

선언형으로 만드는 멀티 클러스터

Cluster API와 App of Apps 패턴

Who am I

문지현

- 3년차 엔지니어
- 학부생때 해커톤에서 CI/CD를 경험 후 감동(?)을 받고 Cloud, DevOps에 흥미를 느낌

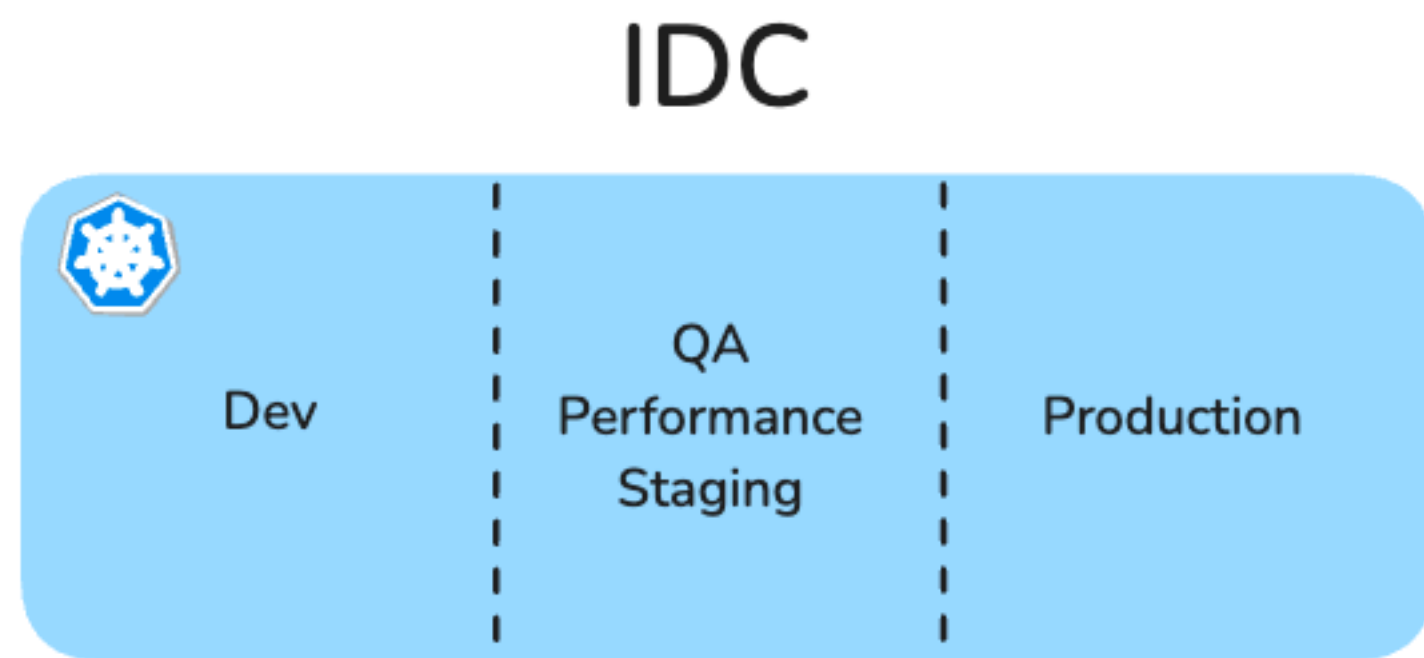
Career

- **Smilegate Megaport (2022.01 ~ 2024.07)**
플랫폼 엔지니어링: Stove 플랫폼 서비스의 클라우드, 쿠버네티스 운영 및 고도화
- **Kbank (2024.08 ~ now)**
SRE: 클라우드/쿠버네티스 기반의 혁신 서비스 고도화

Index

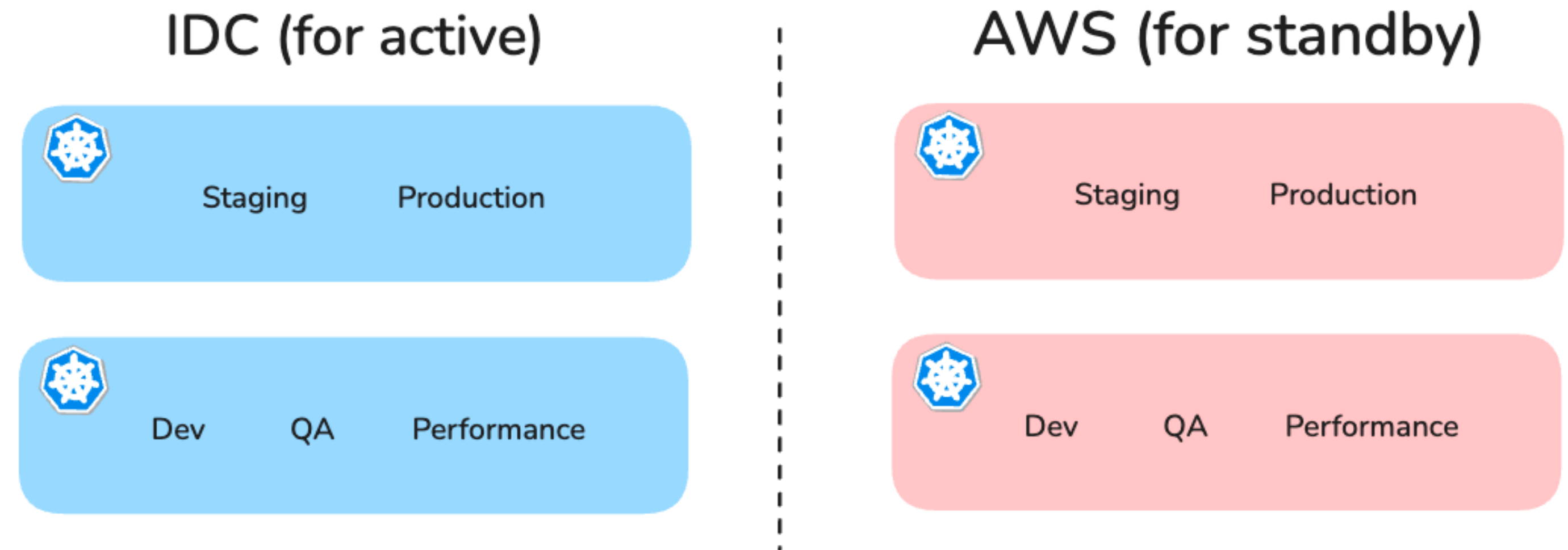
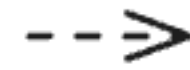
- **Background**
- **Cluster API 기반의 k8s 프로비저닝**
 - Cluster API → Concept 소개
 - IDC Workload → Cluster API를 통한 실제 세부 구성 소개
 - 정리
- **Helm으로 구현하는 App of Apps 패턴**
 - App of Apps 패턴 & ApplicationSet
 - ApplicationSet을 쓰지 않은 이유
 - Helm으로 구현하는 App of Apps 패턴 → Sub Chart와 kustomize 활용
 - 요약
- **Lesson Learned**

