

1. 사용하기



argocd 라는 namespace에 ArgoCD가 이미 설치되어 있다고 가정합니다.

ArgoCD Image Updater를 사용하려면 관련된 리소스들이 ArgoCD와 연계되어 있어야 합니다.

0. Argo CD Image Updater 개요

1. 개요

- [공식 문서](#)
- ArgoCD Image Updater는 Kubernetes에 배포되어 있는 컨테이너 이미지의 새로운 버전을 확인하고, ArgoCD를 통해 새로운 버전의 이미지를 클러스터에 배포합니다.
- 사용법은 간단한데, ArgoCD Application 리소스에 대해 ArgoCD Image Updater를 적용하고 싶다면 알맞은 annotation을 추가해주면 됩니다.
- ArgoCD의 Sync Policy에 따라 ArgoCD Image Updater는 ArgoCD를 통해 새로운 이미지를 배포하거나, Out of Sync로 상태를 변경할 수 있습니다.



ArgoCD Image Updater는 설정된 이미지의 registry를 2분에 1번씩 polling해 새로운 버전의 이미지가 배포되었는지를 검사합니다.

2. 필수 요구사항

- ArgoCD Image Updater를 통해 새로운 이미지 버전을 추적하고 싶은 애플리케이션들은 무조건 Argo CD를 통해 관리되어야 합니다. 그렇지 않으면 사용할 수 없습니다.
- ArgoCD Image Updater는 Kustomize 또는 Helm을 사용해 만들어지는 manifest들의 컨테이너 이미지만 갱신시킬 수 있습니다. 특히 Helm의 경우, 이미지의 태그를 `image.tag` 로 가져와 사용해야 합니다.
- 마지막으로, 이미지를 pull 하기 위한 secret들은 ArgoCD Image Updater가 실행되고 있는 cluster와 동일한 cluster에 위치해야 합니다.

3. 동작 방식 - 이미지 업데이트

- [공식 문서](#)
- 전반적인 업데이트 과정
 - ArgoCD Image Updater는 ArgoCD Application으로 관리되는 컨테이너 이미지들의 새로운 버전을 가져와 배포합니다. 그리고 종합적인 workflow는 아래와 같습니다.
 1. ArgoCD의 Application 리소스들을 Kubernetes 또는 ArgoCD API를 사용해 스캔합니다.
 2. 발견된 Application들 중 `argocd-image-updater.argoproj.io/image-list` annotation이 적용되어 있는 Application을 선택합니다.
 3. 위에서 발견된 각 이미지들에 대해 ArgoCD Image Updater가 우선 해당 이미지가 배포되어 있는지를 확인합니다. 이는 registry name을 포함해 이미지의 전체 이름을 통해 확인합니다.
 4. 만약 ArgoCD Image Updater가 특정 이미지의 새로운 버전을 검사하기로 결정하면, 해당 이미지의 container registry에 연결해 새로운 버전이 있는지를 확인합니다.(업데이트 전략)
 5. 새로운 버전의 이미지가 발견되면, ArgoCD Image Updater는 지정된 업데이트 방식을 통해 기존 애플리케이션에 새로운 버전의 이미지를 배포하려 합니다.
 - Sync policy, image update
 - ArgoCD Image Updater는 ArgoCD가 클러스터 내의 manifest를 갱신할 것을 요구합니다. 따라서 ArgoCD application의 automatic sync가 활성화되어 있을 때 가장 잘 동작합니다.
 - Rollback, image updates
 - 현재 ArgoCD Image Updater는 ArgoCD application의 rollback 상태를 고려하지 않고, rollback되고 있는 애플리케이션이라 할지라도 새로운 버전의 이미지를 적용하려 합니다.

