# Argo CD 운영

# Argo CD를 사용한 애플리케이션 배포

- 매니페스트 작성
  - ✓ manifest/ns.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: bgd
```

# Argo CD를 사용한 애플리케이션 배포

- 매니페스트 작성
  - ✓ manifest/deployment.yaml

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: bgd
  name: bgd
  namespace: bgd
```

# Argo CD를 사용한 애플리케이션 배포

- 매니페스트 작성
  - ✓ manifest/deployment.yaml

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bgd
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: bgd
    spec:
      containers:
      - image: ggnagpae1/bgd:1.0.0
        name: bgd
        env:
        - name: COLOR
          value: "blue"
        resources: {}
---
```

# Argo CD를 사용한 애플리케이션 배포

- 매니페스트 작성
  - ✓ manifest/svc.yaml

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: bgd
  name: bgd
  namespace: bgd
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
    nodePort: 31080
  selector:
    app: bgd
  type: NodePort
---
```

# Argo CD를 사용한 애플리케이션 배포

- Git Hub에 Push

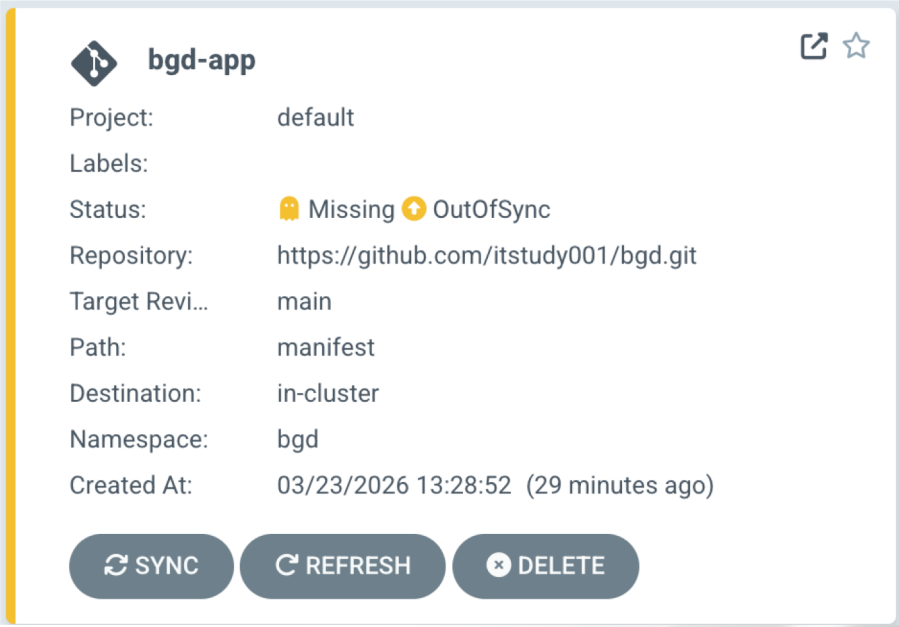
# Argo CD를 사용한 애플리케이션 배포

- 배포
  - ✓ 배포용 yaml 파일 생성: bgd-app.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: bgd-app
  namespace: argocd
spec:
  destination:
    namespace: bgd
    server: https://kubernetes.default.svc
  project: default
  source:
    repoURL: https://github.com/itstudy001/bgd.git
    path: manifest
    targetRevision: main
```

# Argo CD를 사용한 애플리케이션 배포

- 배포
  - ✓ 배포: kubectl apply -f bgd-app.yaml
  - ✓ 확인



# Argo CD를 사용한 애플리케이션 배포

- ❖ 배포
  - ✓ 확인
    - argocd app list

      NAME CLUSTER NAMESPACE PROJECT STATUS HEALTH SYNCPOLICY CONDITIONS REPO PATH TARGET

      argocd/bgd-app https://kubernetes.default.svc bgd default OutOfSync Missing Manual <none> https://github.com/itstudy001/bgd.git manifest main
    - kubectl get pods -n bgd

      No resources found in bgd namespace
      - ◆ Argo CD는 기본적으로 애플리케이션을 자동으로 동기화하지 않음
      - ◆ 생겼음을 알려줄 뿐
      - ◆ 사용자는 수동으로 동기화 작업을 게시하여 이 문제를 해결

# Argo CD를 사용한 애플리케이션 배포

❖ 배포

✓ 동기화: argocd app sync bgd-app

```
GROUP  KIND        NAMESPACE  NAME  STATUS   HEALTH       HOOK  MESSAGE
       Namespace   bgd        bgd   Running  Synced             namespace/bgd created
       Service     bgd        bgd   Synced   Healthy            service/bgd created
apps   Deployment  bgd        bgd   Synced   Progressing        deployment.apps/bgd created
       Namespace                bgd   Synced
```

✓ 확인

- kubectl get pods -n bgd

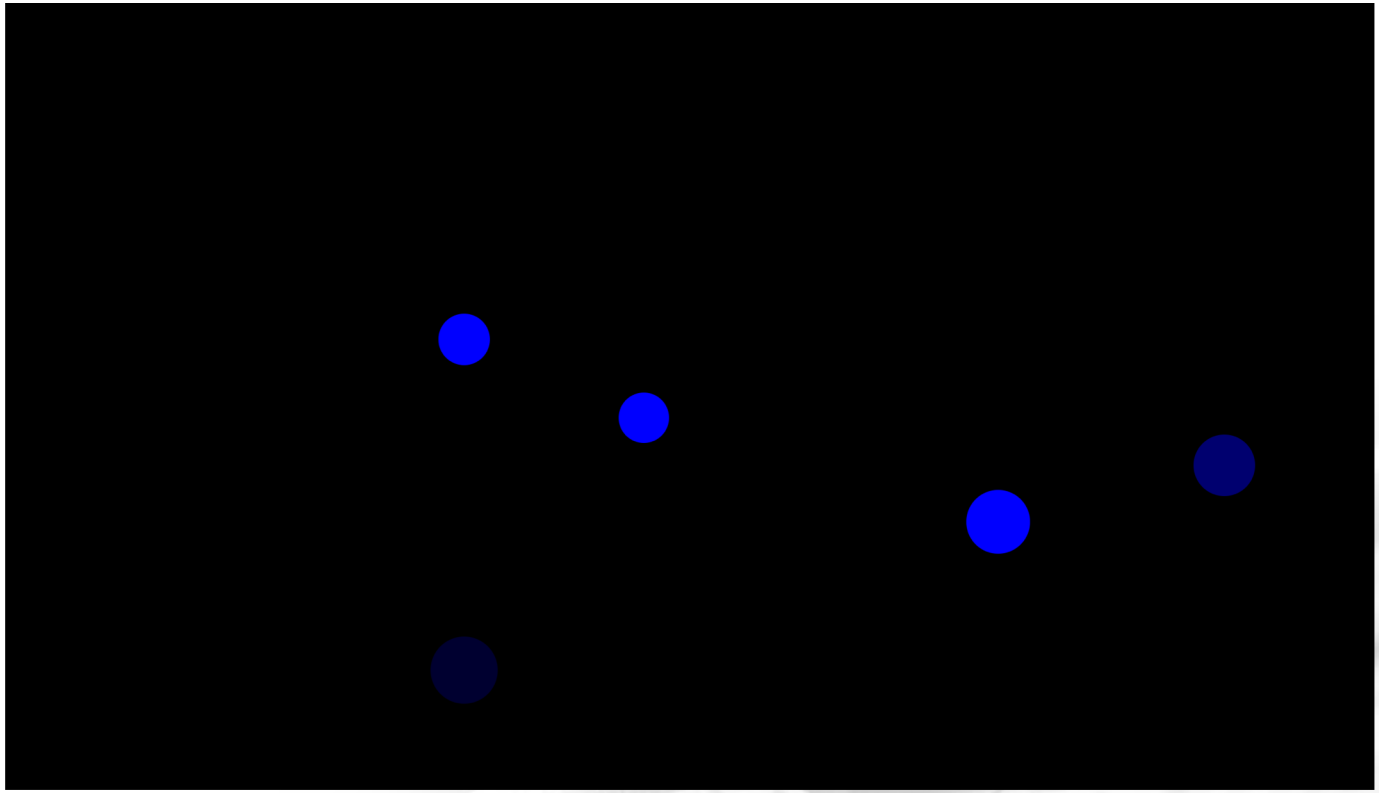
```
NAME                   READY   STATUS    RESTARTS      AGE
bgd-7984d6f558-ddczk   1/1     Running   2 (30s ago)   61s
```

- kubectl get svc -n bgd

```
NAME   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
bgd    ClusterIP   10.111.252.195  <none>        8080/TCP   130m
```

# Argo CD를 사용한 애플리케이션 배포

- 배포
  - ✓ 서비스 확인: MasterIP: 31080



# Argo CD를 사용한 애플리케이션 배포

- ❖ 변경 내용 적용
  - ✓ deployment.yaml 파일을 열어서 COLOR 환경 변수의 값을 blue에서 green으로 수정하고 push
    - 현재 상태: argocd app list

      NAME CLUSTER NAMESPACE PROJECT STATUS HEALTH SYNCPOLICY CONDITIONS REPO PATH TARGET

      argocd/bgd-app https://kubernetes.default.svc bgd default Synced Progressing Manual <none> https://github.com/itstudy001/bgd.git manifest main
    - 일정 시간이 지난 후의 상태: argocd app list

      NAME CLUSTER NAMESPACE PROJECT STATUS HEALTH SYNCPOLICY CONDITIONS REPO PATH TARGET

      argocd/bgd-app https://kubernetes.default.svc bgd default OutOfSync Progressing Manual <none> https://github.com/itstudy001/bgd.git manifest main

# Argo CD를 사용한 애플리케이션 배포

- ❖ 앱 제거 및 저장소 복원
  - ✓ 앱 제거: argocd app delete bgd-app
  - ✓ git hub 저장소를 이전으로 되돌리기

# Argo CD를 사용한 애플리케이션 배포

- ❖ 자동 동기화
  - ✓ 개요
    - Argo CD는 Git과 쿠버네티스 클러스터 매니페스트 간의 차이를 감지하면 애플리케이션을 자동으로 동기화할 수 있음
    - 자동 동기화의 장점은 Argo CD API에 로그인 할 필요가 없다는 것
    - 로그인을 통해 동기화를 처리하려면 다양한 보안 문제를 해결해야 함(시크릿, 네트워크 보안 등)
    - 자동 동기화를 채택하면 argocd CLI를 사용할 필요가 없음
    - 추적 중인 Git 저장소에 매니페스트 변경 사항이 푸시되면 자동으로 클러스터에 적용됨

# Argo CD를 사용한 애플리케이션 배포

- ❖ 자동 동기화
  - ✓ 적용
    - bgd-app.yaml 파일을 수정

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: bgd-app
  namespace: argocd
spec:
  destination:
    namespace: bgd
    server: https://kubernetes.default.svc
  project: default
  source:
    repoURL: https://github.com/itstudy001/bgd.git
    path: manifest
    targetRevision: main
  syncPolicy:
    automated: {}
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ 자동 동기화
  - ✓ 적용
    - 배포: kubectl apply -f bgd-app.yaml
    - 확인: kubectl get pods -n bgd

      NAME READY STATUS RESTARTS AGE

      bgd-7984d6f558-bkp75 1/1 Running 1 (10s ago) 11s
    - deployment.yaml 파일을 열어서 COLOR 환경 변수의 값을 blue에서 green으로 수정하고 push
    - 확인: kubectl get pods -n bgd

      NAME READY STATUS RESTARTS AGE

      bgd-67ffff4b65-gb9fq 1/1 Running 4 (53s ago) 2m15s

      bgd-7984d6f558-bkp75 1/1 Running 5 (2m17s ago) 5m25s

# Argo CD를 사용한 애플리케이션 배포

- ❖ 자동 동기화
  - ✓ 적용
    - ● 서비스 확인

# Argo CD를 사용한 애플리케이션 배포

- ❖ 자동 동기화
  - ✓ 기본 보존 전략
    - ● 삭제된 리소스를 정리하는 전략
      - ◆ 기본적으로 Argo CD는 Git에서 더 이상 사용할 수 없게 된 리소스도 삭제/정리하지 않으며 단순히 OutOfSync 상태로만 변경
      - ◆ 이런 리소스를 삭제하는 방법
        - ▪ argocd app sync <앱 이름> --prune
        - ▪ yaml 파일에 설정