Background



AS-IS (2022)

단일 k8s에 모든 환경 서비스를 운영



TO-BE (2023)

멀티 클러스터 Active-Standby 구조의 DR 구성

멀티 클러스터 환경 만들기

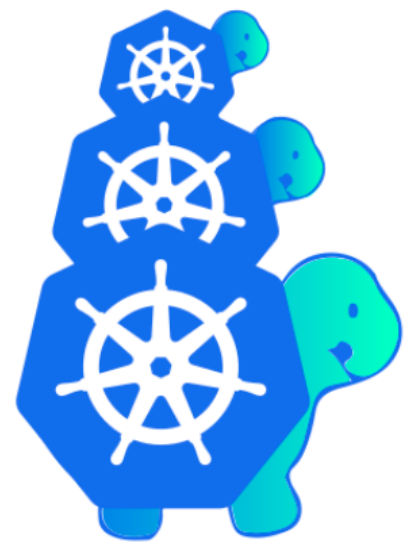
가장 먼저 필요한 일들

1. 클러스터 새로 만들기

→ Cluster API 기반의 프로비저닝

2. 각 클러스터에 필요한 컴포넌트 / 서비스 배포

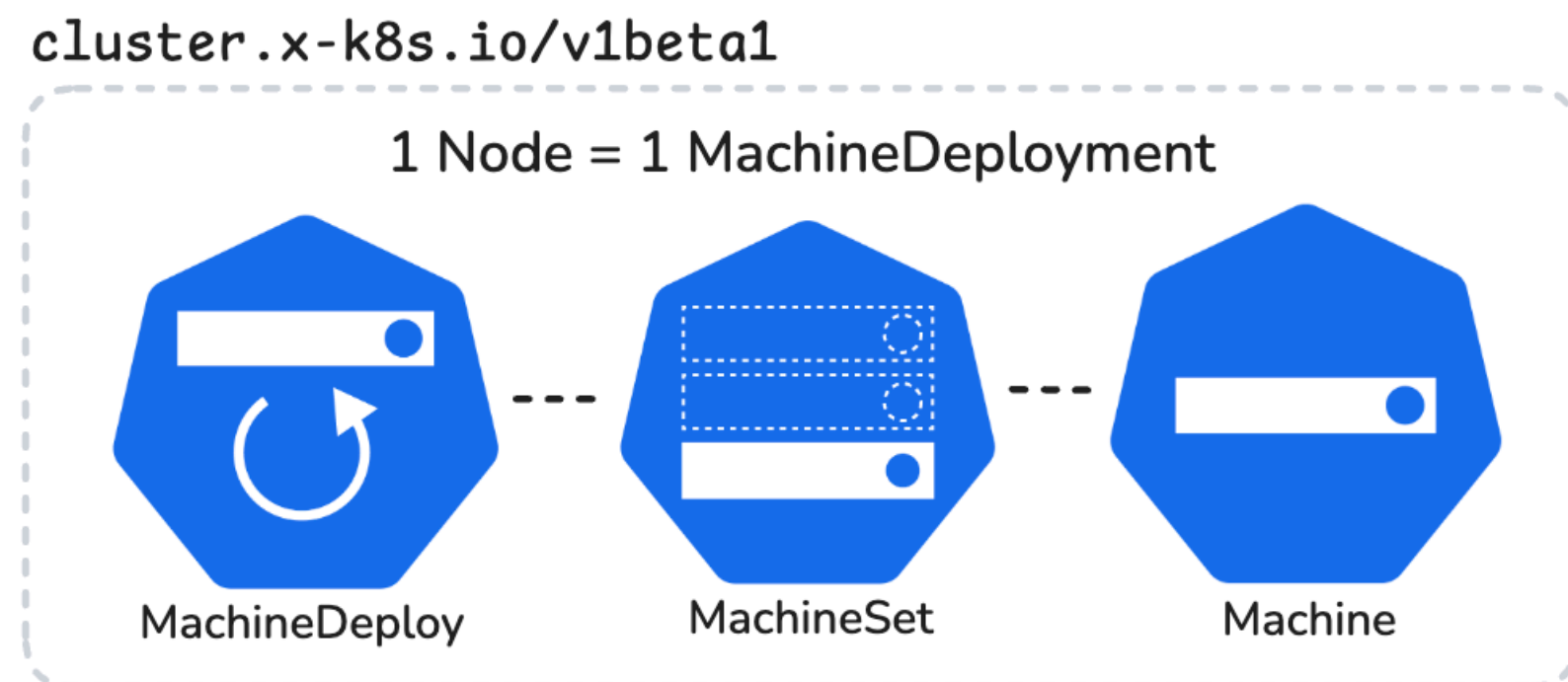
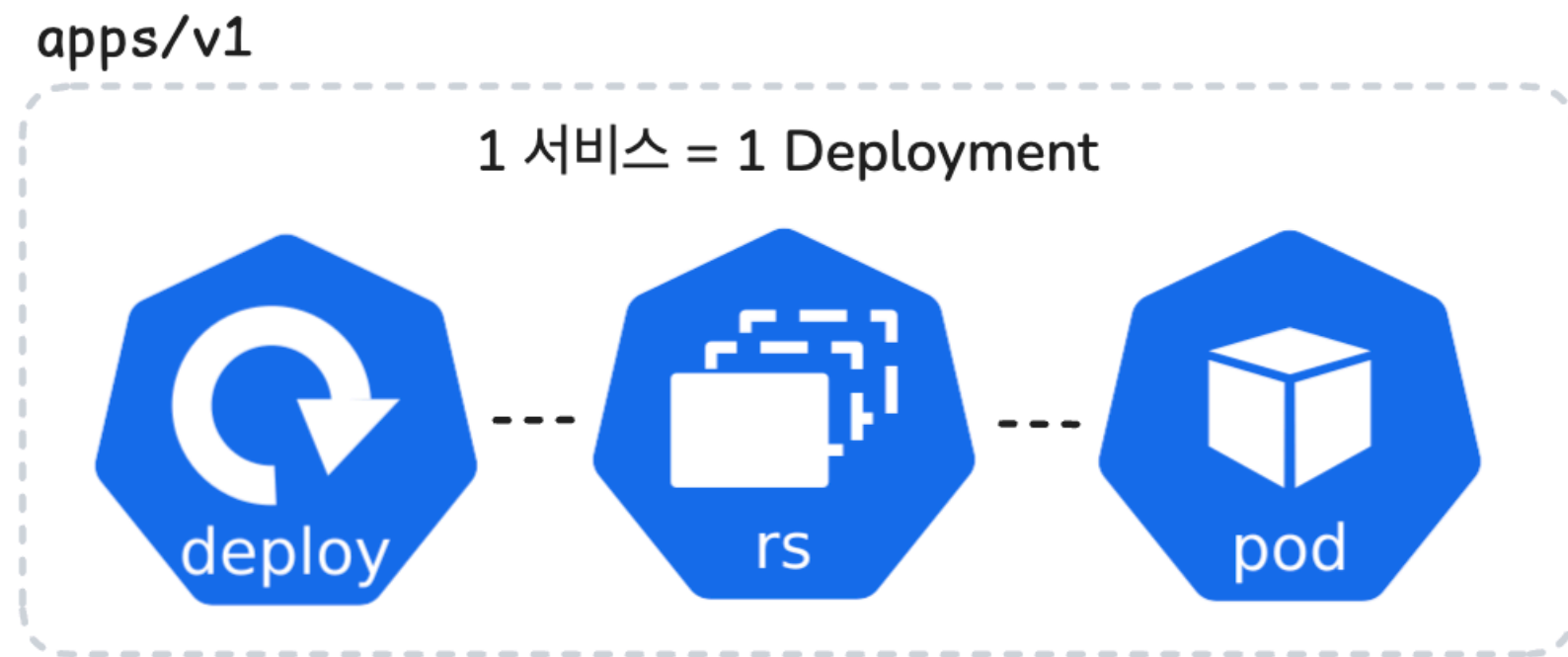
→ Helm으로 구현하는 App of Apps 패턴