4. 동작 방식 - 업데이트 방식

- [공식 문서](#)
- 여기서는 ArgoCD Image Updater가 ArgoCD에게 새로운 이미지가 있음을 알리는 방식들에 대해 다룹니다. 아래의 두 가지 방식이 있습니다.
 - `argocd write-back method`: Kubernetes API 혹은 ArgoCD API를 활용해 ArgoCD의 Application 리소스를 직접 수정합니다.
 - `git write-back method`: Application에 연결된 git repository에 업데이트할 이미지의 정보를 담은 commit을 생성합니다.
- `argocd write-back method`
 - 이 방식을 활용하면 ArgoCD Image Updater가 `argocd app set --parameter ..` 과 같은 명령어를 실행해 ArgoCD가 변경된 파라미터로 애플리케이션을 배포합니다.
 - 이 방식은 영속적이지 않습니다. 즉, 만약 클러스터에서 Application 리소스를 제거하고 다시 생성하면, Image Updater에 의해 만들어진 변경 사항들은 모두 초기화됩니다. Application 리소스를 Git으로 관리하더라도 만약 Git 버전이 클러스터에 sync된다면 초기화됩니다.
- `git write-back method`

- 이 방식은 변경된 파라미터(이미지 갱신 내용)를 git을 활용해 영구적으로 저장합니다.
- 기본 설정으로 ArgoCD Image Updater는 `.argocd-source-<appName>.yaml` 파일에 변경 사항들을 기록합니다. 아래 annotation을 적용하면 git write-back method를 사용합니다.

```
argocd-image-updater.argoproj.io/write-back-method: git
```

- ArgoCD Image Updater가 git에 변경 사항을 기록하는 과정은 아래와 같습니다.
 1. Application의 `spec.source.repoURL` 에 지정된 remote repository를 fetch합니다.
 2. Target branch에 checkout합니다.
 3. `.argocd-source-<appName>.yaml` 을 생성하거나 수정합니다.
 4. 변경된 파일을 commit하고, remote repository에 push 합니다.
- 위 workflow에서 중요한 사항들은 아래와 같습니다.
 - ArgoCD에 설정된 credential들은 재사용됩니다.
 - Application이 tracking하도록 설정된 branch에 commit을 합니다.
- Git commit을 push하거나 remote repository를 fetch하기 위한 credential을 설정하는 방법은 [여기](#)에 있습니다.

5. 동작 방식 - 업데이트 전략

- [공식 문서](#)
- 업데이트 전략은 ArgoCD Image Updater가 특정 이미지의 새로운 버전이 나왔음을 판단하는 방식을 지정합니다. 아래의 전략들이 지원됩니다.
 - `semver` : Semantic versioning 규칙에 따라 가장 최신 버전의 이미지로 갱신합니다.
 - `latest` : Registry에서 발견할 수 있는 가장 최근에 build된 이미지로 갱신합니다.
 - `digest` : 주어진 버전(tag)의 SHA digest를 활용해 가장 최신 버전의 이미지로 갱신합니다.
 - `name` : Tag를 알파벳 순서로 정렬해 가장 먼저 오는 이미지로 갱신합니다.
- Mutable vs Immutable tags
 - `digest` 전략을 제외한 모든 전략은 tag가 immutable(불변) 하다고 가정합니다. 즉, 모든 새로운 이미지들이 새로운 tag로 push될 것이라 가정합니다.
- `semver` - Semantic version에 따른 갱신 (기본 업데이트 전략)
 - 이 전략은 [semantic versioning 스키마](#)의 형식에 따르는 tag를 위해 사용됩니다. 태그들은 semantic versioning 스키마에 맞는 `x.y.z` 의 형식을 가져야 하며, 선택적으로 `v1.0.0` 처럼 앞에 `v`가 붙는 것도 허용합니다. 또한 `-rc1` 처럼 release candidate tag도 지원됩니다. (명시적으로 설정 필요)

```
argocd-image-updater.argoproj.io/image-list: some/image[:<version_constraint>]
argocd-image-updater.argoproj.io/<image>.update-strategy: semver
```

- 만약 `1.2.x` 태그만 ArgoCD Image Updater로 추적하고 싶다면, 아래와 같이 할 수 있습니다.

```
argocd-image-updater.argoproj.io/image-list: some/image:1.2.x
```

