

# ArgoCD ApplicationSet 설명



argoCD에서 yaml파일의 kind가 ApplicationSet에 대해서 자세  
히 알려줘.



`kind: ApplicationSet` 은 ArgoCD에서 여러 **Application** 리소스를 한 번에 생성/관리하기 위한 상위 개념이에요.

보통 ArgoCD의 기본 단위는 `**Application**` 하나죠. 하지만 팀 단위, 환경 단위(개발/스테이징/운영), 리전 단위 등으로 수십~수백 개의 애플리케이션을 선언하려면 Application 리소스를 일일이 만들기 번거롭습니다. 이때 ApplicationSet을 쓰면 템플릿 + 생성 규칙을 정의해서 자동으로 여러 Application을 만들어줘요.

## 구조

기본적인 YAML 골격은 이렇습니다:

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: my-applicationset
spec:
  generators: # Application들을 어떻게 만들어낼지 정의
    - list: # list, git, cluster, matrix, pullRequest 등 다양한 generator 사용 가능
      elements:
        - cluster: dev
          url: https://kubernetes.default.svc
        - cluster: prod
          url: https://1.2.3.4
  template: # 생성할 Application의 공통 템플릿
    metadata:
      name: '{{cluster}}-myapp' # generator 값들을 변수로 참조 가능
    spec:
      project: default
```

```
source:
  repoURL: https://github.com/example/repo.git
  targetRevision: HEAD
  path: apps/myapp
destination:
  server: '{{url}}'
  namespace: myapp
syncPolicy:
  automated: {}
```

---

## 주요 개념

### 1. Generators

- 어떤 Application들을 만들지 **목록**을 정의하는 부분입니다.
- 종류:
  - ``list``: YAML 안에 직접 리스트 정의
  - ``git``: 특정 Git repo의 디렉토리/파일 구조를 스캔해서 Application 자동 생성
  - ``cluster``: ArgoCD에 등록된 여러 클러스터 대상
  - ``matrix``: 여러 generator를 조합해서 Cartesian product 생성
  - ``pullRequest``: GitHub/GitLab PR 기반으로 앱 생성
  - ``scmProvider``: GitHub/GitLab 등에서 repo 리스트를 자동으로 가져옴

### 2. Template

- 각 Application의 공통 스펙을 정의합니다.
- ``{{ 변수 }}`` 문법으로 generator가 만든 값들을 주입할 수 있어요.
- 결국 최종적으로 여러 개의 ``Application`` 리소스가 생성됩니다.

### 3. Controller 동작

- ApplicationSet Controller라는 별도 컨트롤러가 ArgoCD에 설치되어 있어야 함.
- 이 컨트롤러가 generators → template을 조합해서 Application을 생성/관리합니다.
- 따라서 ApplicationSet은 직접 배포를 안 하고, 실제 배포 단위인 Application들을 **자동 생성**해주는 선언형 "팩토리" 같은 역할을 합니다.

---

## 장점

- 수십 개의 Application을 일관된 방식으로 쉽게 생성.
- Git 디렉토리 구조, 클러스터 목록, PR 등 동적인 조건에 따라 Application 자동 반영.
- 반복적인 Application YAML 복붙 방지.

---

## 간단한 예시

환경(dev/prod)별 같은 앱을 배포하고 싶을 때:

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: demo-appset
spec:
  generators:
    - list:
        elements:
          - env: dev
            url: https://kubernetes.default.svc
          - env: prod
            url: https://1.2.3.4
  template:
    metadata:
      name: '{{env}}-demo'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/demo.git
        targetRevision: HEAD
        path: apps/demo
      destination:
        server: '{{url}}'
        namespace: demo
```

→ 결과적으로 `dev-demo` Application과 `prod-demo` Application이 자동 생성됩니다.

---

혹시 원하시면 제가 `Application`과 `ApplicationSet`의 차이를 표로 정리해 드릴 수도 있는데, 그게 더 보기 편하시겠어요?



응



좋습니다. 두 리소스의 차이를 표로 정리해드릴게요.

항목	Application	ApplicationSet
역할	Git 리포지토리의 특정 경로를 쿠버네티스에 배포하는 단일 앱 단위	여러 Application을 자동으로 생성·관리하는 팩토리(템플릿 + 생성 규칙)
선언 방식	<code>`kind: Application`</code>	<code>`kind: ApplicationSet`</code>
컨트롤러	ArgoCD 기본 컨트롤러	ApplicationSet Controller (별도 설치 필요)
관리 범위	하나의 환경/하나의 클러스터/하나의 네임스페이스 대상	여러 환경, 여러 클러스터, 여러 네임스페이스를 동시에 커버 가능
중복성	비슷한 앱이 많아지면 YAML을 복붙해서 여러 개 관리해야 함	공통 <code>`template`</code> 에 변수( <code>`{{ }}`</code> )를 넣고, <code>`generators`</code> 로 반복 생성
주요 기능	<ul style="list-style-type: none"> <li>- GitOps의 기본 단위</li> <li>- 동기화 정책 (manual/auto)</li> <li>- health check, rollback 지원</li> </ul>	<ul style="list-style-type: none"> <li>- List, Git, Cluster, PR 등 다양한 generator로 Application 생성</li> <li>- Matrix 조합 가능</li> <li>- 대규모 환경에서 자동화 유리</li> </ul>
결과물	그 자체로 배포 실행	여러 개의 Application 리소스들을 만들어냄