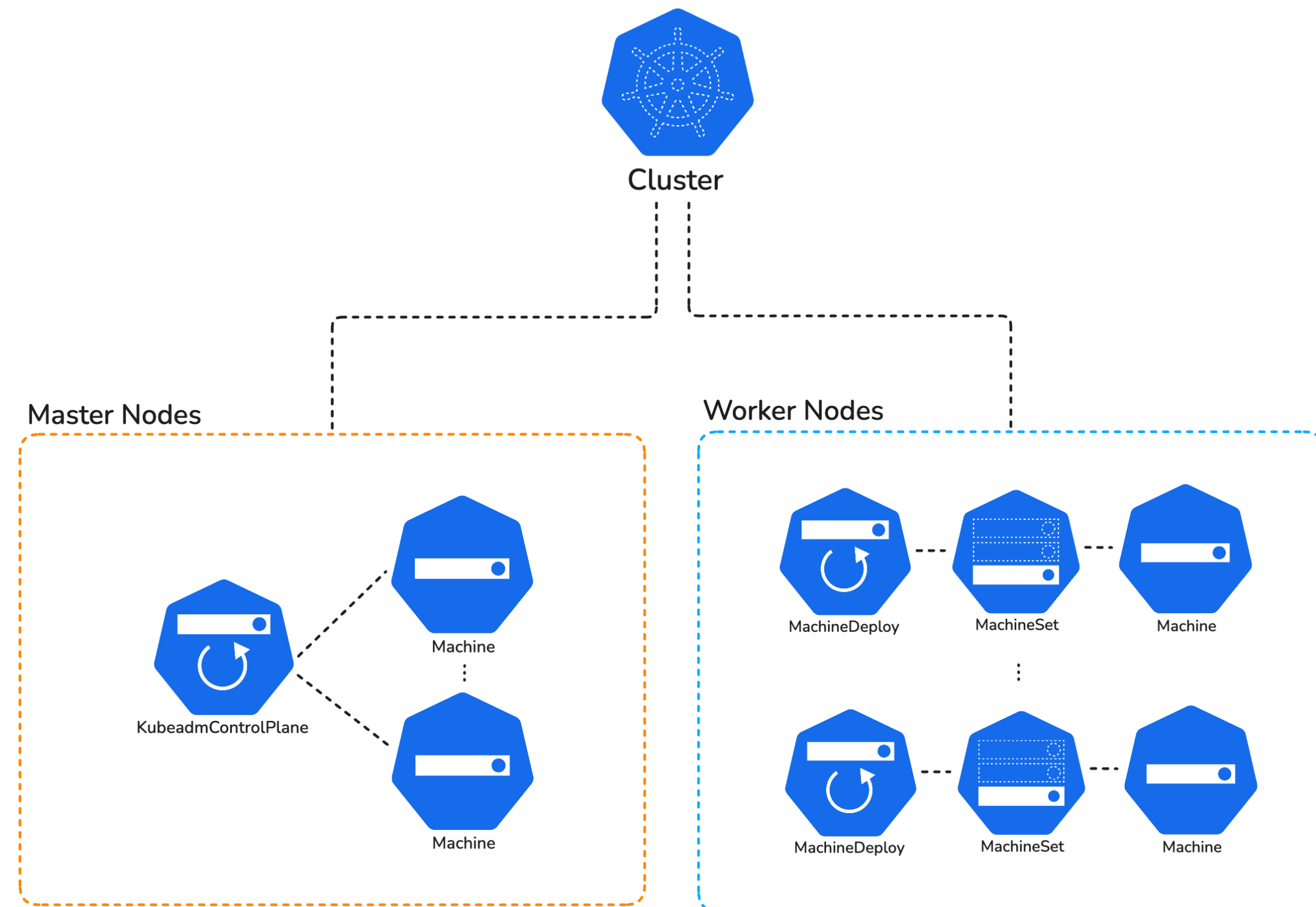
Cluster API 기반의 k8s 프로비저닝

Cluster API

쿠버네티스 라이프사이클 관리를 쿠버네티스 스타일로



노드를 마치 파드처럼...



클러스터를 추상화한 Custom Resource

다양한 조건과 인프라 환경을 지원

리소스별로 사용할 Provider를 참조한다.

Bootstrap

- Amazon Elastic Kubernetes Service (EKS)
- Kubeadm
- MicroK8s
- Oracle Cloud Native Environment (OCNE)
- Talos
- K3s
- k0smotron/k0s

Control Plane

- Kubeadm
- MicroK8s
- Nested
- Oracle Cloud Native Environment (OCNE)
- Talos
- Kamaji
- K3s
- k0smotron/k0s