- Release candidate까지 ArgoCD Image Updater가 추적하도록 하려면, 아래와 같이 `-0` 을 붙여줍니다.

```
argocd-image-updater.argoproj.io/image-list: some/image:1.2.x-0
```

- `latest` - 가장 최근에 build된 이미지로 갱신
 - Docker Hub의 image pull limit 때문에, 만약 공개된 repository의 이미지라면 `latest` tag를 사용하는 방식은 권장되지 않습니다.
- `digest`, `name` 의 설정 방법은 공식 문서를 참고해주세요..

1. ArgoCD 설정하기

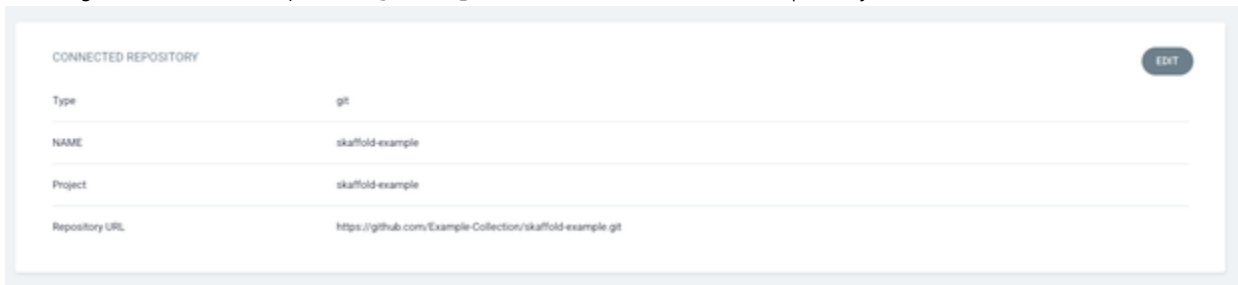
- 이번 PoC는 Helm manifest들이 포함된 [argocd-image-updater-example 레포지토리](#)로 진행합니다.

1. ArgoCD 설치

- 아래 명령어로 ArgoCD를 설치합니다.

```
kubectl create ns argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

- 다음으로 ArgoCD 페이지에 접속해, Settings → Repositories 에 들어가 아래처럼 repository를 추가합니다.



CONNECTED REPOSITORY		EDIT
Type	git	
NAME	skafold-example	
Project	skafold-example	
Repository URL	https://github.com/Example-Collection/skafold-example.git	

2. Application 배포

- 아래 명령어를 통해 demo 애플리케이션을 Kubernetes에 배포합니다.

```
git clone https://github.com/Example-Collection/argocd-image-updater-example.git
cd argocd-image-updater-example
kubectl create ns demo
helm install demo ./helm -f ./helm-internal-values.yaml
```

- 배포가 완료되면, 아래와 같이 하나의 pod가 뜹니다.

```
$ kubectl get po -n demo
NAME                                READY   STATUS    RESTARTS   AGE
demo-6cf769df7d-zszjk             1/1     Running   0           9m30s
```

3. ArgoCD 설정

- 이제 ArgoCD를 위한 2개의 Custom Resource를 생성할 것인데, 첫 번째는 project 입니다. 참고로 이 파일은 clone한 git repository에 이미 정의되어 있습니다.

```
# argocd/appproject.yaml
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  namespace: argocd
  name: demo
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  description: Example project for testing argocd-image-updater
  sourceRepos:
    - '*'
  destinations:
    - namespace: demo
      server: https://kubernetes.default.svc
  clusterResourceWhitelist:
    - group: ''
      kind: Namespace
```

- 다음으로는 Application 입니다. ArgoCD Application은 project에 소속되며, 이번에 생성할 application은 위에서 만든 demo project에 소속시킬 것입니다.