```
syncPolicy:
  automated:
    prune: true
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ 자동 동기화
  - ✓ 기본 보존 전략
    - ● 자가 치유
      - ◆ 애플리케이션이 자동으로 업데이트되는 방식에 영향을 미치는 또 다른 중요한 개념은 자가 치유(self-healing)
      - ◆ Argo CD는 클러스터에서 수동으로 발생한 변경을 교정하지 않음
      - ◆ Argo CD는 클러스터에서 직접 kubectl patch 명령을 실행하여 애플리케이션의 구성 파라미터를 변경할 수 있도록 허용
      - ◆ 애플리케이션이 표시하는 원의 색상은 환경 변수(COLOR)를 통해 지정하는데 kubectl patch 명령어를 사용하여 COLOR 환경 변수의 값을 green으로 변경

        ```
        kubectl -n bgd patch deploy/bgd --type='json' -p='[{"op":"replace",
        "path":"/spec/template/spec/containers/0/env/0/value", "value":"green"}]'
        ```

      - ◆ 동기화 상태 확인: argocd app list

        ```
        NAME              CLUSTER                         NAMESPACE  PROJECT  STATUS
        HEALTH       SYNCPOLICY  CONDITIONS  REPO
        PATH       TARGET

        argocd/bgd-app  https://kubernetes.default.svc  bgd        default
        OutOfSync  Progressing  Auto        <none>
        https://github.com/itstudy001/bgd.git  manifest  main
        ```

# Argo CD를 사용한 애플리케이션 배포

- ❖ 자동 동기화
  - ✓ 기본 보존 전략
    - ● 자가 치유
      - ◆ selfHeal 속성의 기본값은 false이므로 Argo CD는 이 차이를 교정하기 위해 롤백을 시도하지 않음
      - ◆ 기존 애플리케이션 제거: argocd app delete bgd-app

# Argo CD를 사용한 애플리케이션 배포

- ❖ 자동 동기화
  - ✓ 기본 보존 전략
    - ● 자가 치유
      - ◆ self.Healing 속성을 활성화하도록 배포 파일 수정: nano bgd-app.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: bgd-app
  namespace: argocd
spec:
  destination:
    namespace: bgd
    server: https://kubernetes.default.svc
  project: default
  source:
    repoURL: https://github.com/itstudy001/bgd.git
    path: manifest
    targetRevision: main
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ 자동 동기화
  - ✓ 기본 보존 전략
    - ● 자가 치유
      - ◆ 배포: kubectl apply -f bgd-app.yaml
      - ◆ 속성 변경: kubectl -n bgd patch deploy/bgd --type='json' -p='[{"op":"replace", "path":"/spec/template/spec/containers/0/env/0/value", "value":"green"}]'
      - ◆ 현재 상태 확인: argocd app list

        NAME CLUSTER NAMESPACE PROJECT STATUS HEALTH SYNCPOLICY CONDITIONS REPO PATH TARGET

        argocd/bgd-app https://kubernetes.default.svc bgd default Synced Progressing Auto-Prune <none> https://github.com/itstudy001/bgd.git manifest main
      - ◆ Argo CD는 patch 명령으로 인해 발생한 차이를 수정하여 애플리케이션을 Git 저장소에 정의된 올바른 상태로 동기화

# Argo CD를 사용한 애플리케이션 배포

- ❖ Kustomize 연동
  - ✓ Agro CD가 Kustomize 프로젝트로 인식하게 하는 파일
    - kustomization.yaml
    - kustomization.yml
    - Kustomization

# Argo CD를 사용한 애플리케이션 배포

- ❖ Kustomize 연동
  - ✓ 실습
    - 기존 프로젝트에 bgdk 디렉토리 생성
    - bgdk 디레고리에 base 디렉토리 생성
    - base 디렉토리에 이전 프로젝트의 deployment.yaml 파일과 svc.yaml 파일 복사
    - base 디렉토리에 kustomazation.yaml 파일 생성

```
namespace: bgdk
resources:
- svc.yaml
- deployment.yaml
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Kustomize 연동
  - ✓ 실습
    - bgdk 디렉토리에 bgdk 디렉토리 생성
    - bgdk 디렉토리에 bgdk-ns.yaml 파일 생성

```
apiVersion: v1
kind: Namespace
metadata:
  name: bgdk
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Kustomize 연동
  - ✓ 실습
    - bgdk 디렉토리에 kustomazation.yaml 파일 생성

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: bgdk

resources:
  - ../base
  - bgdk-ns.yaml

patchesJson6902:
  - target:
      group: apps
      version: v1
      kind: Deployment
      name: bgd
    patch: |-
      - op: replace
        path: /spec/template/spec/containers/0/env/0/value
        value: yellow
```

    - Git Hub에 push

# Argo CD를 사용한 애플리케이션 배포

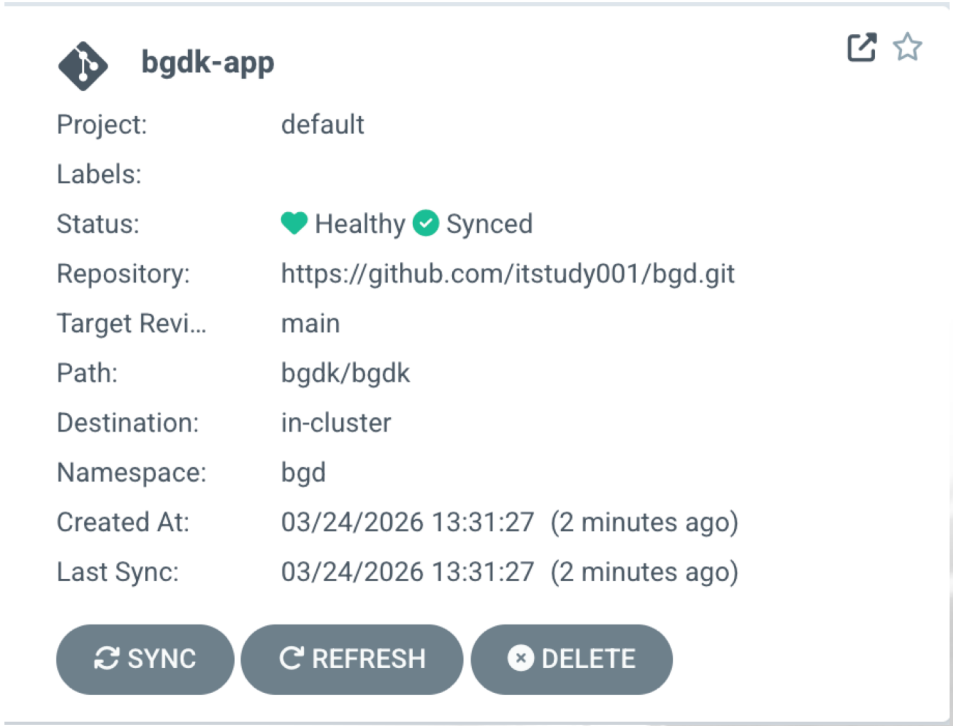
- ❖ Kustomize 연동
  - ✓ 실습
    - 현재 위치에 bgdk-app.yaml 파일 생성

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: bgdk-app
  namespace: argocd
spec:
  destination:
    namespace: bgd
    server: https://kubernetes.default.svc
  project: default
  source:
    repoURL: https://github.com/itstudy001/bgd.git
    path: bgdk/bgdk
    targetRevision: main
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

    - 배포: kubectl apply -f bgdk-app.yaml

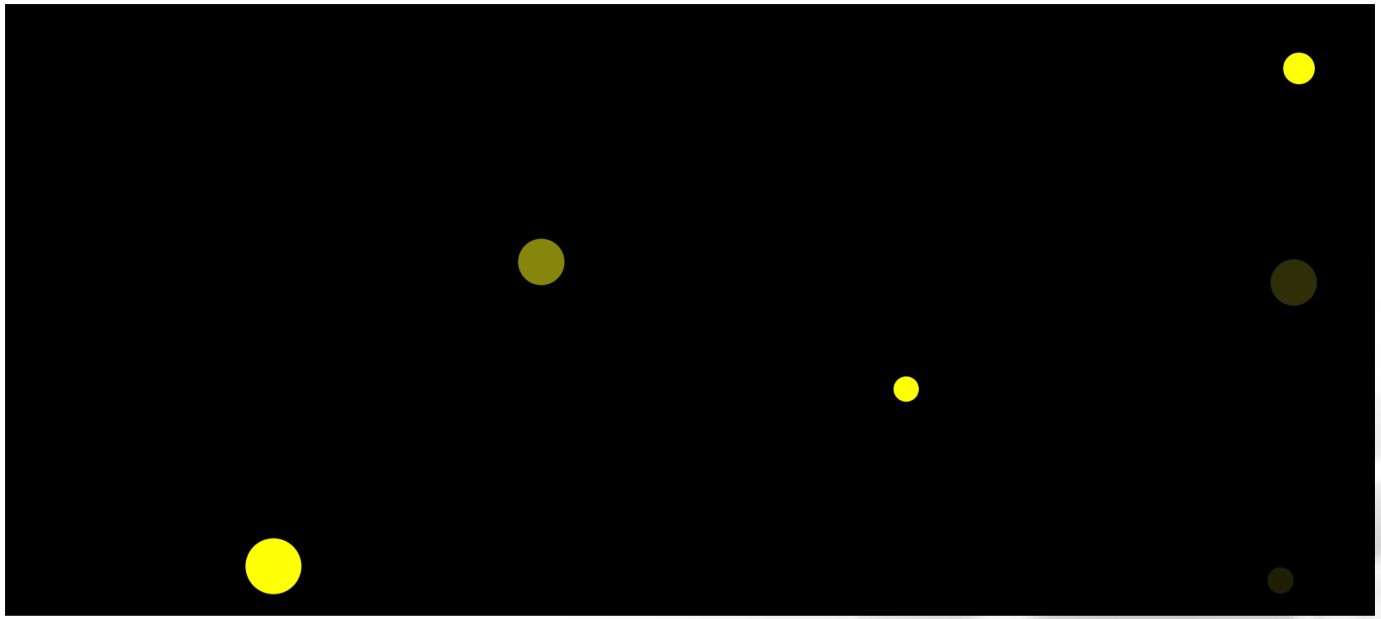
# Argo CD를 사용한 애플리케이션 배포

- ❖ Kustomize 연동
  - ✓ 실습
    - ● 확인 – argocd UI



# Argo CD를 사용한 애플리케이션 배포

- ❖ Kustomize 연동
  - ✓ 실습
    - ● 확인 - 서비스



# Argo CD를 사용한 애플리케이션 배포

- ❖ Kustomize 연동
  - ✓ Argo CD가 사용하는 기본 알고리즘을 재정의
    - 사용할 도구를 명시적으로 지정할 수 있음
    - kustomization.yaml 파일을 찾을 때 일반 디렉토리 전략(plain directory strategy)을 사용하려면 다음과 같이 설정