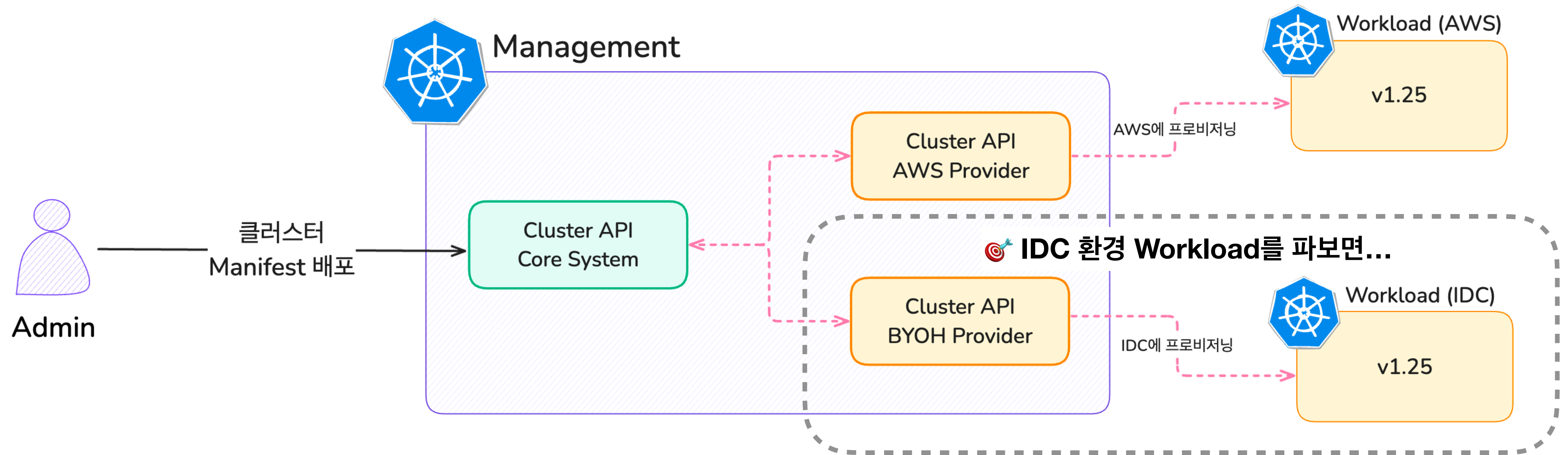
Infrastructure

- Akamai (Linode)
- AWS
- Azure
- Azure Stack HCI
- Bring Your Own Host (BYOH)
- CloudStack
- CoxEdge
- DigitalOcean
- Equinix Metal (formerly Packet)
- Google Cloud Platform (GCP)
- Hetzner
- Hivelocity
- IBM Cloud
- IONOS Cloud
- KubeKey
- KubeVirt
- MAAS
- Metal3
- Microvm
- Nested
- Nutanix
- Oracle Cloud Infrastructure (OCI)
- OpenStack
- Outscale
- Proxmox

(밑에 더 있음)