```
# argocd/application.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: demo
  namespace: argocd
spec:
  project: demo
  source:
    repoURL: https://github.com/Example-Collection/argocd-image-updater-example.git
    targetRevision: main
    path: helm/
    helm:
      valueFiles:
        - ./internal-values.yaml
  destination:
    server: https://kubernetes.default.svc
    namespace: demo
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

- 이제 위 2개를 모두 apply 합니다.

```
kubectl apply -n argocd -f argocd/appproject.yaml
kubectl apply -n argocd -f argocd/application.yaml
```

- 이후 ArgoCD 페이지에 접속하면, 아래와 같이 배포된 애플리케이션을 ArgoCD가 잘 확인하는 것을 볼 수 있습니다.

2. ArgoCD Image Updater 설치하기

- 공식 문서
- ArgoCD Image Updater를 설치하는 방법은 ArgoCD가 설치된 cluster에 설치할 수 있는지의 여부에 따라 달라집니다.
 - 가능한 경우: 1번 방식 사용
 - 불가능한 경우: 2번 방식 사용
- 여기서는 ArgoCD Image Updater를 ArgoCD가 설치된 cluster에 설치할 수 있으므로 1번 방식을 사용하겠습니다.
- 아래 명령어로 ArgoCD Image Updater가 사용하는 manifest들을 생성합니다.
아래 명령어는 ArgoCD가 설치되어 있는 namespace, 즉 argocd namespace에 관련 리소스들을 생성합니다.

```
kubectl apply -n argocd -f https://raw.githubusercontent.com
/argoproj-labs/argocd-image-updater/stable/manifests/install.yaml
```

⚠ ArgoCD Image Updater 공식 문서에 따르면, ArgoCD Image Update를 HA 구성(replica를 2개 이상으로 늘리는 등)하는 것은 의도하지 않은 작업을 야기할 수 있으므로 권장되지 않습니다.

3. ArgoCD Image Updater 관련 설정하기

- 이제 ArgoCD의 리소스인 Application 중 어떤 이미지를 ArgoCD Image Updater가 관리할지를 지정해야 합니다. 여기서는 robyra98 /argo-image-updater-poc 이미지를 semantic versioning으로 업데이트하도록 해보겠습니다. 먼저 관련 Application에 annotation을 추가합니다.

```
# argocd/application.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: demo
  namespace: argocd
  annotations:
    argocd-image-updater.argoproj.io/image-list: aiup=robbyra98
    /argo-image-updater-poc
    argocd-image-updater.argoproj.io/aiup.update-strategy: semver
spec:
  project: demo
  source:
    repoURL: https://github.com/Example-Collection/argocd-image-
    updater-example.git
    targetRevision: main
    path: helm/
    helm:
      valueFiles:
        - ./internal-values.yaml
  destination:
    server: https://kubernetes.default.svc
    namespace: demo
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

1. argocd write-back method 업데이트 방식

- annotation에 아래 내용을 추가합니다.

```
# argocd/application.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: demo
  namespace: argocd
  annotations:
    argocd-image-updater.argoproj.io/image-list: aiup=robbyra98
    /argo-image-updater-poc
    argocd-image-updater.argoproj.io/aiup.update-strategy: semver
    argocd-image-updater.argoproj.io/write-back-method: argocd
spec:
  #..
```

- 위 변경 사항을 apply 합니다.

```
kubectl apply -n argocd -f argocd/application.yaml
```

- 이제 새로운 이미지를 배포해보겠습니다. main.go 파일의 응답을 아래처럼 바꿉니다.

```
package main

import "net/http"


func main() {
    http.HandleFunc("/health", func(w http.ResponseWriter, r
    *http.Request) {
        w.Header().Set("Content-Type", "application/json")
        _, _ = w.Write([]byte(`{"message": "server is
    running, version 0.0.1"}`))
    })
    _ = http.ListenAndServe(":8080", nil)
}
```

- 그리고 아래 명령어로 새로운 이미지를 빌드하고, push 합니다.

