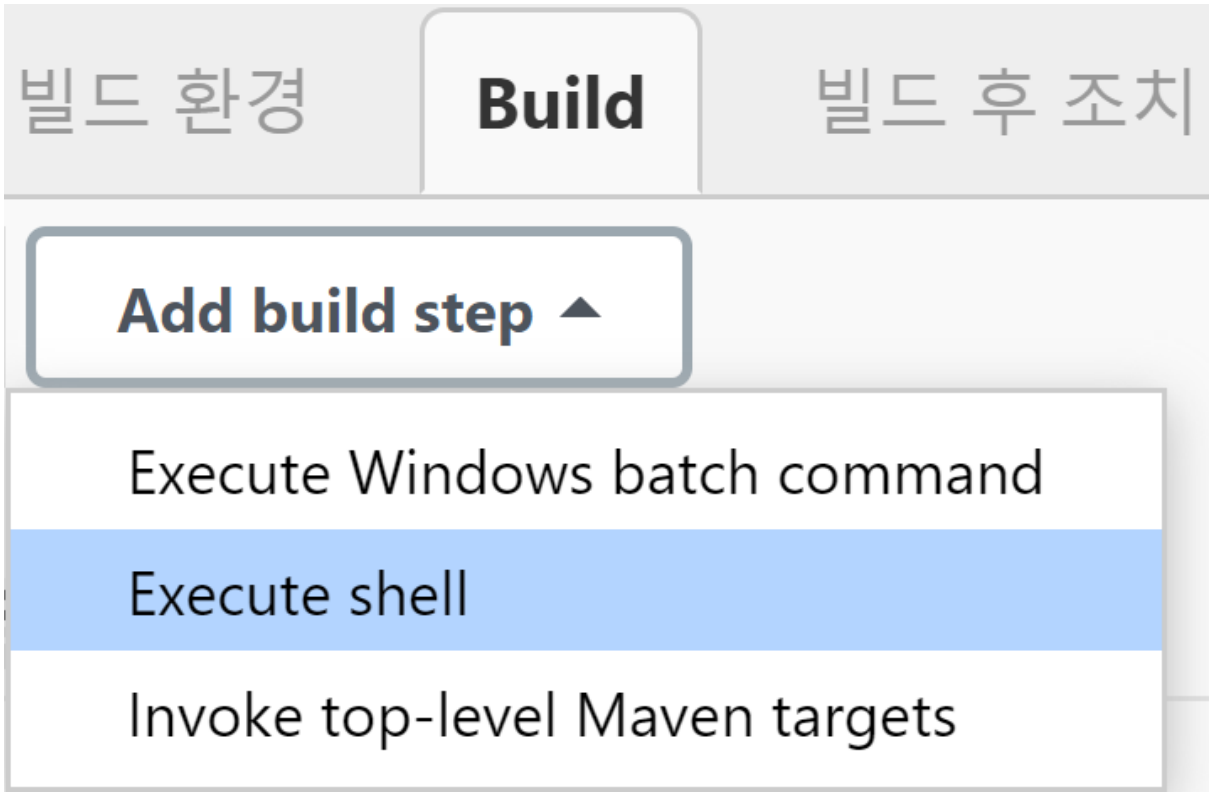
14. Restrict where this project can be run 체크박스를 해제합니다.

그림 10 Restrict where this project can be run 체크 해제



15. **dpy-fs-passed-qa-prod** 프로젝트는 앞선 **dpy-fs-dev-qa** 프로젝트에서 빌드된 애플리케이션의 검증이 성공적으로 완료된 경우에 실행됩니다. 따라서 이 프로젝트는 소스 코드를 가지고 와서 빌드하는 작업을 수행하지 않고 이미 검증이 완료된 애플리케이션 바로 배포합니다. 빌드하지 않고 바로 배포 명령을 작성하기 위해 **Build 단계**의 `Add build step`을 누르고 `Execute Shell`을 눌러 배포 명령을 입력합니다.