```
source:
  directory:
    recurse: true
```

    - 가능한 전략으로는 directory, chart, helm, kustomize, path, plugin 등이 있음

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ 배포 디렉토리에 helm 프로젝트가 있음을 감지하면(Chart.yaml) 클러스터에 차트를 자동으로 배포하도록 Argo CD를 구성할 수 있음
  - ✓ Helm 프로젝트 작성
    - ● Helm 설치
      - ◆ Ubuntu
        - ▪ curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null
        - ▪ sudo apt-get install apt-transport-https --yes
        - ▪ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
        - ▪ sudo apt-get update
        - ▪ sudo apt-get install helm
      - ◆ mac: brew install helm
      - ◆ windows: winget install Helm.Helm

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ Helm 프로젝트 작성
    - helm 프로젝트 생성: helm create bgdh
    - templates 디렉토리에서 hpa.yaml, httproute.yaml, ingress.yaml 파일을 삭제
    - templates/Notes.txt 파일 수정

1. Get the application URL by running these commands:

```
{{- if contains "ClusterIP" .Values.service.type }}
  export POD_NAME=$(kubectl get pods --namespace {{ .Release.Namespace }} -l "app.kubernetes.io/name={{ include "bgdh.name" . }},app.kubernetes.io/instance={{ .Release.Name }}" -o jsonpath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace {{ .Release.Namespace }} $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace {{ .Release.Namespace }} port-forward $POD_NAME 8080:$CONTAINER_PORT
{{- end }}
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ Helm 프로젝트 작성
    - templates/deployment.yaml 파일 수정

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "bgdh.fullname" . }}
  labels:
    {{- include "bgdh.labels" . | nindent 4 }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      {{- include "bgdh.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      {{- with .Values.podAnnotations }}
      annotations:
        {{- toYaml . | nindent 8 }}
      {{- end }}
      labels:
        {{- include "bgdh.selectorLabels" . | nindent 8 }}
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ Helm 프로젝트 작성
    - templates/deployment.yaml 파일 수정

```
spec:
  {{- with .Values.imagePullSecrets }}
  imagePullSecrets:
    {{- toYaml . | nindent 8 }}
  {{- end }}
  serviceAccountName: {{ include "bgdh.serviceAccountName" . }}
  securityContext:
    {{- toYaml .Values.podSecurityContext | nindent 8 }}
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ Helm 프로젝트 작성
    - templates/deployment.yaml 파일 수정

```
containers:
  - name: {{ .Chart.Name }}
    securityContext:
      {{- toYaml .Values.securityContext | nindent 12 }}
    image: "{{ .Values.image.repository }}:{{ .Values.image.tag |
default .Chart.AppVersion }}"
    imagePullPolicy: {{ .Values.image.pullPolicy }}
    ports:
      - name: http
        containerPort: 80
        protocol: TCP
    resources:
      {{- toYaml .Values.resources | nindent 12 }}
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ Helm 프로젝트 작성
    - templates/deployment.yaml 파일 수정

```
{{- with .Values.nodeSelector }}
nodeSelector:
  {{- toYaml . | nindent 8 }}
{{- end }}
{{- with .Values.affinity }}
affinity:
  {{- toYaml . | nindent 8 }}
{{- end }}
{{- with .Values.tolerations }}
tolerations:
  {{- toYaml . | nindent 8 }}
{{- end }}
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ Helm 프로젝트 작성
    - templates/service.yaml 파일 수정

```
apiVersion: v1
kind: Service
metadata:
  name: {{ include "bgdh.fullname" . }}
  labels:
    {{- include "bgdh.labels" . | nindent 4 }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: http
      protocol: TCP
      name: http
      nodePort: 31080
  selector:
    {{- include "bgdh.selectorLabels" . | nindent 4 }}
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ Helm 프로젝트 작성
    - templates/serviceaccount.yaml 파일 수정

```
{{- if .Values.serviceAccount.create -}}
apiVersion: v1
kind: ServiceAccount
metadata:
  name: {{ include "bgdh.serviceAccountName" . }}
  labels:
    {{- include "bgdh.labels" . | nindent 4 }}
  {{- with .Values.serviceAccount.annotations }}
  annotations:
    {{- toYaml . | nindent 4 }}
  {{- end }}
{{- end }}
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ Helm 프로젝트 작성
    - templates/ns.yaml 파일 생성

```
apiVersion: v1
kind: Namespace
metadata:
  name: bgdh
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ Helm 프로젝트 작성
    - values.yaml 파일 수정

```
replicaCount: 1

image:
  repository: quay.io/rhdevelopers/bgd
  pullPolicy: IfNotPresent
  tag: 1.0.0

imagePullSecrets: []
nameOverride: ""
fullnameOverride: ""

serviceAccount:
  create: true
  annotations: {}
  name: ""
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ Helm 프로젝트 작성
    - values.yaml 파일 수정

```
podAnnotations: {}

podSecurityContext: {}

securityContext: {}

service:
  type: NodePort
  port: 80

resources: {}

nodeSelector: {}
tolerations: []
affinity: {}
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ Helm 프로젝트 작성
    - Charts.yaml 파일 수정

```
apiVersion: v2
name: bgdh
description: Deployment with Helm

type: application
version: 0.1.0
appVersion: "1.16.0"
```

    - GitHub에 Push

# Argo CD를 사용한 애플리케이션 배포

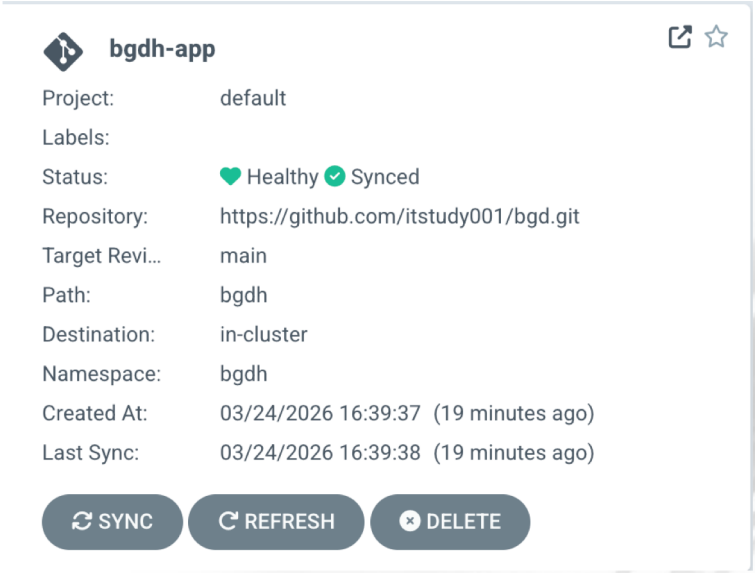
- ❖ Helm 연동
  - ✓ 배포
    - ● 배포를 위한 파일 작성: bgdh-app.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: bgdh-app
  namespace: argocd
spec:
  destination:
    namespace: bgdh
    server: https://kubernetes.default.svc
  project: default
  source:
    repoURL: https://github.com/itstudy001/bgd.git
    path: bgdh
    targetRevision: main
  syncPolicy:
    automated: {}
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ 배포
    - 배포: kubectl apply –f bgdh-app.yaml
    - 확인: argcd app list

```
NAME                CLUSTER                         NAMESPACE  PROJECT  STATUS
HEALTH   SYNCPOLICY  CONDITIONS  REPO                                PATH
TARGET
argocd/bgdh-app     https://kubernetes.default.svc  bgdh       default  Synced
Healthy  Auto        <none>      https://github.com/itstudy001/bgd.git  bgdh
main
```



# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ 빌드 환경 변수
    - ● 제공되는 변수
      - ◆ ARGOCD_APP_NAME
      - ◆ ARGOCD_APP_NAMESPACE
      - ◆ ARGOCD_APP_REVISION
      - ◆ ARGOCD_APP_SOURCE_PATH
      - ◆ ARGOCD_APP_SOURCE_REPO_URL
      - ◆ ARGOCD_APP_SOURCE_TARGET_REVISION
      - ◆ KUBE_VERSION
      - ◆ KUBE_API_VERSION