```
docker build --platform=linux/amd64 -t robyra98/argo-image-updater-
poc:0.0.1 .
docker push robyra98/argo-image-updater-poc:0.0.1
```

- 이제 argocd namespace에 있는 argo-image-updater-`<randomString>` pod의 로그를 보면, 아래와 같이 새로운 이미지를 발견해 적용 했다는 로그가 나옵니다.



```
time="2023-02-20T07:25:20Z" level=info msg="Starting image update
cycle, considering 1 annotated application(s) for update"
time="2023-02-20T07:25:22Z" level=info msg="Setting new image to
robbyra98/argo-image-updater-poc:0.0.1" alias=aiup application=demo
image_name=robbyra98/argo-image-updater-poc image_tag=0.0.0
registry=
time="2023-02-20T07:25:22Z" level=info msg="Successfully updated
image 'robbyra98/argo-image-updater-poc:0.0.0' to 'robbyra98/argo-
image-updater-poc:0.0.1', but pending spec update (dry run=false)"
alias=aiup application=demo image_name=robbyra98/argo-image-updater-
poc image_tag=0.0.0 registry=
time="2023-02-20T07:25:22Z" level=info msg="Committing 1 parameter
update(s) for application demo" application=demo
time="2023-02-20T07:25:22Z" level=info msg="Successfully updated
the live application spec" application=demo
time="2023-02-20T07:25:22Z" level=info msg="Processing results:
applications=1 images_considered=1 images_skipped=0
images_updated=1 errors=0"
```

 Helm Chart 사용 시 image 관련 value 중 image 이름, 태그는 꼭 아래의 형식을 따라야 합니다.

- image 이름: `image.name`
- image tag: `image.tag`

이 사항을 따르지 않으면 ArgoCD Image Updater가 새로운 이미지가 배포되었음을 인지하긴 하지만, ArgoCD를 통해 새로운 이미지를 배포하지 못합니다.

- 이후에 ArgoCD에 의해 새로운 이미지를 가진 pod가 배포됩니다.
- ArgoCD Image Updater가 새로운 버전의 이미지를 가져왔음은 ArgoCD Web UI에서 Application 선택 후 PARAMETERS 부분에서 확인할 수 있습니다. 아래 사진과 같이 `image.name` 과 `image.tag` 가 새로운 버전인 0.0.1로 변경되어 있음을 확인할 수 있습니다.

SUMMARY	PARAMETERS	MANIFEST	EVENTS
HELM			
VALUES FILES	./internal-values.yaml		
VALUES			
PARAMETERS			
image.name	 robbyra98/argo-image-updater-poc		
image.tag	 0.0.1		
container.name	demo		

2. git write-back method 업데이트 방식

- 이 방식은 git commit을 통해 Application의 이미지 변경 사항을 영구적으로 기록하는 방식입니다. 우선 git commit을 위한 PAT를 발급받은 후, 아래의 secret을 생성합니다.

```
# argocd/git-cred.yaml
apiVersion: v1
kind: Secret
metadata:
  namespace: argocd
  name: git-cred
data:
  username: "base64 encoding github username"
  password: "base64 encoding PAT"
```

- 그 다음, 아래 annotation들을 추가합니다.

```
# argocd/application.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: demo
  namespace: argocd
  annotations:
    argocd-image-updater.argoproj.io/image-list: aiup=robbyra98
    /argo-image-updater-poc
    argocd-image-updater.argoproj.io/aiup.update-strategy: semver
    argocd-image-updater.argoproj.io/write-back-method: git:secret:
    argocd/git-cred
spec:
  #..
```

- write-back-method는 git:secret:<namespace>/<secret name> 형식을 따릅니다.
- 위 변경 사항을 apply 합니다.

```
kubectl apply -n argocd -f argocd/application.yaml
```

- 이제 새로운 이미지를 배포해보겠습니다. main.go 파일의 응답을 아래처럼 바꿉니다.