그림 11 Execute shell 빌드 단계 추가



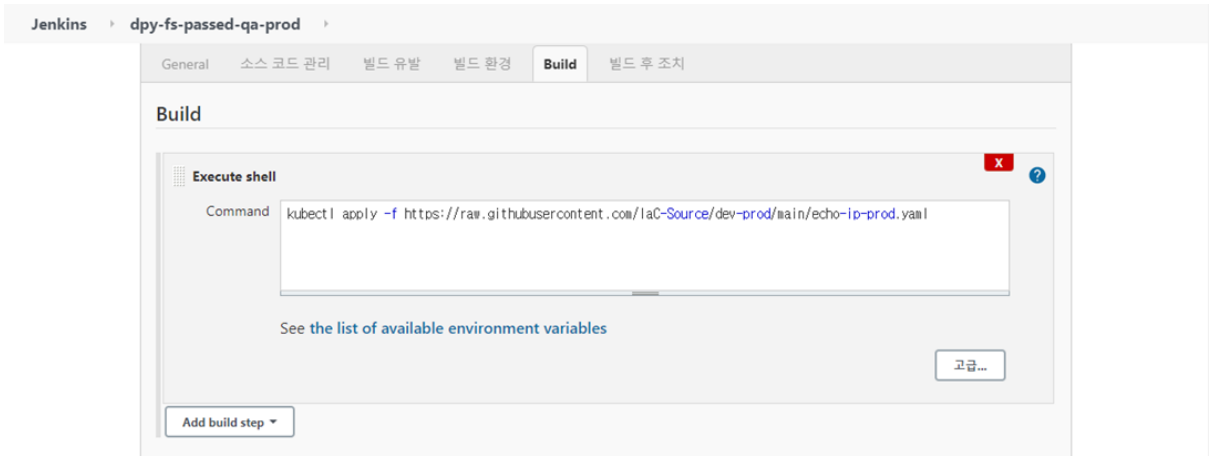
16. 추가한 build 단계에 검증을 성공한 echo-ip 애플리케이션 배포를 위한 스크립트를 입력하겠습니다. `cat` 명령으로 입력된 스크립트를 출력한 후 복사하여 젠킨스 화면에 붙여 넣겠습니다.

<명령>

```
[root@m-k8s ~]# cat ~/_Book_k8sInfra/ch5/5.4.2/deploy-qa-passed-prod.freestyle
kubectl apply -f https://raw.githubusercontent.com/IaC-Source/dev-prod/main/echo-ip-prod.yaml
```

</명령>

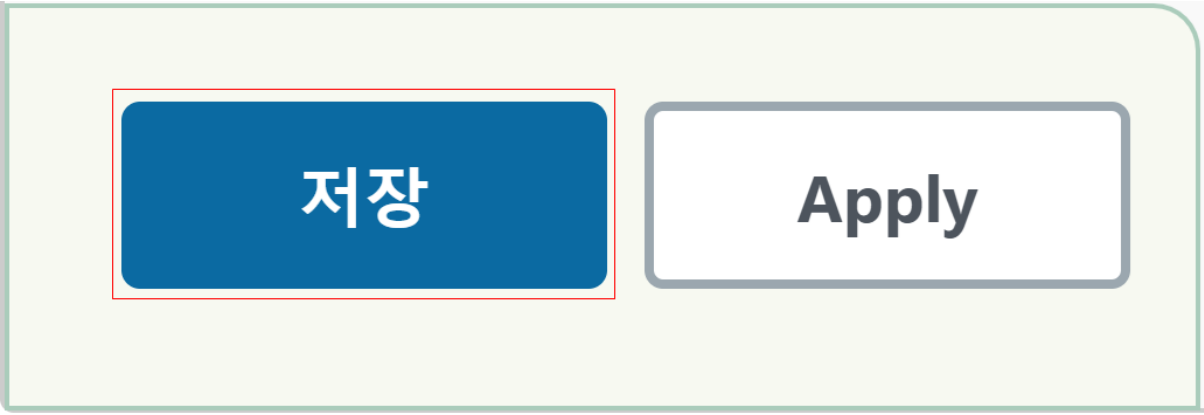
그림 12 품질 검증이 끝난 후에 echo-ip-prod.yaml을 배포하도록 설정



17. 배포에 사용하는 **echo-ip-prod.yaml**은 **echo-ip-dev.yaml**에서 `readinessProbe` 항목을 제외하면 동일합니다. 즉 echo-ip-dev.yaml을 통해 정상적으로 품질 검증을 마쳤기 때문에 상용으로 배포되는 환경에서는 해당 테스트 구문을 제외한 것입니다.