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: capim-quickstart
spec:
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: capim-quickstart-control-plane
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1alpha1
    kind: InMemoryCluster
    name: capim-quickstart
---
apiVersion: infrastructure.cluster.x-k8s.io/v1alpha1
kind: InMemoryCluster
metadata:
  name: capim-quickstart
---
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: capim-quickstart-control-plane
spec:
  kubeadmConfigSpec: ...
  machineTemplate:
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1alpha1
      kind: InMemoryMachineTemplate
      name: capim-quickstart-control-plane
    replicas: 1
    version: v1.27.3
---
apiVersion: infrastructure.cluster.x-k8s.io/v1alpha1
kind: InMemoryMachineTemplate
metadata:
  name: capim-quickstart-control-plane
spec: ...
```


Management 클러스터에서 Workload 클러스터를 생성/관리



IDC 환경의 Workload

BYOH (Bring Your Own Host) Provider

IDC 환경에서, VM이 아니라 Baremetal로 쿠버네티스를 구성해야 함.

OS 배포부터 수행하는 Baremetal Provider도 고려했으나 PoC때 사내 환경과 안맞는 부분이 확인됨.

OS가 배포된 Baremetal 서버를 관리하는 Provider가 있을까?

→ **BYOH (전용 agent를 서버에 Daemon 형식으로 설치하여 Custom Resource로 관리)**

What is Cluster API Provider BYOH

[Cluster API](#) brings declarative, Kubernetes-style APIs to cluster creation, configuration and management.

BYOH is a Cluster API Infrastructure Provider for already-provisioned hosts running Linux. This provider allows operators to adopt Cluster API for deploying and managing kubernetes nodes without also having to adopt a specific infrastructure service. This enables users to decouple kubernetes node provisioning from host and infrastructure provisioning.