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ 빌드 환경 변수
    - ● 사용 방법

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: bgdh-app
  namespace: openshift-gitops
spec:
  destination:
  source:
    path: bgdh
    helm: -----1
      parameters: -----2
      - name: app -----3
        value: $ARGOCD_APP_NAME -----4
```

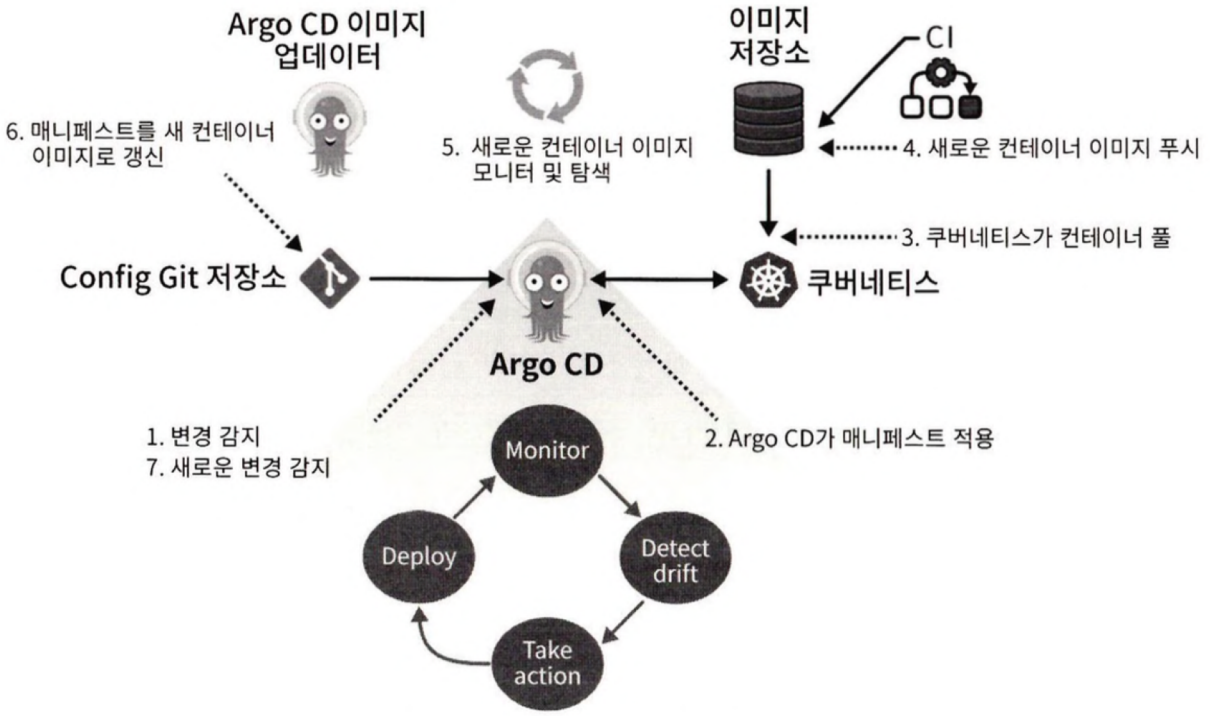
1. Helm에 인자를 전달하기 위한 절
2. 설정할 추가 매개변수로 values.yaml로 전달하는 것과 같은 방식이지만 우선순위가 높음
3. 인자 이름
4. 인자 값으로 이 경우 빌드 환경 변수에서 가져온 값

# Argo CD를 사용한 애플리케이션 배포

- ❖ Helm 연동
  - ✓ 다른 이름의 values.yaml 파일을 사용하거나 재정의하고자 하는 경우
    - argocd app set bgdh-app --values new-values.yaml
    - argocd app set bgdh-app -p service.type= LoadBalancer

# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ 개요
    - 새로운 컨테이너 버전을 모니터링하고 Argo CD Application 파일에 정의된 매니페스트를 갱신하는 쿠버네티스 컨트롤러
    - Argo CD 이미지 업데이터의 생명 주기



# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ 개요
    - Argo CD 이미지 업데이터는 커스터마이즈 또는 헬름 매니페스트에 대해서만 동작
    - 헬름의 경우 image.tag 파라미터를 사용하여 이미지 태그를 지정하도록 구성되어 있어야 함
  - ✓ 설치
    - kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj-labs/argocd-image-updater/v0.12.2/manifests/install.yaml
    - 확인: kubectl get pods -n argocd -l app.kubernetes.io/name=argocd-image-updater

      NAME READY STATUS RESTARTS AGE

      argocd-image-updater-778d955ff5-lwqz9 1/1 Running 0 59s
  - ✓ Git Hub Credential 생성: 업데이트된 매니페스트를 저장소에 push 하기 위해서
    - kubectl -n argocd create secret generic git-creds --from-literal=username=<계정이름> --from-literal=password=<Access Token>
  - ✓ 매니페스트를 만들 때 어노테이션을 추가
    - image-list: 업데이트할 이미지를 하나 이상 지정하는데 이미지가 여러 개일 때는 쉼표로 연결
    - write-back-method: 새 버전 이름을 반영할 때 사용할 방식으로 git 및 argocd 두 가지 방식이 있는데 git은 변경 사항을 Git 저장소에 커밋하고 argocd는 쿠버네티스나 Argo CD API를 사용하여 리소스를 직접 갱신

# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ 배포
    - 이전 bgdk의 내용을 이용 – bgdui 디렉토리를 생성하고 bgdk 디렉토리를 복사
    - GitHub에 push

# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ 배포
    - 배포 파일 생성 – bgdiu.yaml

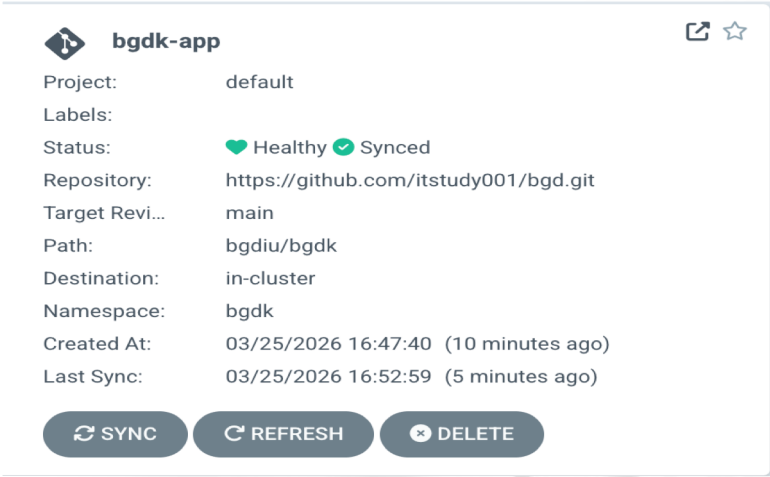
```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: bgdk-app
  namespace: argocd
  annotations:
    argocd-image-updater.argoproj.io/image-list: myalias=ggnagpae1/bgd
    argocd-image-updater.argoproj.io/write-back-method: git:secret:argocd/git-
creds
    argocd-image-updater.argoproj.io/git-branch: main
spec:
  destination:
    namespace: bgdk
    server: https://kubernetes.default.svc
  project: default
  source:
    repoURL: https://github.com/itstudy001/bgd.git
    path: bgdiu/bgdk
    targetRevision: main
  syncPolicy:
    automated: {}
```

# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ 배포
    - 배포: kubectl apply -f bgdiu.yaml
    - 확인
      - argocd app list

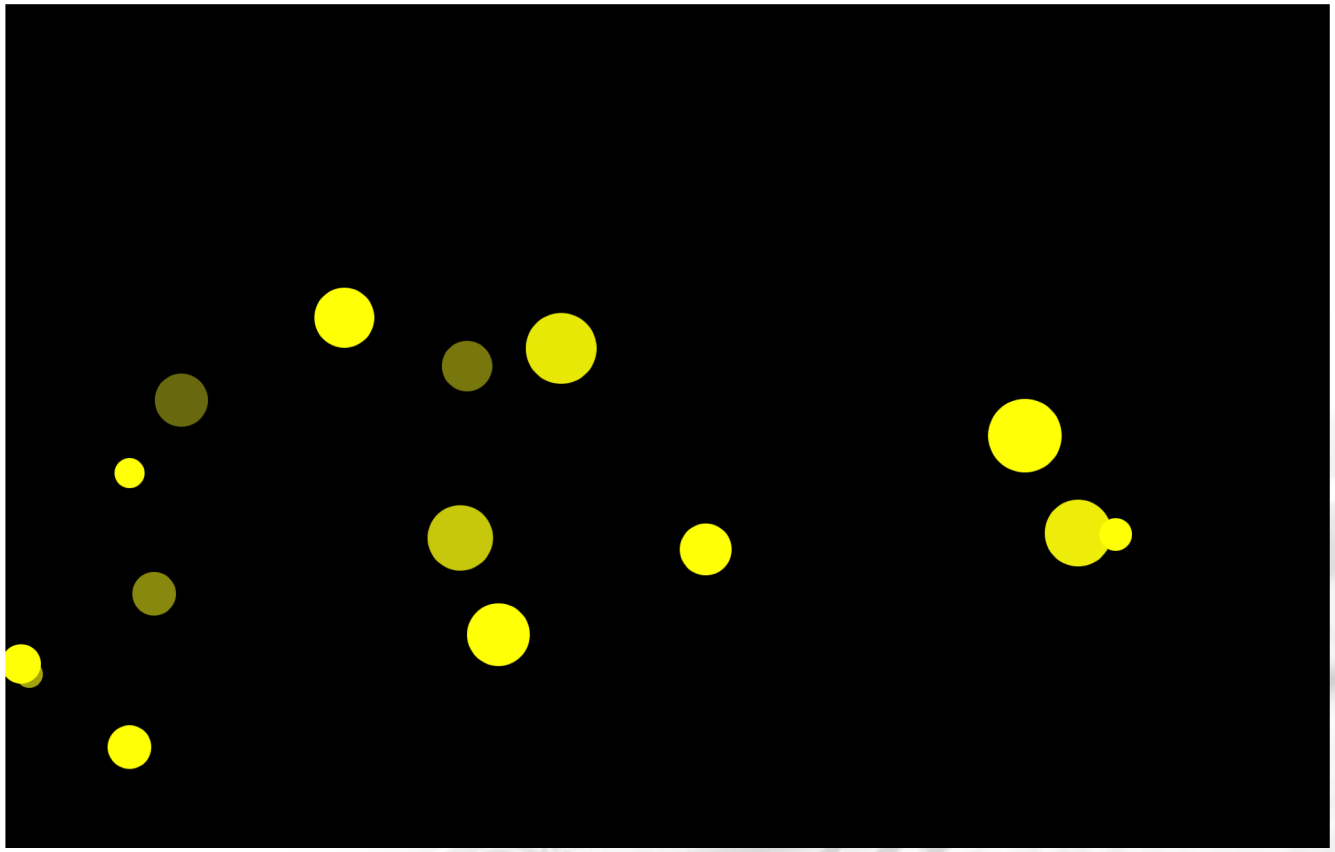
```
NAME                CLUSTER                         NAMESPACE  PROJECT  STATUS
HEALTH   SYNCPOLICY  CONDITIONS  REPO
PATH    TARGET