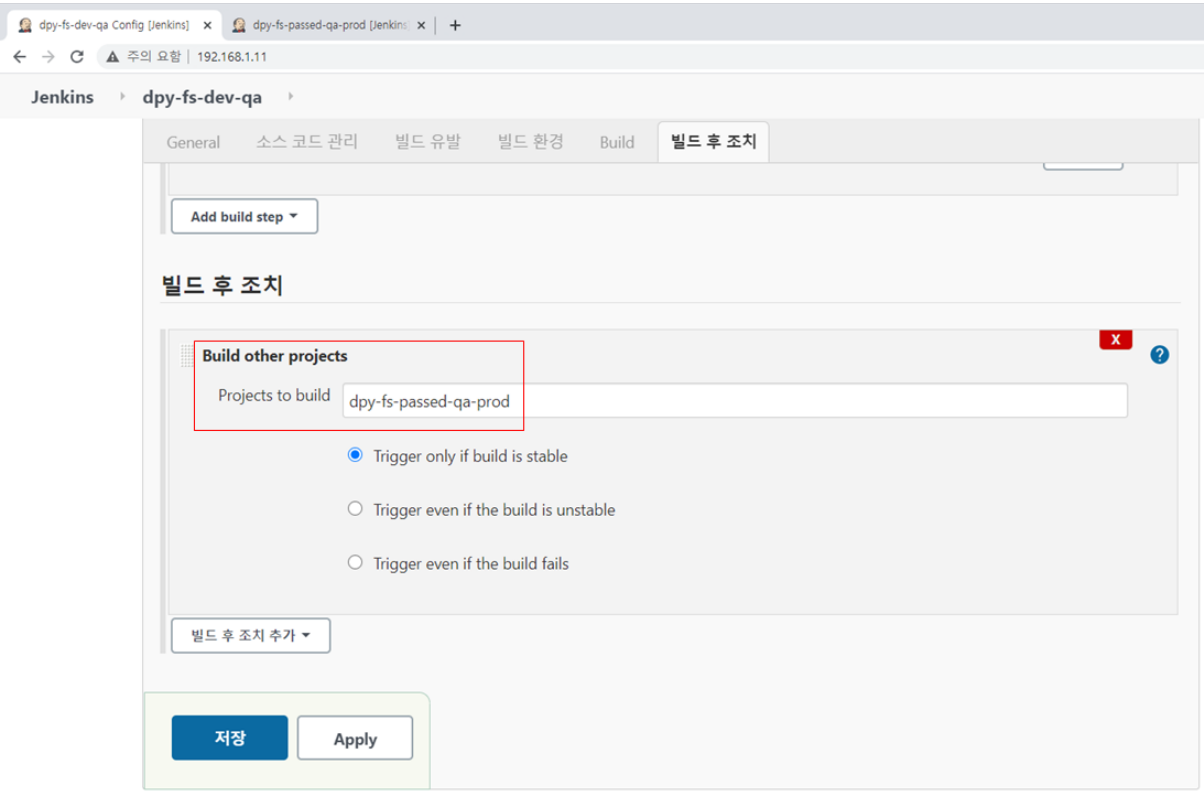
이제 상용 배포를 위해서 설정한 **dpy-fs-passed-qa-prod** 프로젝트를 저장합니다.