정리하면,

- **Application** → "앱 하나를 어떻게 배포할지 정의"
- **ApplicationSet** → "앱 여러 개를 어떻게 찍어낼지 정의"

혹시 제가 예제로 같은 **Git repo**를 환경(dev/prod/staging)별로 **ApplicationSet**으로 생성하는 YAML을 만들어드리면 이해가 더 쉬울까요?



yaml 템플릿을 만들어 주고 자세한 설명도 부탁해.  
generators 이하에 list.elements 하위 항목에 cluster,  
enviroment 같은게 포함되도록해서.



1m 2s 동안 생각함

- 사용자가 "ApplicationSet" 템플릿을 요청했으니, "generators" 목록에 "cluster"와 "environment"를 포함한 항목을 제공할 것입니다. 사용자 요청에서는 "enviroment"로 철자가 틀렸으나 정확히 따라야 혼란을 피할 수 있을 듯하고, "environment"로 수정하는 것이 더 바람직합니다. 그리고 "minikube wsl 2"는 필수는 아닙니다.
- 사용자에게 템플릿 이름 지정, 레이블/주석, 자동 prune 및 selfHeal을 포함한 syncPolicy, retry, syncOptions 및 Helm 옵션에 대한 모범 사례를 제공할 것입니다. 또한 repoURL과 path를 위한 자리 표시자, 명령어, validation을 강조할 필요가 있습니다. YAML과 함께 각 필드와 예시, kubectl 명령어를 제공하고, 오류 방지 사항도 안내할 거예요.
- 사용자가 Helm과 ArgoCD를 사용하기 때문에, Helm의 파라미터와 값 오버라이드를 "helm.parameters"나 "helm.valueFiles"를 통해 제시할 수 있습니다. 또한, 환경별 값을 위해 "valuesFile" 경로 또는 "valueOverrides"를 사용하며, ApplicationSet 템플릿에 이를 반영할 수 있습니다. 예시로 헬름과 아르고CD를 함께 보여주겠습니다.
- 현재 ApplicationSet은 고유한 템플릿 스타일을 사용하여, `{{ }}`로 표기된 변수들을 포함합니다. goTemplate 함수는 필요하지 않으며, 그냥 기본적인 텍스트 템플릿 방식으로 파라미터를 선언합니다. 예를 들어 `{{cluster}}`, `{{environment}}` 같은 변수들을 사용하는 걸로, 이렇게 하는 게 더 안전합니다.

🕒 완료

좋아요. **list.generator**로 `cluster`, `environment` 값을 넣어 여러 Application을 찍어내는 표준 템플릿이랑, 필드별 설명/주의점을 한번에 드릴게요.

(혹시 철자를 `enviroment`로 쓰고 싶으시면, 템플릿의 `{{environment}}`만 `{{enviroment}}`로 바꾸시면 됩니다.)

## ApplicationSet YAML 템플릿 (list.elements 사용)

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: team-a-apps
  namespace: argocd
```

```

spec:
  generators:
    - list:
        elements:
          # --- dev 환경 (in-cluster)
          - app: web
            cluster: in-cluster
            url: https://kubernetes.default.svc
            namespace: web-dev
            environment: dev
            path: apps/web # repo 내 경로
            valuesFile: values-dev.yaml
            revision: main # 필요 없으면 삭제 가능

          # --- prod 환경 (외부 클러스터)
          - app: web
            cluster: prod-cluster
            url: https://k8s-prod.example.com
            namespace: web
            environment: prod
            path: apps/web
            valuesFile: values-prod.yaml
            revision: v1.2.3

  template:
    metadata:
      name: '{{environment}}-{{app}}'
      labels:
        app.kubernetes.io/part-of: '{{app}}'
        app.kubernetes.io/managed-by: applicationset
        env: '{{environment}}'
        cluster: '{{cluster}}'
    spec:
      project: default
      source:
        repoURL: https://github.com/your-org/your-repo.git
        targetRevision: HEAD # 공통 브랜치(요소별로 쓰려면 아래 참고)
        path: '{{path}}'
      helm:
        valueFiles:
          - 'helm/{{app}}/{{valuesFile}}'

```

```

parameters:
  # 필요 시 per-app/per-env 파라미터 예시
  - name: image.tag
    value: '{{revision}}'      # 요소에 revision이 있으면 사용, 없으면 빈
값(삭제해도 됨)
destination:
  server: '{{url}}'
  namespace: '{{namespace}}'
syncPolicy:
  automated:
    prune: true
    selfHeal: true
  syncOptions:
    - CreateNamespace=true
retry:
  backoff:
    duration: 10s
    factor: 2
    maxDuration: 2m

```

브랜치를 요소별로 다르게 쓰고 싶다면(예: dev=main, prod=tag)

``spec.template.spec.source.targetRevision: HEAD`` 대신 아래처럼 바꾸세요.

(Go 템플릿 기능을 쓰면 default 처리도 가능하지만, 기본은 가장 단순한 치환만 권장합니다.)