argocd/bgdk-app  https://kubernetes.default.svc  bgdk       default  Synced
Healthy  Auto        <none>      https://github.com/itstudy001/bgd.git
bgdiu   main
```



# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ 배포
    - 확인



# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ 이미지 업데이트
    - 이미지에 태그를 다시 붙여서 Docker Hub에 업로드
      - docker tag ggnagpae1/bgd:1.0.0 ggnagpae1/bgd:1.1.0
      - docker push ggnagpae1/bgd:1.1.0
      - 2분 정도 후 로그 확인: kubectl logs argocd-image-updater-778d955ff5-lwqz9 -f -n argocd

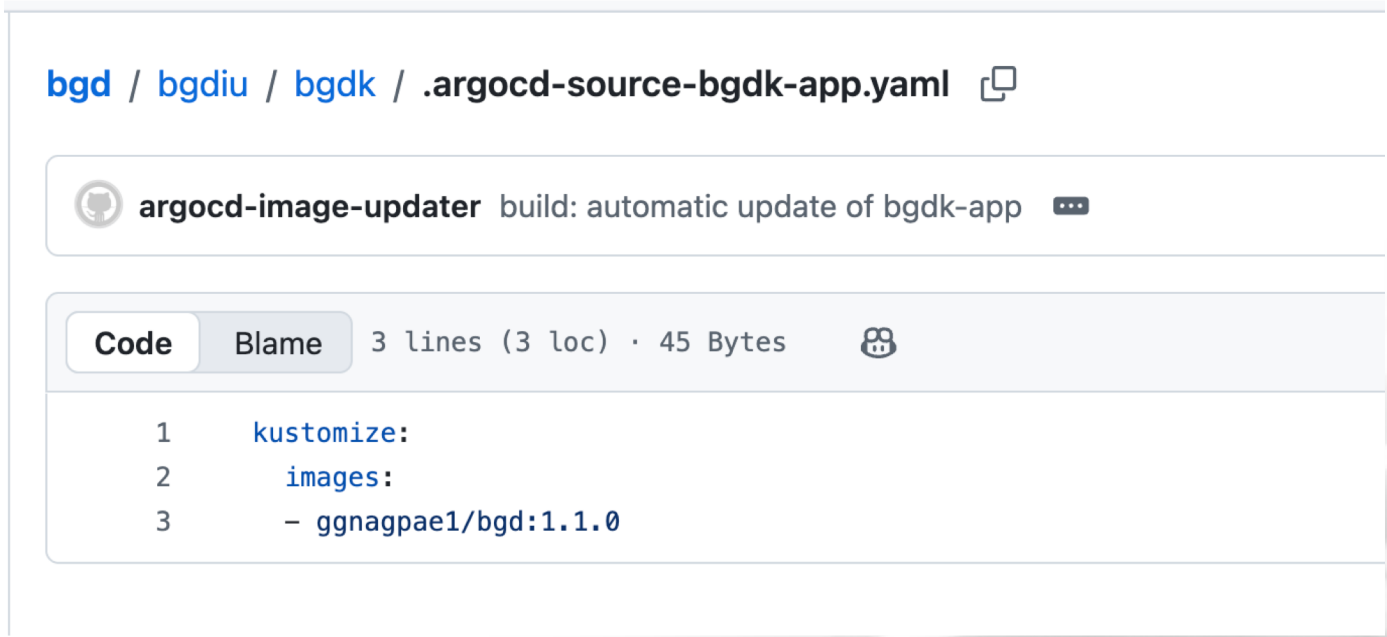
        time="2026-03-25T08:02:16Z" level=info msg="Starting image update cycle, considering 1 annotated application(s) for update"

        time="2026-03-25T08:02:17Z" level=info msg="Setting new image to ggnagpae1/bgd:1.1.0" alias=myalias application=bgdk-app image_name=ggnagpae1/bgd image_tag=1.0.0 registry=

        time="2026-03-25T08:02:17Z" level=info msg="Successfully updated image 'ggnagpae1/bgd:1.0.0' to 'ggnagpae1/bgd:1.1.0', but pending spec update (dry run=false)" alias=myalias application=bgdk-app image_name=ggnagpae1/bgd image_tag=1.0.0 registry=

# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ 이미지 업데이트
    - Git Hub 확인: .argocd-source-bgdk-app.yaml 파일이 생성됨



# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ 이미지 업데이트
    - 이미지에 태그를 다시 붙여서 Docker Hub에 업로드
      - docker tag ggnagpae1/bgd:1.0.0 ggnagpae1/bgd:1.1.0
      - docker push ggnagpae1/bgd:1.1.0
      - 2분 정도 후 로그 확인: kubectl logs argocd-image-updater-778d955ff5-lwqz9 -f -n argocd

        time="2026-03-25T08:02:16Z" level=info msg="Starting image update cycle, considering 1 annotated application(s) for update"

        time="2026-03-25T08:02:17Z" level=info msg="Setting new image to ggnagpae1/bgd:1.1.0" alias=myalias application=bgdk-app image_name=ggnagpae1/bgd image_tag=1.0.0 registry=

        time="2026-03-25T08:02:17Z" level=info msg="Successfully updated image 'ggnagpae1/bgd:1.0.0' to 'ggnagpae1/bgd:1.1.0', but pending spec update (dry run=false)" alias=myalias application=bgdk-app image_name=ggnagpae1/bgd image_tag=1.0.0 registry=

# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ 제약 설정
    - 업데이트 전략(update strategy)은 Argo CD IU가 새 버전을 찾는 방법
    - 지정하지 않으면 Argo CD IU는 시맨틱 버전(semantic version, semver)을사용하여 최신 버전을 찾음
    - 패치 버전만 자동으로 업데이트 하고자 하는 경우: argocd-image-updater.argoproj.io/image-list: myalias=ggnagpae1/bgd:1.2.x
    - 가장 최신 버전으로 업데이트 하고자 하는 경우: argocd-image-updater.argoproj.io/image-list: myalias=ggnagpae1/bgd: latest
    - 이미지의 digest가 변경된 경우 이미지 변경: argocd-image-updater.argoproj.io/myapp.update-strategy: digest 를 추가
    - 정규 표현식 이용

```
# 업데이트 전략을 'regexp'로 설정
argocd-image-updater.argoproj.io/myapp.update-strategy: regexp

# 허용할 태그 패턴 정의 (예: v로 시작하고 숫자로 구성된 태그)
# 예: v1.0, v2.1.3 등은 매칭되지만, latest나 dev는 제외됨
argocd-image-updater.argoproj.io/myapp.allow-tags: |
  ^v[0-9]+₩.[0-9]+.*$
```

# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ private 저장소 이용
    - secret 생성