그림 13 두 번째 탭의 dpy-fs-passed-qa-prod 프로젝트 설정 저장



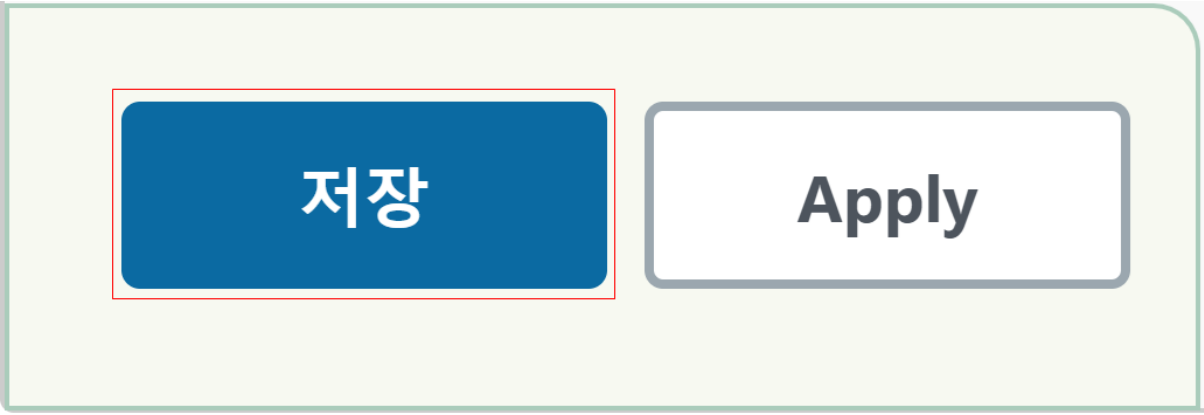
18. 설정을 저장했으니 다시 브라우저의 첫 번째 탭으로 돌아와서 `dpy-fs-passed-qa-prod`를 다시 입력하면 다음 그림과 같이 에러가 사라지고 설정을 완료할 수 있는 화면이 나타나는 것을 확인할 수 있습니다.

그림 14 dpy-fs-dev-qa 빌드 성공 후에 실행될 프로젝트를 다시 입력



19. 모든 설정이 완료되었으니 **dpy-fs-dev-qa** 프로젝트를 저장합니다.

그림 15 첫 번째 탭의 dpy-fs-dev-qa 프로젝트 설정 저장



20. 이제 저장 후에 나오는 품질 검증 프로젝트인 **dpy-fs-dev-qa** 프로젝트에서 Build Now를 누르고 실행이 정상적으로 종료되어 파란색 원으로 변경되는 것을 확인합니다. 그리고 나서 두 번째 탭으로 이동하여 아무 것도 누르지 않고 Build History를 지켜봅니다. 그러면 자동으로 실행되어 완료되는 **dpy-fs-passed-qa-prod** 프로젝트를 확인할 수 있습니다. 이는 품질 검증이 끝나서 검증된 애플리케이션을 배포하는 상용 환경이 구성되어 진 것으로 볼 수 있습니다.

그림 16 dpy-fs-dev-qa의 하위 프로젝트로 실행되는 dpy-fs-passed-qa-prod

21. 네임스페이스 내의 오브젝트를 조회할 수 있는 -n 옵션과 각각의 오브젝트를 구분할 수 있는,(쉼표)를 사용하여 각 네임스페이스로 별로 배포된 Pod와 서비스를 확인합니다.

<명령>

```
[root@m-k8s ~]# kubectl get pod,service -n dev
NAME                                  READY   STATUS    RESTARTS   AGE
pod/fs-echo-ip-dev-6dc4685fb5-p9qdh   1/1     Running   0          6m46s

NAME                          TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)          AGE
service/fs-echo-ip-dev-svc    LoadBalancer   10.98.216.25   192.168.1.12   8080:31659/TCP   6m46s
[root@m-k8s ~]# kubectl get pod,service -n prod
NAME                                   READY   STATUS    RESTARTS   AGE
pod/fs-echo-ip-prod-8694685599-scckd   1/1     Running   0          3m25s

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