byoh-agent

agent가 수행하는 역할

BYOH Provider는 전용 agent를 기반으로 노드용 서버들을 관리하며, agent는 총 3개의 역할을 수행할 수 있다.

- **Registration:** agent를 통해 서버를 Management 클러스터의 Custom Resource로 등록
- **Setup:** OCI 레지스트리를 통해 서버에 k8s 컴포넌트(kubelet, containerd 등) 설치/삭제 관리
- **Bootstrap:** kubeadm을 사용하여 서버를 k8s 노드로 변경 (kubeadm init, join과 같음)

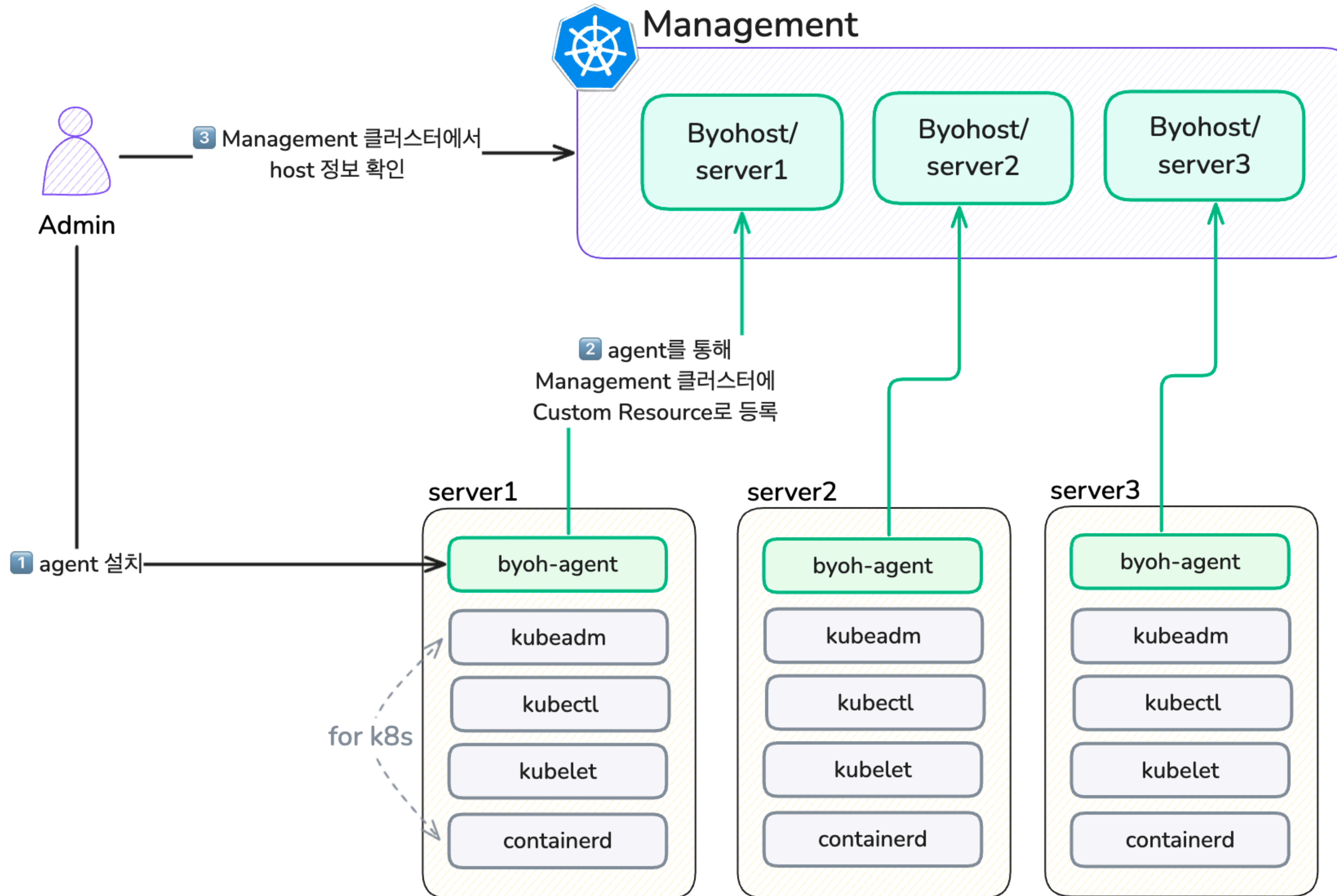
BYOH Agent

BYOH agent is a binary that runs on the hosts and its responsibility include -

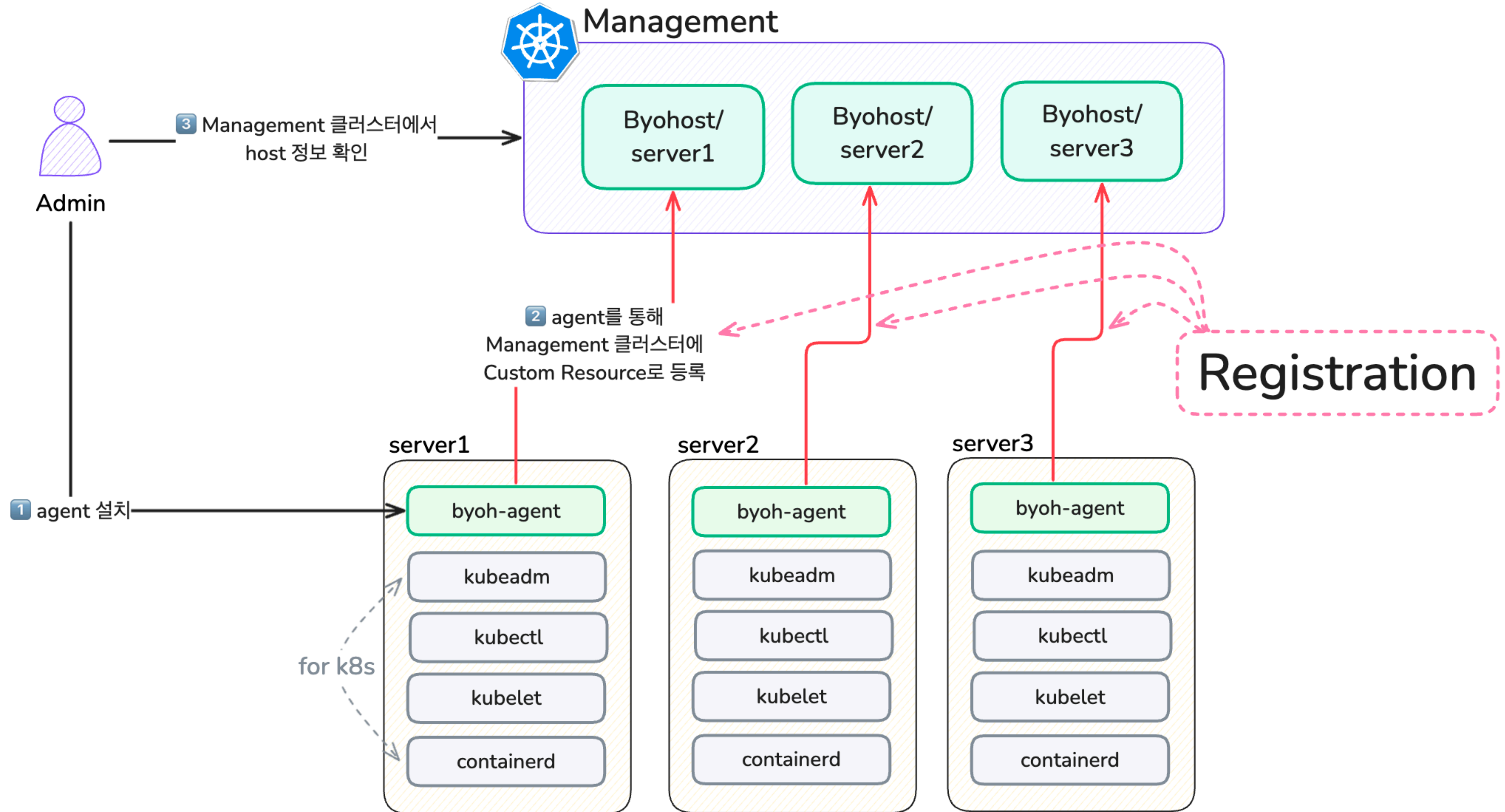
1. Registration - Register the host to the cluster capacity pool
2. Setup - Install / Uninstall Kubernetes components on the host
3. Bootstrap - Convert the host to a Kubernetes node using kubeadm



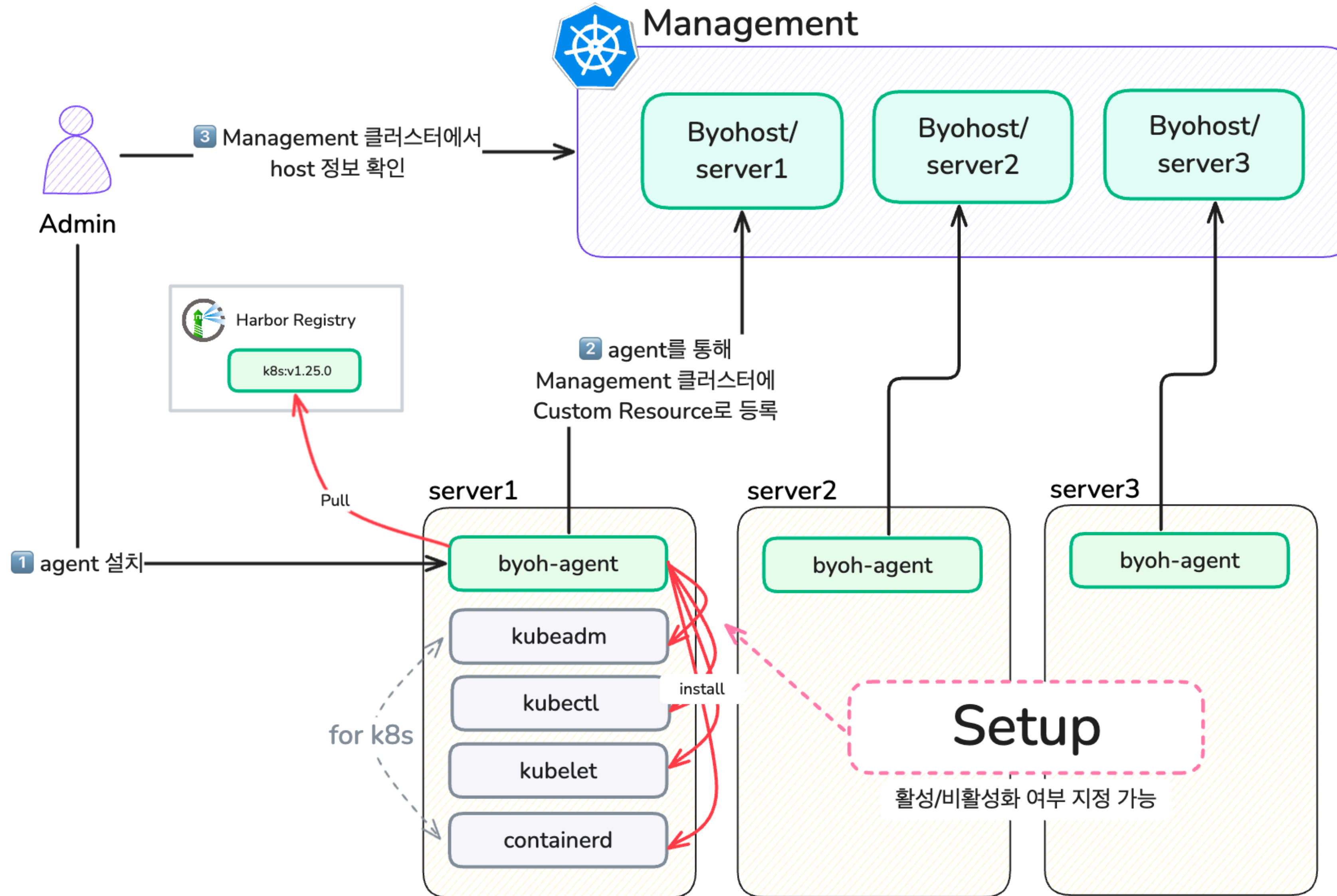
Agent 기반 Host 등록/관리 flow



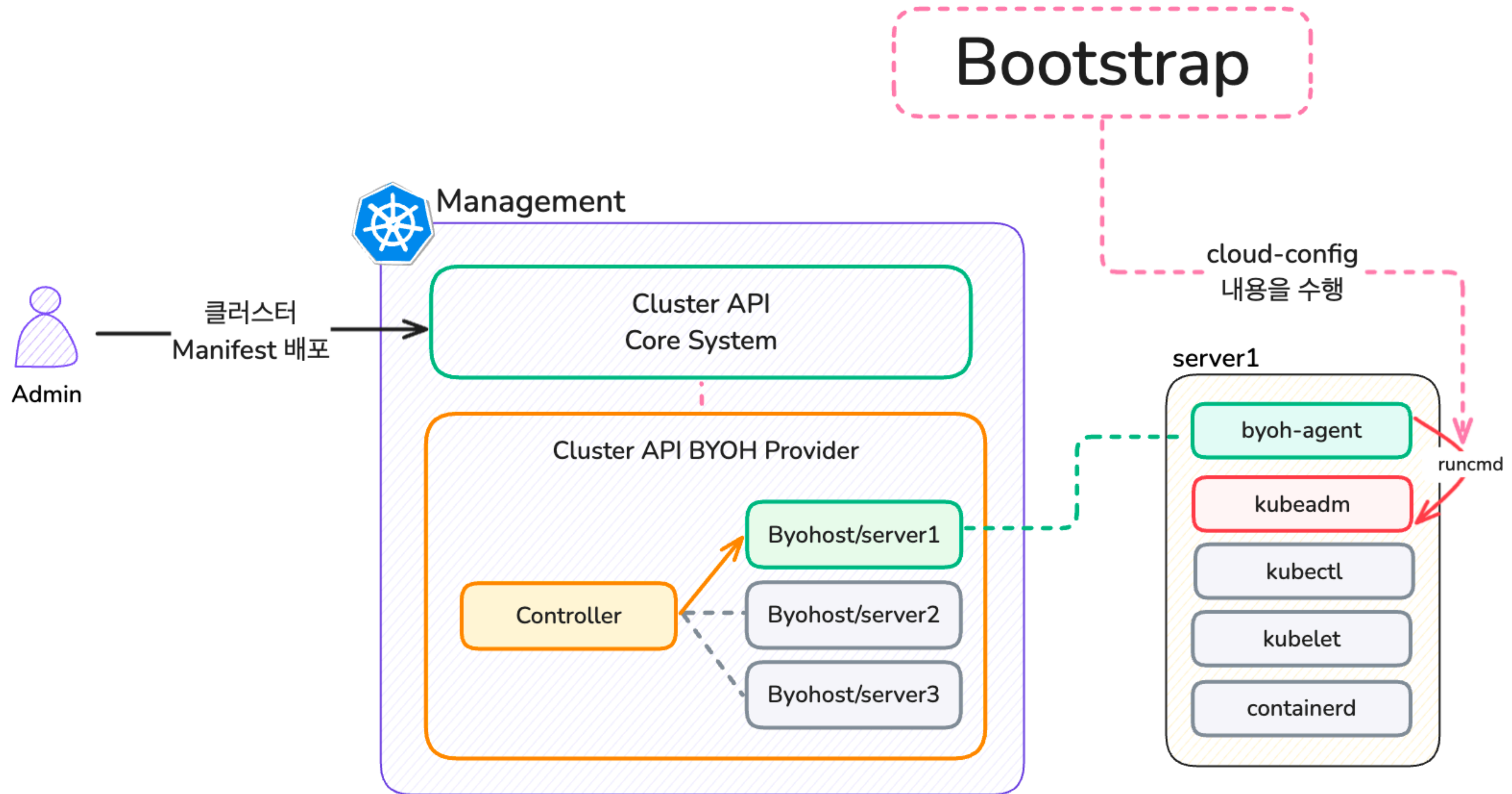
Registration: Custom Resource로 등록



Setup: k8s 컴포넌트 설치



Bootstrap: 노드 투입을 위한 작업 / 커맨드 실행



장점은 차치하고... 솔직히 말해서 복잡하다.

Cluster API의 어려운 점들

쿠버네티스 외부의 자원을 쿠버네티스로 관리하는 구조라 진입장벽이 있음

- 도입, 운영, 트러블슈팅 과정에서 쿠버네티스 오퍼레이터 패턴에 대한 이해가 필요 (Code 레벨까지 볼 일도..)

Cluster API가 관리하는 영역과 그렇지 않은 영역이 헷갈림 (즉 구분을 명확히 해야 함)

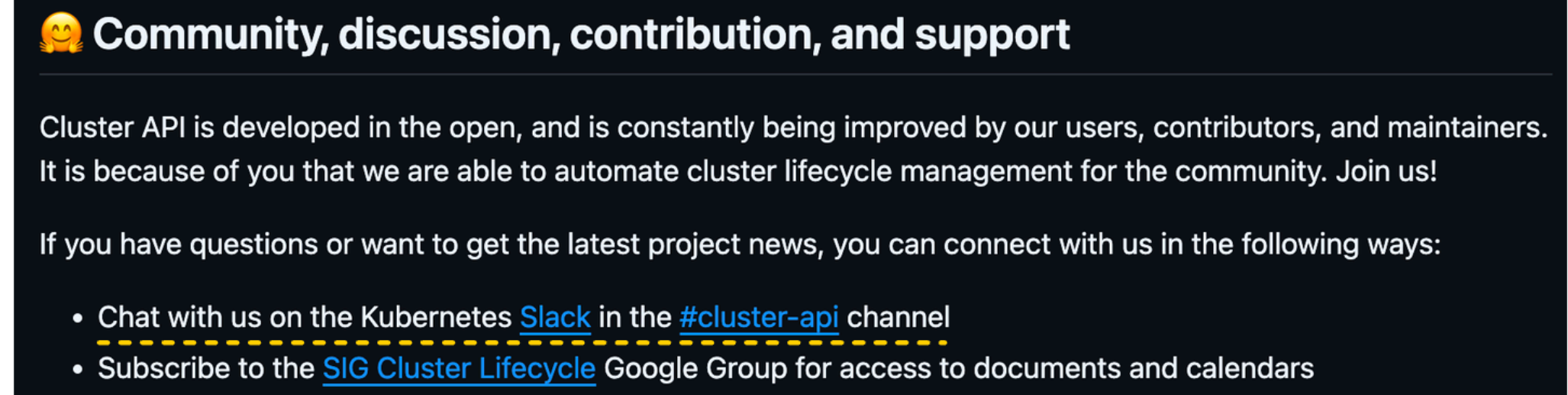
- Cluster API가 뭔가 이것저것 다 해주는 것은 아님. 그냥 k8s/인프라 관련한 복잡한 내용과 절차를 추상화시킨 것에 불과

진입장벽을 낮추기 위해...

커뮤니티와 Docs를 활용하자

✓ 공식 커뮤니티 Slack 채널

- Provider별로 Slack 채널이 존재함 (Github Repo README 참고)
- 질문, 새로운 정보 확인 등 다방면으로 활용하면 좋음 (추천)



Community, discussion, contribution, and support