```
kubectl create secret docker-registry dockerhub-secret ₩
  --n argocd ₩
  --docker-server=https://index.docker.io/v1/ ₩
  --docker-username=<사용자_ID> ₩
  --docker-password=<Access_Token_또는_비밀번호> ₩
  --docker-email=<이메일_주소>
```

# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ private 저장소 이용
    - 어노테이션에 추가해서 사용
      - ◆ 대상 이미지 지정

        argocd-image-updater.argoproj.io/image-list: myapp=docker.io/<사용자_ID>/<저장소명>
      - ◆ 위에서 만든 Secret 이름을 지정 (중요!)

        argocd-image-updater.argoproj.io/myapp.pull-secret: secret:argocd/dockerhub-secret
    - Image Updater는 개인 이미지 저장소에 대한 설정을 ConfigMap에서 가져오는데 argocd-image-update-config 라는 Configmap 리소스가 있는데 이 리소스의 데이터 절에 추가해서 사용
    - kubectl edit configmap argocd-image-updater-config -n argocd

# Argo CD를 사용한 애플리케이션 배포

- ImageUpdater
  - ✓ private 저장소 이용
    - Image Updater는 개인 이미지 저장소에 대한 설정을 ConfigMap에서 가져오는데 argocd-image-update-config 라는 ConfigMap 리소스가 있는데 이 리소스의 데이터 절에 추가해서 사용
      - kubectl edit configmap argocd-image-updater-config -n argocd

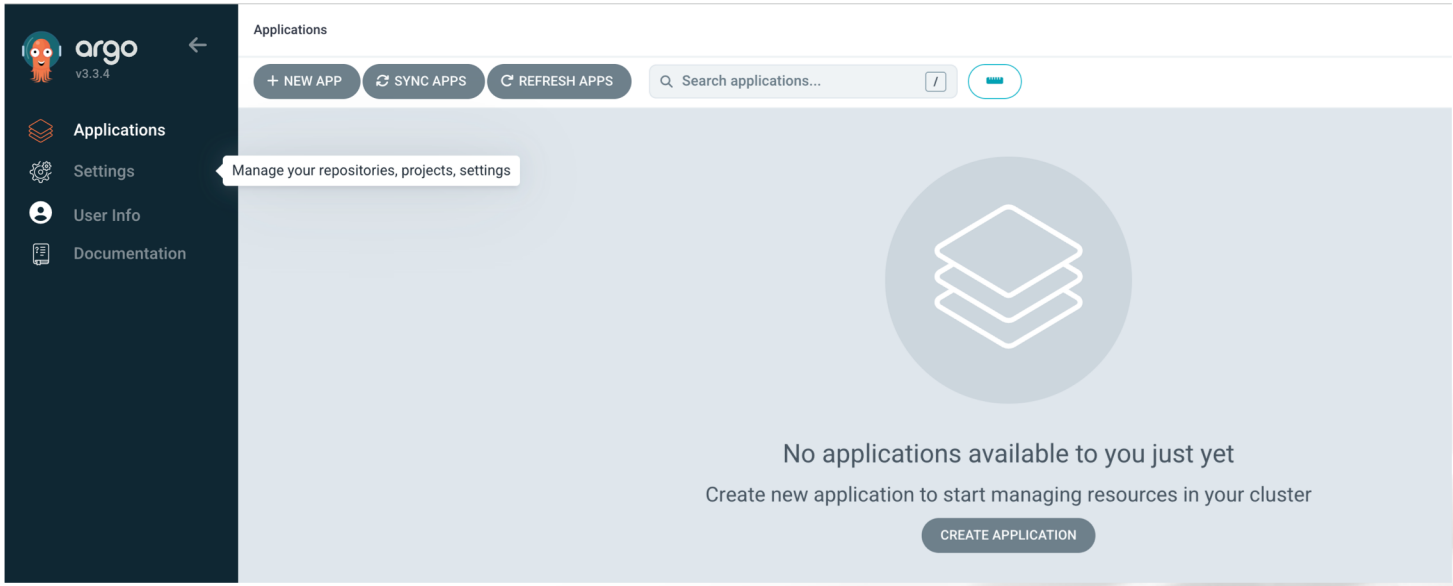
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-image-updater-config
  namespace: argocd
data:
  registries.conf: |
    registries:
    - name: Docker Hub
      api_url: https://index.docker.io/v1/
      prefix: docker.io
      ping: yes
      credentials: secret:argocd/dockerhub-secret # 앞서 만든 Secret 이름
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Private Git Repository
  - ✓ Argo CD CLI를 이용한 등록
    - 설정: argocd repo add <REPO_URL> --username <USER_NAME> --password <PERSONAL_ACCESS_TOKEN>
    - 확인: argocd repo list
    - 삭제: argocd repo rm <REPO_URL>

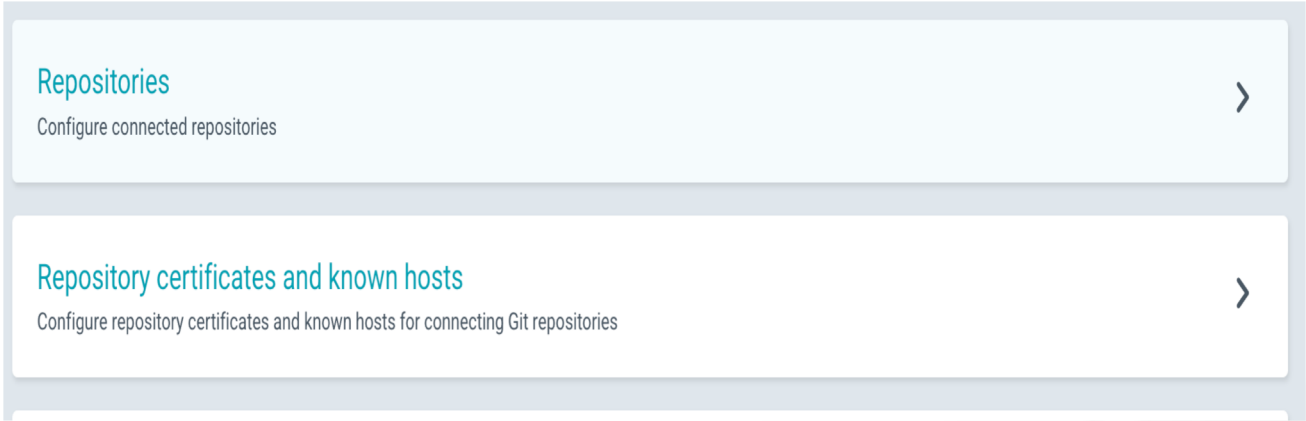
# Argo CD를 사용한 애플리케이션 배포

- Private Git Repository
  - ✓ Argo CD UI를 이용한 등록
    - 설정 버튼 클릭



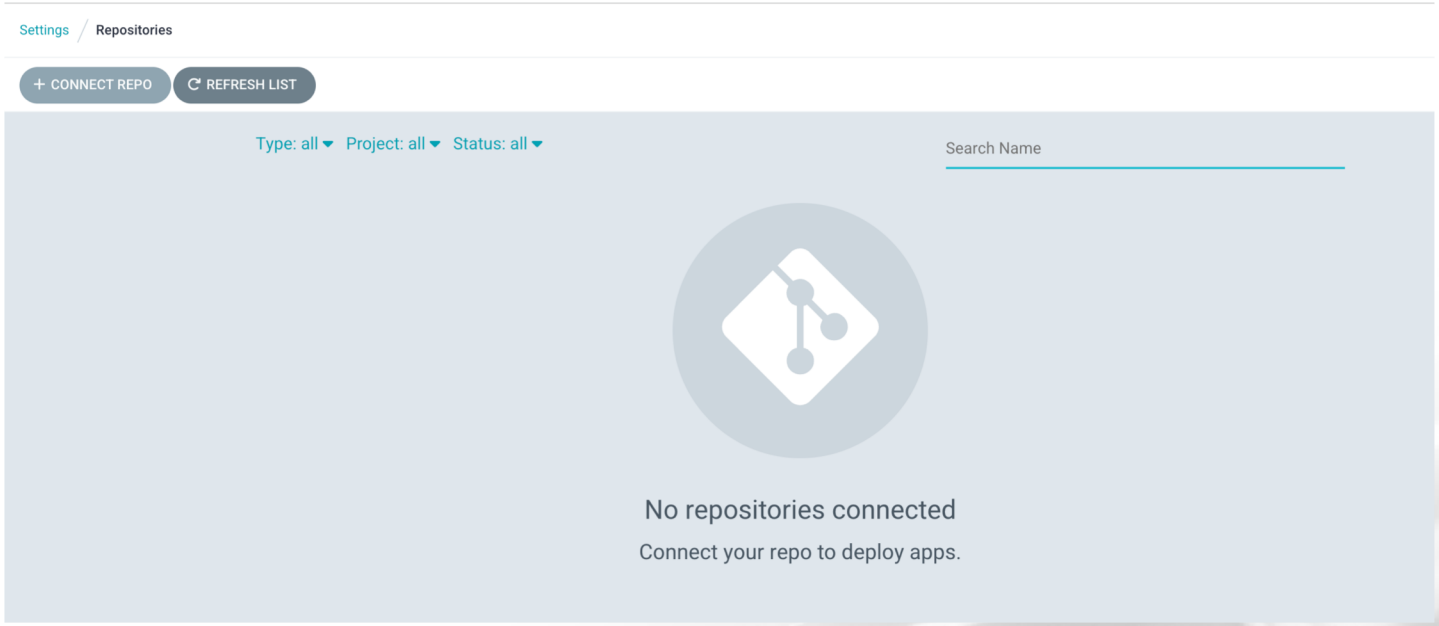
# Argo CD를 사용한 애플리케이션 배포

- ❖ Private Git Repository
  - ✓ Argo CD UI를 이용한 등록
    - ● Repositories 클릭



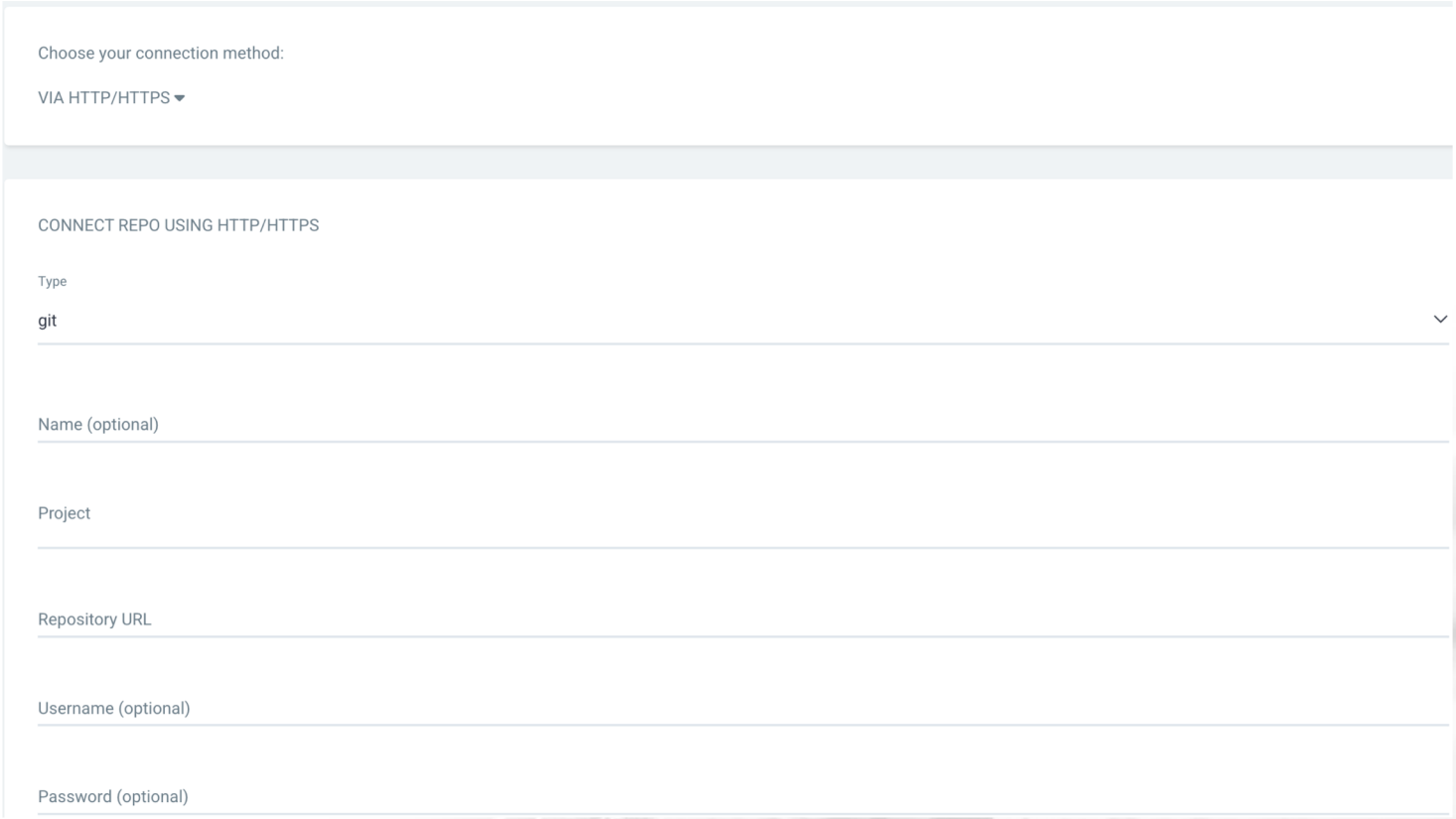
# Argo CD를 사용한 애플리케이션 배포

- ❖ Private Git Repository
  - ✓ Argo CD UI를 이용한 등록
    - ● CONNECT REPO 클릭



# Argo CD를 사용한 애플리케이션 배포

- ❖ Private Git Repository
  - ✓ Argo CD UI를 이용한 등록
    - ● 필요한 정보 설정



# Argo CD를 사용한 애플리케이션 배포

- ❖ Private Git Repository
  - ✓ Secret 이용
    - Secret 생성

```
apiVersion: v1
kind: Secret
metadata:
  name: private-repo-secret
  namespace: argocd
  labels:
    # 이 레이블이 있어야 Argo CD가 저장소 정보로 인식합니다.
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/your-user/your-repo.git
  # GitHub PAT 또는 GitLab Token 입력
  username: your-username
  password: your-personal-access-token
```

    - Secret 적용

```
kubectl apply -f repo-secret.yaml
```

    - Repo 확인

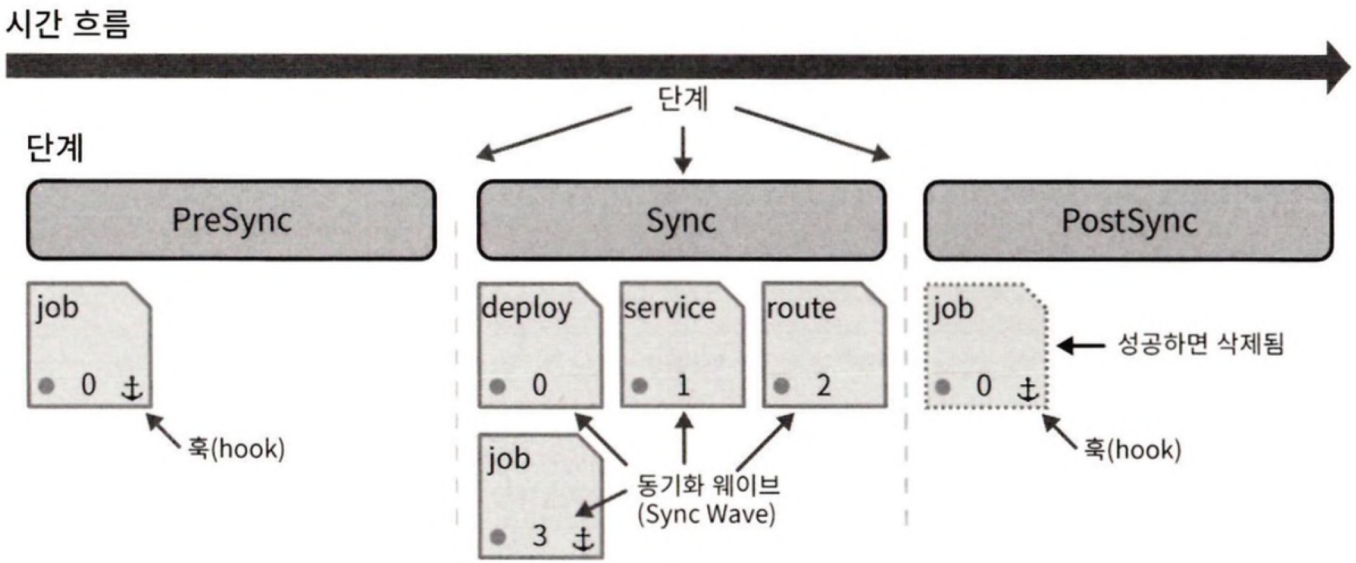
```
argocd repo list
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 기본적인 규칙
    - 종류가 다른 경우
      1. Namespace
      2. NetworkPolicy
      3. LimitRange
      4. ServiceAccount
      5. Secret
      6. ConfigMap
      7. StorageClass
      8. PersistentVol 니 me
      9. ClusterRole
      10. Role
      11. Service
      12. DaemonSet
      13. Pod
      14. ReplicaSet
      15. Deployment
      16. StatefulSet
      17. Job
      18. Ingress
    - 종류가 같으면 알파벳 순

# Argo CD를 사용한 애플리케이션 배포

- Manifest 배포 순서
  - ✓ Argo CD의 리소스 적용 과정
    - 매니페스트를 적용하기 전에 실행되는 PreSync 단계
    - 매니페스트가 실제로 적용되는 Sync 단계
    - 모든 매니페스트가 적용되고 동기화된 후에 실행되는 PostSync 단계



# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 동기화 웨이브(sync wave)와 리소스 훅(resource hook)을 사용하면 매니페스트를 적용하는 기본 순서를 수정할 수 있음
  - ✓ Resource Hook
    - 리소스 훅은 특정 단계에서 실행되는 스크립트이며 동기화 단계가 실패한 경우에는 롤백 작업을 실행하는 데도 이용될 수 있음
    - 사용 가능한 리소스 훅의 목록

| Hook | 설명 | 용례 |
|---|---|---|
| PreSync | 매니페스트 적용 전에 실행 | 데이터베이스 마이그레이션 |
| Sync | 매니페스트 적용과 동시에 실행 | 카나리 배포(Canary Release) 또는 비공개 출시(Dark Launch)와 같은 복잡한 롤링 업데이트 전략 |
| PostSync | 모든 Sync 훅이 성공적으로 완료된 후 실행 | 배포가 올바르게 수행되었는지 확인하기 위한 테스트 실행 |
| SyncFail | 동기화 작업이 실패하면 실행 | 실패 시 롤백 작업 |
| Skip | 매니페스트 적용 건너뛰기 | 애플리케이션 배포에 수동 작업이 필요한 경우(예: 사용자 트래픽을 새 버전에 할당) |

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ Resource Hook
    - ● PostSync를 이용하는 방법

```
apiVersion: batch/vl
kind: Job
metadata:
  name: todo-insert #Job 이름
  annotations:
    argocd.argoproj.io/hook: PostSync #매니페스트가 적용될 때 설정됨
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ Resource Hook
    - PostSync를 이용하는 방법

```
apiVersion: batch/vl
kind: Job
metadata:
  name: todo-insert #Job 이름
  annotations:
    argocd.argoproj.io/hook: PostSync #매니페스트가 적용될 때 설정됨
```

    - 삭제 정책
      - ◆ 훅은 실행이 끝나도 삭제되지 않음
      - ◆ 훅으로 설정된 쿠버네티스 Job은 실행이 끝나면 단순히 상태가 Completed로 바뀌기만 하는데 이 상태가 원하는 상태일 수도 있지만 argocd.argoproj.io/hook-delete-policy 어노테이션을 사용하여 policy 값을 설정하면 훅 리소스가 자동 삭제되도록 할 수 있음
      - ◆ 정책
        - HookSucceeded: 훅 실행이 성공한 후 삭제됨
        - HookFailed: 훅 실행이 실패한 후 삭제됨
        - BeforeHookCeation: 새 훅이 생성되기 전에 삭제됨

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ Sync Wave
    - Argo CD가 Git에 저장된 매니페스트를 어떤 순서로 적용할지 명시하는 방법으로 모든 매니페스트는 기본적으로 웨이브 값 0을 갖음
    - 웨이브 값이 낮은 리소스가 먼저 적용되는데 웨이브 값을 다르게 지정하고 싶은 경우에는 argocd.argoproj.io/sync-wave 어노테이션을 사용하여 리소스에 웨이브 값을 할당
    - 데이터베이스를 먼저 배포한 다음 데이터베이스 스키마를 만들고 싶다면 데이터베이스 배포 파일의 sync-wave를 스키마 생성 작업보다 낮게 설정

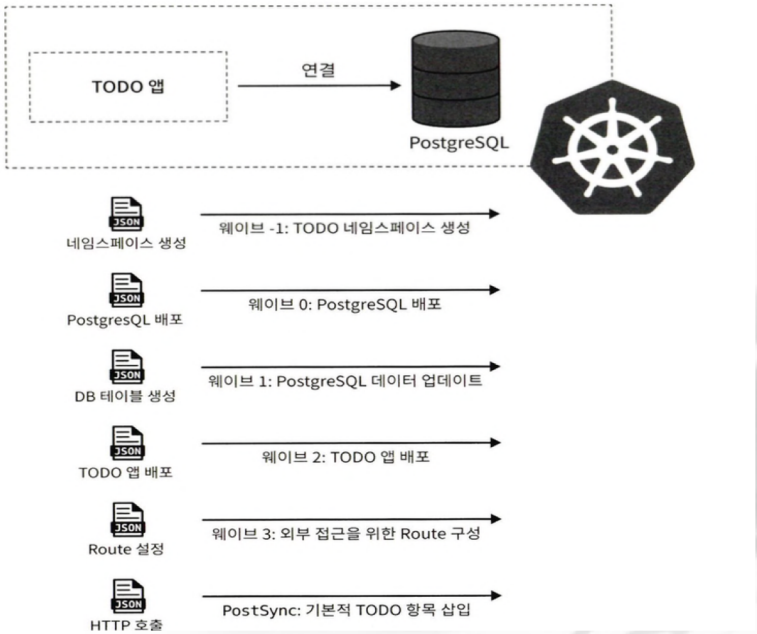
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgresql
  namespace: todo
  annotations:
    argocd.argoproj.io/sync-wave: "0"
apiVersion: batch/v1
  kind: Job
  metadata:
    name: todo-table
    namespace: todo
    annotations:
      argocd.argoproj.io/sync-wave: "1"
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 적용 순서 결정
    - 단계(Phase)
    - Wave(낮은 값 우선)
    - 종류(Kind)
    - 이름(Name)

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - 데이터베이스(PostgreSQL)에 할 일 목록을 저장하는 TODO 애플리케이션
    - 애플리케이션을 배포하려 면 데이터베이스 스키마 생성 전에 데이터베이스 서버를 실행하는 등 순서에 관한 요건 몇 가지를 충족해야 함
    - 전체 애플리케이션 동기화 후에 실행할 매니페스트를 위해 몇가지 기본 TODO 항목들을 데이터베이스에 삽입해야 함
    - 전체 프로세스



# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 네임스페이스: todo-namespace.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: todo
  annotations:
    argocd.argoproj.io/sync-wave: "-1"
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 데이터베이스 파드: postgres-deployment.yaml

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgresql
  namespace: todo
  annotations:
    argocd.argoproj.io/sync-wave: "0"
spec:
  selector:
    matchLabels:
      app: postgresql
  template:
    metadata:
      labels:
        app: postgresql
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 데이터베이스 파드: postgres-deployment.yaml

```
spec:
  containers:
    - name: postgresql
      image: quay.io/redhatdemo/openshift-pgsql12-primary:centos7
      imagePullPolicy: Always
      ports:
        - name: tcp
          containerPort: 5432
      env:
        - name: PG_USER_PASSWORD
          value: admin
        - name: PG_USER_NAME
          value: admin
        - name: PG_DATABASE
          value: todo
        - name: PG_NETWORK_MASK
          value: all
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 데이터베이스 서비스: postgres-service.yaml

```
---
apiVersion: v1
kind: Service
metadata:
  name: postgresql
  namespace: todo
  annotations:
    argocd.argoproj.io/sync-wave: "0"
spec:
  selector:
    app: postgresql
  ports:
    - name: pgsql
      port: 5432
      targetPort: 5432
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 데이터베이스 삽입 작업: postgres-service.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: todo-table
  namespace: todo
  annotations:
    argocd.argoproj.io/sync-wave: "1"
spec:
  template:
    spec:
      containers:
        - name: postgresql-client
          image: postgres:12
          imagePullPolicy: Always
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 데이터베이스 테이블 생성 작업: postgres-service.yaml

```
env:
  - name: PGPASSWORD
    value: admin
command: ["psql"]
args:
  [
    "--host=postgresql",
    "--username=admin",
    "--no-password",
    "--dbname=todo",
    "--command=create table Todo (id bigint not null,completed
boolean not null,ordering integer,title varchar(255),url varchar(255),primary
key (id));create sequence hibernate_sequence start with 1 increment by 1;",
  ]
restartPolicy: Never
backoffLimit: 1
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 애플리케이션 파드 작업: todo-deployment.yaml

```
---
apiVersion: "v1"
kind: "ServiceAccount"
metadata:
  labels:
    app.kubernetes.io/name: "todo-gitops"
    app.kubernetes.io/version: "1.0.0"
  name: "todo-gitops"
  namespace: todo
  annotations:
    argocd.argoproj.io/sync-wave: "2"
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 애플리케이션 파드 작업: todo-deployment.yaml

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
  labels:
    app.kubernetes.io/name: "todo-gitops"
    app.kubernetes.io/version: "1.0.0"
  name: "todo-gitops"
  namespace: todo
  annotations:
    argocd.argoproj.io/sync-wave: "2"
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: "todo-gitops"
      app.kubernetes.io/version: "1.0.0"
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 애플리케이션 파드 작업: todo-deployment.yaml

```
template:
  metadata:
    labels:
      app.kubernetes.io/name: "todo-gitops"
      app.kubernetes.io/version: "1.0.0"
  spec:
    containers:
    - env:
      - name: "KUBERNETES_NAMESPACE"
        valueFrom:
          fieldRef:
            fieldPath: "metadata.namespace"
      image: "quay.io/rhdevelopers/todo-gitops:1.0.0"
      imagePullPolicy: "Always"
      name: "todo-gitops"
      ports:
      - containerPort: 8080
        name: "http"
        protocol: "TCP"
    serviceAccount: "todo-gitops"
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 애플리케이션 서비스 작업: todo-service.yaml

```
---
apiVersion: "v1"
kind: "Service"
metadata:
  labels:
    app.kubernetes.io/name: "todo-gitops"
    app.kubernetes.io/version: "1.0.0"
  name: "todo-gitops"
  annotations:
    argocd.argoproj.io/sync-wave: "2"
  namespace: todo
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 애플리케이션 서비스 작업: todo-service.yaml