```
package main

import "net/http"

func main() {
    http.HandleFunc("/health", func(w http.ResponseWriter, r
    *http.Request) {
        w.Header().Set("Content-Type", "application/json")
        _, _ = w.Write([]byte(`{"message": "server is
    running, version 0.0.1"}`))
    })
    _ = http.ListenAndServe(":8080", nil)
}
```

- 그리고 아래 명령어로 새로운 이미지를 빌드하고, push 합니다.

```
docker build --platform=linux/amd64 -t robyra98/argo-image-updater-
poc:0.0.1 .
docker push robyra98/argo-image-updater-poc:0.0.1
```

- 이제 argocd namespace에 있는 argo-image-updater-<randomString> pod의 로그를 보면, 아래와 같이 새로운 이미지를 발견해 적용했다는 로그가 나옵니다.

```
time="2023-02-20T07:41:34Z" level=info msg="Starting image update
cycle, considering 1 annotated application(s) for update"
time="2023-02-20T07:41:36Z" level=info msg="Setting new image to
robbyra98/argo-image-updater-poc:0.0.1" alias=aiup application=demo
image_name=robbyra98/argo-image-updater-poc image_tag=0.0.0
registry=
time="2023-02-20T07:41:36Z" level=info msg="Successfully updated
image 'robbyra98/argo-image-updater-poc:0.0.0' to 'robbyra98/argo-
image-updater-poc:0.0.1', but pending spec update (dry run=false)"
alias=aiup application=demo image_name=robbyra98/argo-image-updater-
poc image_tag=0.0.0 registry=
time="2023-02-20T07:41:36Z" level=info msg="Committing 1 parameter
update(s) for application demo" application=demo
time="2023-02-20T07:41:36Z" level=info msg="Initializing
https://github.com/Example-Collection/argocd-image-updater-example.
git to /tmp/git-demol488347215"
time="2023-02-20T07:41:36Z" level=info msg="rm -rf /tmp/git-
demol488347215" dir= execID=5bda4
time="2023-02-20T07:41:36Z" level=info msg=Trace args="[rm -rf /tmp
/git-demol488347215]" dir= operation_name="exec rm" time_ms=1.
6627079999999999
time="2023-02-20T07:41:36Z" level=info msg="git fetch origin --tags
--force" dir=/tmp/git-demol488347215 execID=fccf4
time="2023-02-20T07:41:37Z" level=info msg=Trace args="[git fetch
origin --tags --force]" dir=/tmp/git-demol488347215 operation_name="
exec git" time_ms=858.283004
time="2023-02-20T07:41:37Z" level=info msg="git config user.name
argocd-image-updater" dir=/tmp/git-demol488347215 execID=79ee3
time="2023-02-20T07:41:37Z" level=info msg=Trace args="[git config
user.name argocd-image-updater]" dir=/tmp/git-demol488347215
operation_name="exec git" time_ms=2.039078
time="2023-02-20T07:41:37Z" level=info msg="git config user.email
noreply@argoproj.io" dir=/tmp/git-demol488347215 execID=3d800
time="2023-02-20T07:41:37Z" level=info msg=Trace args="[git config
user.email noreply@argoproj.io]" dir=/tmp/git-demol488347215
operation_name="exec git" time_ms=3.247265
time="2023-02-20T07:41:37Z" level=info msg="git checkout --force
main" dir=/tmp/git-demol488347215 execID=7a9c2
time="2023-02-20T07:41:37Z" level=info msg=Trace args="[git
checkout --force main]" dir=/tmp/git-demol488347215 operation_name="
exec git" time_ms=6.0346439999999999
time="2023-02-20T07:41:37Z" level=info msg="git clean -fdx" dir=/tmp
/git-demol488347215 execID=12c16
time="2023-02-20T07:41:37Z" level=info msg=Trace args="[git clean -
fdx]" dir=/tmp/git-demol488347215 operation_name="exec git"
time_ms=2.060007
time="2023-02-20T07:41:37Z" level=info msg="git add /tmp/git-
```