```
service/fs-echo-ip-prod-svc    LoadBalancer    10.107.114.4    192.168.1.13    8080:32070/TCP    3m24s
```

</명령>

22. 다음 실습을 구성하기 위해 이번 실습에서 사용한 네임스페이스를 삭제함으로써 배포했던 echo-ip 애플리케이션들을 모두 한 번에 삭제하겠습니다. 네임스페이스를 삭제함으로써 네임스페이스로 속해 있는 여러 쿠버네티스 오브젝트들이 모두 손쉽게 삭제됩니다.

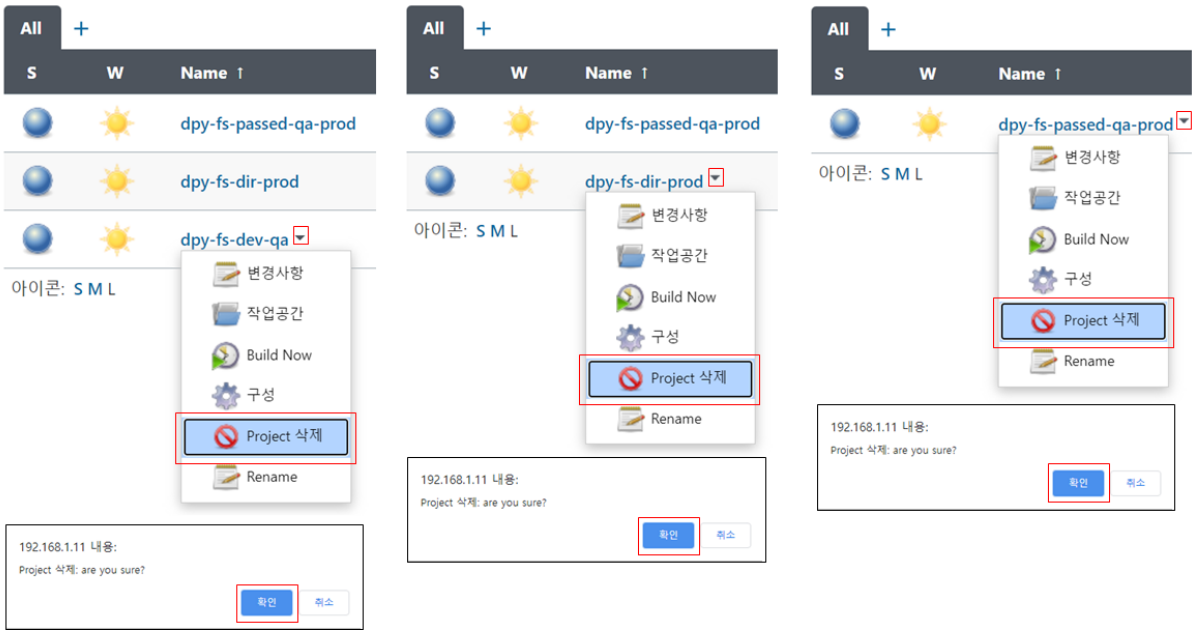
<명령>

```
[root@m-k8s ~]# kubectl delete namespaces dev prod
namespace "dev" deleted
namespace "prod" deleted
```

</명령>

23. 이후 실습에서 혼동을 피하기 위해서 배포한 프로젝트 3개를 모두 삭제합니다.

그림 17 배포한 프로젝트 3개 삭제



</번호스타일>

Freestyle 프로젝트는 젠킨스의 웹 화면에 직접 셸 스크립트를 입력하기 때문에 빌드 작업의 명령어에 변경이 있을 경우 작업 관리 및 변경 사항의 추적이 쉽지 않습니다. 이러한 내용을 파일 레벨로 관리하고 이를 변경 관리를 도와주는 깃허브 저장소를 함께 사용한다면 어떨까요?

이와 같이 구성한다면 이력 관리 및 변경 추적 그리고 애플리케이션 통합이 매우 수월할 것입니다. 이렇게 파일 레벨로 CI/CD를 구성하게 도와주는 아이템이 Pipeline입니다. 그러면 바로 Pipeline에 대해서 알아보겠습니다.