Cluster API is developed in the open, and is constantly being improved by our users, contributors, and maintainers. It is because of you that we are able to automate cluster lifecycle management for the community. Join us!

If you have questions or want to get the latest project news, you can connect with us in the following ways:

- Chat with us on the Kubernetes [Slack](#) in the [#cluster-api](#) channel
- Subscribe to the [SIG Cluster Lifecycle](#) Google Group for access to documents and calendars

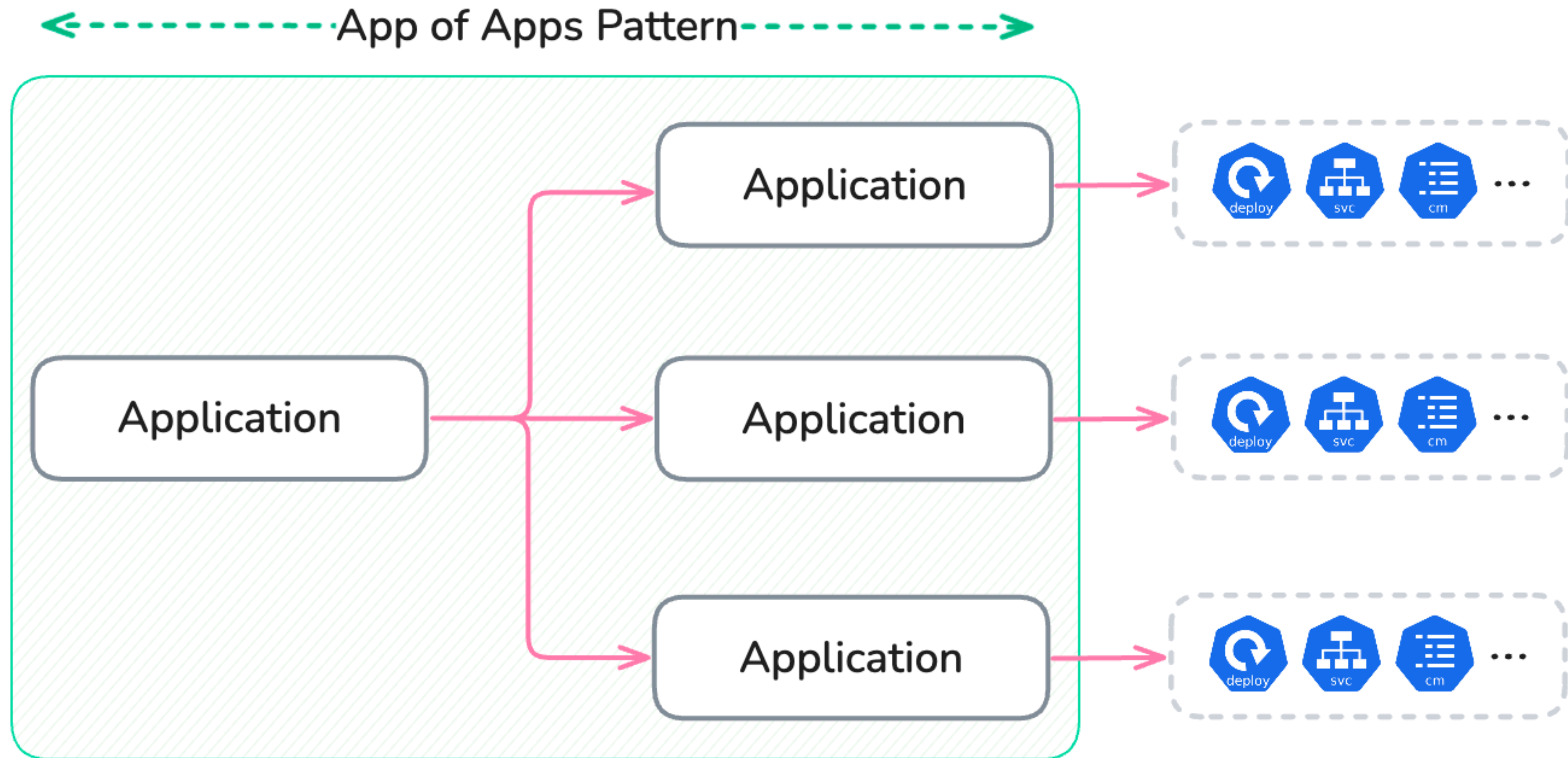
✓ Kubernetes Docs

- 생각보다 Cluster API Docs보다 Kubernetes Docs 볼 일이 훨씬 많다.
 - 쿠버네티스를 생성/변경하는 툴이다 보니 쿠버네티스 설정 자체를 봐야할 경우가 많음
- Control Plane 설정, kubeadm 설정, kubelet 설정 등이 필요할 때... 필요한 내용은 다 있음.
 - 바이너리 인자: <https://kubernetes.io/docs/reference/command-line-tools-reference/>
 - 컴포넌트 설정: <https://kubernetes.io/docs/reference/config-api/>
 - 기타 등등...

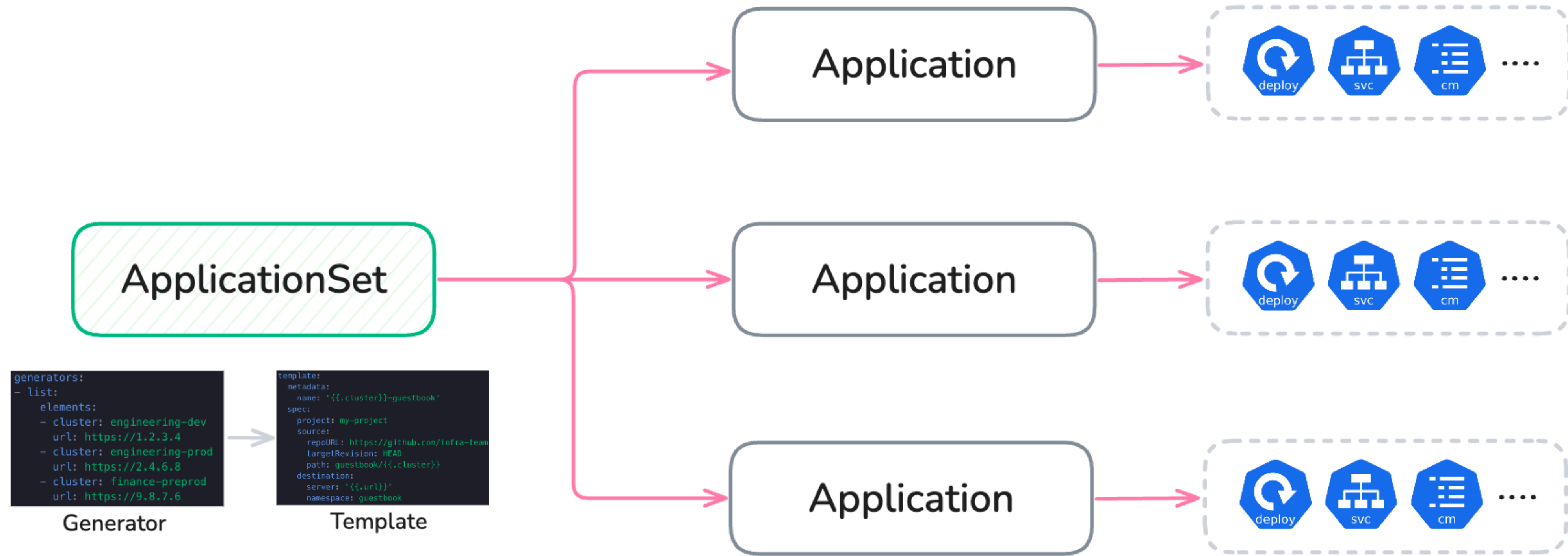


Helm으로 구현하는 App of Apps 패턴

App of Apps 패턴



ApplicationSet (=AppSet)



처음엔 ApplicationSet을 쓰다가 걷어냄

좋긴 하지만... 운영 상황에 잘 맞지 않는 점도 있었음 😓

운영 도중, App의 일부 설정 변경에 대한 관리가 어려움 (SyncPolicy 등)

- AppSet 리소스가 특정 App에 대한 Controller처럼 동작하며, 운영자가 변경한 설정을 원복시킴
 - ex) 급하게 App들의 auto-sync를 꺼야할 때? → UI에서 바로 바꾸면 안되고, AppSet 리소스의 템플릿을 바꿔야 함
- v2.9 이후부터는 개선되었지만, 일부 형식의 값은 아직 제한이 있음
 - 관련 issue: <https://github.com/argoproj/argo-cd/issues/9101>

환경별 Template 구성에 대한 복잡도 증가

- 총 7개의 환경, 4개의 클러스터, 4개의 Git Repo를 조합하는 조건을 template으로 만드니 내용이 꽤 복잡...
- 복잡한 template 규칙을 쓸 바엔 그냥 AppSet 말고 Application으로 관리하는게 낫지 않을까?

그런데, App of Apps 패턴

막상 만들려 하니...

Application 리소스?

현황은 보이긴 하지만 뒤죽박죽...


날것의 yaml들이 너무 많음...

이게 정말 최선일까...