```

demo1488347215/helm/.argocd-source-demo.yaml" dir=/tmp/git-
demo1488347215 execID=cc115
time="2023-02-20T07:41:37Z" level=info msg=Trace args="[git add /tmp
/git-demo1488347215/helm/.argocd-source-demo.yaml]" dir=/tmp/git-
demo1488347215 operation_name="exec git" time_ms=3.496874
time="2023-02-20T07:41:37Z" level=info msg="git commit -a -F /tmp
/image-updater-commit-msg1279233135" dir=/tmp/git-demo1488347215
execID=25b22
time="2023-02-20T07:41:37Z" level=info msg=Trace args="[git commit -
a -F /tmp/image-updater-commit-msg1279233135]" dir=/tmp/git-
demo1488347215 operation_name="exec git" time_ms=8.491329
time="2023-02-20T07:41:37Z" level=info msg="git push origin main"
dir=/tmp/git-demo1488347215 execID=7f3d6
time="2023-02-20T07:41:38Z" level=info msg="Successfully updated
the live application spec" application=demo
time="2023-02-20T07:41:38Z" level=info msg=Trace args="[git push
origin main]" dir=/tmp/git-demo1488347215 operation_name="exec git"
time_ms=1181.422111
time="2023-02-20T07:41:38Z" level=info msg="Processing results:
applications=1 images_considered=1 images_skipped=0
images_updated=1 errors=0"

```

- 그리고 Github Repository에 아래와 같이 새로운 commit이 올라옵니다.

The screenshot shows a GitHub commit page for the repository 'argocd-image-updater'. The commit is titled 'build: automatic update of demo' and updates the image 'robbyra98/argo-image-updater-poc' from tag '0.0.0' to '0.0.1'. The commit is made by 'argocd-image-updater' 2 minutes ago. The diff view shows changes to 'helm/.argocd-source-demo.yaml', with 8 additions and 0 deletions. The diff content is as follows:

```

@@ -0,0 +1,8 @@
1 helm:
2   parameters:
3   - name: image.name
4     value: robyra98/argo-image-updater-poc
5   - name: image.tag
6     value: 0.0.1
7   - name: image.tag
8     value: 0.0.1

```

Below the diff, there are 0 comments on the commit. At the bottom, there is a 'Write' tab and a 'Preview' tab, with a text area for leaving a comment and a 'Comment on this commit' button.

- 그리고 ArgoCD가 Github Repository를 주기적으로 polling할 때, 새로운 변경 사항이 있음을 감지하고 새로운 이미지를 배포합니다.
- 이미 레포지토리에 '.argocd-source-<appName>.yaml' 파일이 존재할 경우에는 아래와 같은 commit이 생깁니다.

The screenshot shows a GitHub commit page for the repository 'argocd-image-updater'. The commit is titled 'build: automatic update of demo' and updates the image 'robbyra98/argo-image-updater-poc' from tag '0.0.1' to '0.0.2'. The commit is made by 'argocd-image-updater' committed now. The diff view shows changes to 'helm/.argocd-source-demo.yaml', with 1 parent '28ef3d7' and commit '14ee108'. The diff content is as follows:

```

@@ -0,0 +1,8 @@
1 helm:
2   parameters:
3   - name: image.name
4     value: robyra98/argo-image-updater-poc
5   - name: image.tag
6     value: 0.0.1
7   - name: image.tag
8     value: 0.0.2

```

Below the diff, there are 0 comments on the commit. At the bottom, there is a 'Write' tab and a 'Preview' tab, with a text area for leaving a comment and a 'Comment on this commit' button.

Showing 1 changed file with 1 addition and 1 deletion.

No Whitespace

Split

Unified

```
helm/.argocd-source-demo.yaml
@@ -4,5 +4,5 @@ helm:
+
4 4     ---value: robbyra98/argo-image-updater-poc
5 5     ---forcestring: true
6 6     -- name: image.tag
7 7     ---value: 0.0.1
+7 7     ---value: 0.0.2
8 8     ---forcestring: true
```

0 comments on commit 14ee108

Lock conversation

Write

Preview

<>

Leave a comment

Comment on this commit