```

spec:
  goTemplate: true
  generators: ...
  template:
    spec:
      source:
        targetRevision: '{{ default "HEAD" .revision }}'

```

``goTemplate: true``를 쓰면 ``default``, ``quote`` 같은 함수 사용이 가능합니다.

## 필드별 설명 (핵심만 콕)

- **generators.list.elements**

동일한 템플릿으로 여러 Application을 찍어낼 “자료행(row)”들입니다. 각 객체의 키가 템플릿 변수로 그대로 들어갑니다.

- ``app``: 앱 식별자(이름 구성, values 파일 경로 등에 재사용)
- ``cluster``: 사람 친화적 표기(라벨로만 사용), 실제 배포 대상은 ``url``
- ``url``: ArgoCD에 등록된 대상 클러스터 API 서버 주소
  - in-cluster: ``https://kubernetes.default.svc``
- ``namespace``: 배포 네임스페이스 (없으면 ``CreateNamespace=true``로 생성)
- ``environment``: dev/stage/prod 등 환경태그(이름/라벨 구성에 사용)
- ``path``: repo 내 차트/매니페스트 경로
- ``valuesFile``: 환경별 Helm values 파일
- ``revision``: 이미지 태그나 Helm 파라미터로 넘길 값(선택)
- **template.metadata.name**

``{{environment}}-{{app}}`` → ``dev-web``, ``prod-web`` 같은 Application 이름이 생성됩니다. 클러스터·환경 조합이 늘어도 이름 규칙이 흔들리지 않게 해주세요.
- **template.spec.source**
  - ``repoURL``, ``path``: GitOps 소스 위치
  - ``targetRevision``: 공통 브랜치/태그(요소별로 바꾸면 위 “goTemplate” 팁 참고)
  - ``helm.valueFiles``: 요소의 ``valuesFile``을 경로 패턴으로 연결  
예: ``helm/web/values-dev.yaml``
  - ``helm.parameters``: 환경·앱별로 자주 바뀌는 값을 간단히 덮어씁니다.
- **template.spec.destination**
  - ``server: '{{url}}`` 로 요소별 클러스터에 배포
  - ``namespace``도 요소별 값 사용
- **syncPolicy**
  - ``automated.prune/selfHeal``: Git과 동기화 자동화
  - ``CreateNamespace=true``: 대상 네임스페이스가 없으면 생성
  - ``retry``: 일시 장애에 재시도 백오프

## 적용 & 확인

```
# 생성
kubectl apply -n argocd -f appset.yaml

# 생성된 Application 목록
```



```
kubectl get applications -n argocd
# 또는
argocd app list

# 특정 앱 상태 보기
kubectl describe application dev-web -n argocd
argocd app get dev-web
```

## 자주 겪는 이슈와 빠른 체크리스트

### 1. **ApplicationSet** 컨트롤러 미설치

- ``kubectl get deploy -n argocd | grep applicationset`` 로 배포 확인.

### 2. 클러스터 **URL** 미등록

- ArgoCD에 대상 클러스터가 등록돼 있어야 합니다.  
``argocd cluster add <context>`` 후 ``destination.server``와 매칭되는지 점검.

### 3. 이름/라벨 길이·문자 제한

- ``metadata.name``은 253자 이하, 소문자 영문/숫자/하이픈 조합 권장.

### 4. **values** 파일 경로 오류

- ``helm.valueFiles``는 repo 기준 경로입니다. 오타, 디렉토리 구조 확인.

### 5. 브랜치/태그 혼용

- 요소별로 다르게 쓰려면 ``goTemplate: true`` + ``{{ default "HEAD" .revision }}`` 패턴이 깔끔.

## 확장 아이디어(필요 시)

- 네임스페이스 규칙화: ``namespace: '{{app}}-{{environment}}``
- 여러 앱 한 번에: elements에 ``app: web``, ``app: api`` 등 다수 행 추가
- **Helm 차트 공유 + values만 환경 분리**: 지금 템플릿이 정석
- **Matrix/Cluster generator**: 나중에 repo 스캔 또는 클러스터 목록과 곁집합으로 확장 가능

원하시면 지금 템플릿을 **사용자님의 실제 repo/경로 명**에 맞춰 바로 맞춤 수정까지 해서 드릴게요. 사용 중인 repo URL과 디렉토리 구조만 알려주시면 됩니다.