```
spec:
  ports:
  - name: "http"
    port: 8080
    targetPort: 8080
    nodePort: 31080
  type: NodePort
  selector:
    app.kubernetes.io/name: "todo-gitops"
    app.kubernetes.io/version: "1.0.0"
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 애플리케이션 데이터 삽입 작업: todo-insert-data.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: todo-insert
  annotations:
    argocd.argoproj.io/hook: PostSync # <1>
    argocd.argoproj.io/hook-delete-policy: HookSucceeded
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ 애플리케이션 데이터 삽입 작업: todo-insert-data.yaml

```
spec:
  ttlSecondsAfterFinished: 100
  template:
    spec:
      containers:
        - name: httpie
          image: alpine/httpie:2.4.0
          imagePullPolicy: Always
          command: ["http"]
          args:
            [
              "POST",
              "todo-gitops:8080/api",
              "title=Finish ArgoCD tutorial",
              "--ignore-stdin"
            ]
      restartPolicy: Never
  backoffLimit: 1
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - ● 쿠버네티스 매니페스트 생성
      - ◆ Git Hub에 push

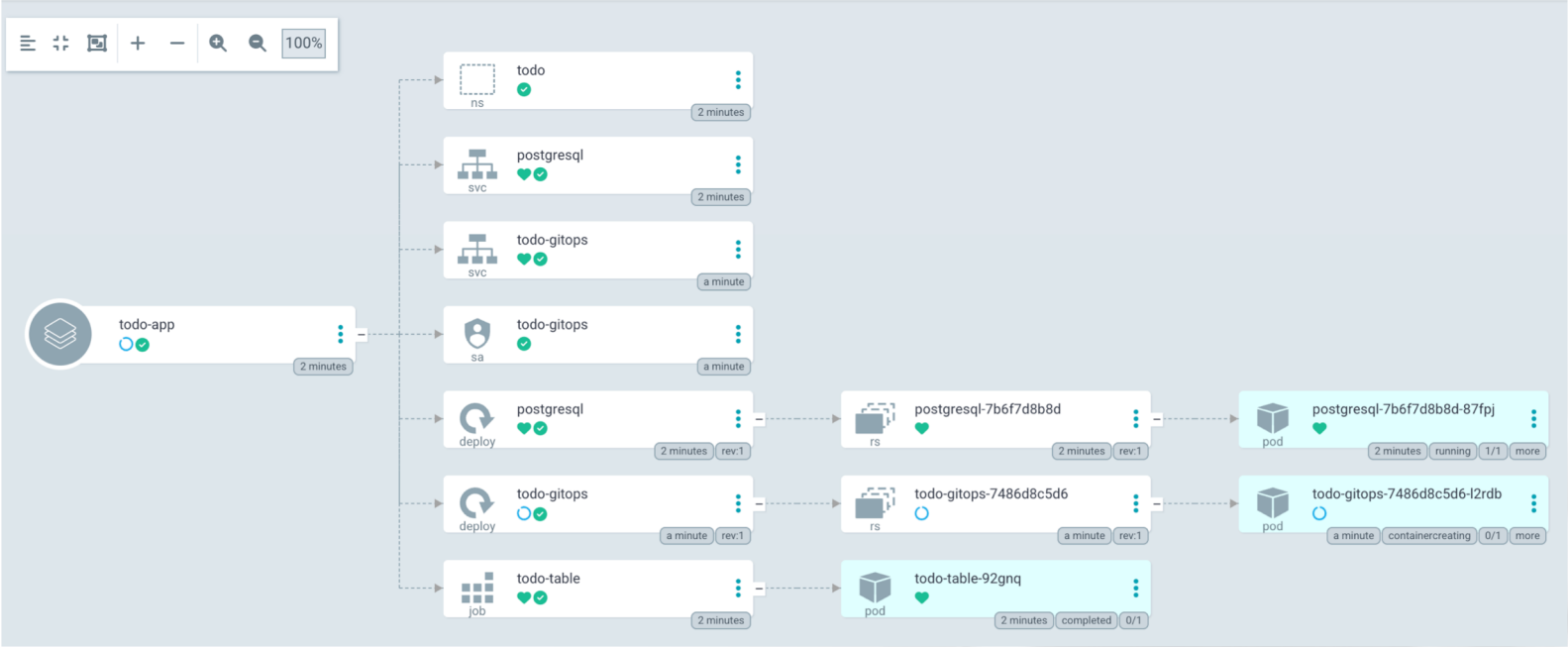
# Argo CD를 사용한 애플리케이션 배포

- ❖ Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - Argocd 배포 파일 생성: bgd-order.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: todo-app
  namespace: argocd
spec:
  destination:
    namespace: todo
    server: https://kubernetes.default.svc
  project: default
  source:
    repoURL: https://github.com/itstudy001/bgd.git
    path: todo
    targetRevision: main
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
    - CreateNamespace=true
```

# Argo CD를 사용한 애플리케이션 배포

- Manifest 배포 순서
  - ✓ 배포 파일, 동기화 웨이브, 훅을 사용하여 애플리케이션 배포
    - 배포 후 확인 – 순서대로 수행



# Argo CD를 사용한 애플리케이션 배포

- ❖ 동기화 윈도 정의
  - ✓ 개요
    - Argo CD는 애플리케이션 동기화를 차단하거나 허용할 시간대를 구성할 수 있도록 동기화 윈도(synchronization win-dow)라는 개념을 지원
    - 동기화 윈도를 정의하려면 AppProject 매니페스트를 만들어 주면 됨
    - 신경 써서 설정해야 하는 절은 allow, denay 중 한 값을 갖는 kind, 동기화 윈도의 시작 시간을 cron 형식으로 지정할 수 있도록 하는 schedule, 동기화 윈도의 크기를 나타내는 duration, 그리고 동기화 윈도를 적용할 리소스(Application일 수도 있고 네임스페이스일 수도 있고 클러스터일 수도 있음)를 나타내는 application

# Argo CD를 사용한 애플리케이션 배포

- 동기화 윈도 정의
  - ✓ 22:00 ~ 23:00 한 시간 동안만 이름이 –prod로 끝나는 경우만 동기화를 허용

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: default
spec:
  syncWindows:
  - kind: allow
    schedule: '0 22 * * * '
    duration: 1h
    applications:
    - '*-prod'
```

# Argo CD를 사용한 애플리케이션 배포

- ❖ 동기화 윈도 정의
  - ✓ 수동 동기화 허용
    - CLI 명령으로 수행: argocd proj windows enable-manual-sync <PROJECT_ID>
    - yaml 파일로 설정

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: default
spec:
  syncWindows:
  - kind: deny
    schedule: '0 22 * * * '
    duration: 1h
    manualSync: true
    namespaces:
    - bgd
  - kind: allow
    schedule: '0 23 * * * '
    duration: 1h
    clusters:
    - prod-cluster
```

# GitHub Actions

- ❖ 간접 배포 방식(GitOps 표준)
  - ✓ GitHub Actions가 ArgoCD를 직접 호출하지 않고 애플리케이션의 설정 파일(Manifest)이 담긴 Git 저장소의 값을 업데이트하는 방식
  - ✓ 동작 과정
    - GitHub Actions에서 소스 코드를 빌드하고 Docker 이미지를 푸시
    - 새로운 이미지 태그(예: v1.2.0)를 Deployment YAML이나 Kustomize/Helm 설정 파일에 업데이트하고 커밋/푸시
    - ArgoCD가 해당 Git 저장소의 변경을 감지하고, 클러스터의 상태를 자동으로 업데이트(Auto-Sync)
  - ✓ 장점: GitHub Actions에 쿠버네티스나 ArgoCD 접근 권한을 줄 필요가 없어 보안상 안전

# GitHub Actions

- 직접 배포 방식(Sybc Trigger)
  - ✓ 배포 속도를 높이거나, 배포 성공 여부를 GitHub Actions에서 즉시 확인해야 할 때 ArgoCD CLI나 API를 직접 호출하여 동기화를 명령하는 방식
  - ✓ 동작 과정
    - 빌드 및 이미지 푸시 완료 후, GitHub Actions에서 argocd app sync 명령을 실행
    - ArgoCD 서버에 접속하여 즉시 동기화를 시작하도록 트리거
  - ✓ 장점: ArgoCD의 폴링(Polling) 대기 시간 없이 즉시 배포가 시작됨
  - ✓ 활용 도구: GitHub Marketplace에 있는 ArgoCD Sync Action 등을 사용하면 간편하게 설정할 수 있음

# GitHub Actions

❖ 간접 배포 와 직접 배포 비교

| | 간접 배포 (Git Update) | 직접 배포 (Sync Trigger) |
|---|---|---|
| 보안 | 높음 (Git 권한만 필요) | 보통 (ArgoCD 서버 권한 필요) |
| 속도 | ArgoCD 감지 주기만큼 대기 | 즉시 실행 |
| 적합한 경우 | 표준 GitOps 운영 시 | 빠른 피드백이 필요한 개발 환경 |

✓ Actions는 빌드(CI)를 담당하고 ArgoCD는 배포(CD)를 담당하는 구조로 연동하는 것이 현대적인 배포 파이프라인의 정석

# GitHub Actions

- GitHub Webhook
  - ✓ ArgoCD의 Webhook 설정은 GitHub Actions와 같은 CI 도구와 ArgoCD 사이의 속도 문제를 해결해 주는 핵심 요소
  - ✓ 기본적으로 ArgoCD는 3분마다 Git 저장소를 확인(Polling)하는데, Webhook을 설정하면 소스 코드가 푸시되는 즉시 ArgoCD가 변경 사항을 감지하여 배포를 시작
  - ✓ 필요한이유
    - 즉각적인 동기화: 기본 3분의 대기 시간 없이 수 초 내에 배포가 시작됨
    - 리소스 효율성: ArgoCD가 계속 Git 서버에 "변경사항 있니?"라고 물어보는 부하를 줄여줌
    - 이벤트 기반: 푸시라는 사건이 발생했을 때만 동작하므로 더 정교한 자동화가 가능

# GitHub Actions

- GitHub Webhook
  - ✓ GitHub - ArgoCD Webhook 설정 방법
    - ArgoCD API 서버 주소 확인
      - ArgoCD 외부 접속 주소가 필요(예: https://argocd.example.com)
      - Webhook 엔드포인트는 항상 다음과 같음 - https://<ARGOCD_SERVER_URL>/api/webhook
    - GitHub 저장소 설정
      - GitHub 저장소의 Settings > Webhooks > Add webhook으로 이동
      - Payload URL: 위에서 확인한 엔드포인트 주소를 입력
      - Content type: application/json을 선택
      - Secret: 보안을 위해 복잡한 문자열을 입력(나중에 ArgoCD 설정에도 사용됨)
      - Events: Just the push event를 선택

# GitHub Actions

- ❖ GitHub Webhook
  - ✓ GitHub - ArgoCD Webhook 설정 방법
    - ● ArgoCD에 Secret 등록 (Kubernetes)

```
apiVersion: v1
kind: Secret
metadata:
  name: argocd-secret
  namespace: argocd
  labels:
    app.kubernetes.io/part-of: argocd
type: Opaque
data:
  # GitHub에서 입력한 Secret 문자열을 base64로 인코딩하여 입력
  webhook.github.secret: <BASE64_ENCODED_SECRET>
```

# GitHub Actions

- GitHub Webhook
  - ✓ 주의사항
    - 네트워크 노출: ArgoCD 서버가 인터넷(공용망)에 노출되어 있거나 GitHub의 Webhook IP 대역을 허용(Allowlist)해야 함
    - TLS/SSL: https 사용을 강력히 권장하는데 자가 서명 인증서(Self-signed)를 사용하는 경우 GitHub 설정에서 Disable SSL verification을 체크해야 할 수도 있음
    - Application 설정: ArgoCD Application 설정에서 Self-Heal이나 Auto-Sync가 활성화되어 있어야 Webhook 수신 후 즉시 실제 클러스터 반영까지 이어짐
  - ✓ GitHub Actions와 연합 작전
    - GitHub Actions: 코드를 빌드하고 이미지 태그를 Manifest 저장소에 커밋
    - GitHub Webhook: 커밋이 발생하자마자 ArgoCD에게 알림을 보냄
    - ArgoCD: 알림을 받자마자 Git을 당겨와서(Pull) 클러스터를 업데이트

# GitHub Actions

- ArgoCD Webhook
  - ✓ Argo CD는 3분마다 Git 저장소를 폴링하여 쿠버네티스 매니페스트의 변경 사항을 감지하지만 깃허브, 깃랩 또는 빗버킷 같은 인기 있는 Git 서버의 웹훅 알림을 통한 이벤트 중심 접근법도 지원
  - ✓ Argo CD 웹훅(Webhook)은 Argo CD를 설치하면 활성화되며 앤드포인트 /api/webhook에서 사용할 수 있음
  - ✓ 미니큐브나 도커 데스크톱 등을 사용하여 Argo CD 웹훅을 테스트하려면 다음과 같이 헬름을 사용하여 Go로 작성 된 오픈 소스 경량 Git 서버 Gitea를 설치하면 됨
    - helm repo add gitea-charts https://dl.gitea.io/charts/
    - helm install gitea gitea-charts/gitea