```
! root.yaml x ... ! app-1.yaml x
local > app-of-apps-sample > ! root.yaml
1  apiVersion: argoproj.io/v1alpha1
2  kind: Application
3  metadata:
4    name: app-of-apps
5    namespace: argocd
6  spec:
7    destination:
8      server: https://kubernetes.default.svc
9    project: default
10   source:
11     path: resources
12     repoURL: https://github.com/app-of-apps/app-of-apps.git
13     targetRevision: main

! app-2.yaml x ! app-3.yaml x
local > app-of-apps-sample > ! app-2.yaml
1  apiVersion: argoproj.io/v1alpha1
2  kind: Application
3  metadata:
4    name: app-2
5    namespace: argocd
6  spec:
7    destination:
8      server: https://kubernetes-B
9    project: default
10   source:
11     path: kustomize
12     repoURL: https://github.com/app-2/app-2.git
13     targetRevision: main

local > app-of-apps-sample > ! app-3.yaml
1  apiVersion: argoproj.io/v1alpha1
2  kind: Application
3  metadata:
4    name: app-3
5    namespace: argocd
6  spec:
7    destination:
8      server: https://kubernetes-C
9    project: default
10   source:
11     path: kustomize
12     repoURL: https://github.com/app-3/app-3.git
13     targetRevision: main
```



Helm으로 구현하는 App of Apps 패턴

원본 Chart - ArgoCD Application 1개만 관리

오직 ArgoCD Application 1개만 관리하는 Chart

- 재사용성을 높일 수 있도록 최대한 모든 항목을 템플릿화
- 네이밍 컨벤션, 라벨 규칙 등은 template 내용에 미리 정의
→ 사람이 직접 작성할 때 오타/실수나기 쉬운 부분에 중점을 둔다
- 완성된 Chart는 Harbor 등 OCI Registry에 올려 재사용

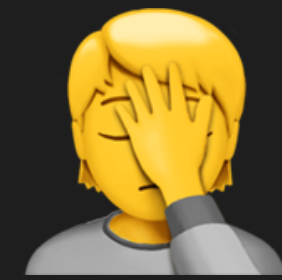
```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: app-1
  namespace: argocd
spec:
  destination:
    server: https://kubernetes-A
    project: default
  source:
    path: kustomize
    repoURL: https://github.com/app-1/app-1.git
    targetRevision: main
```

날것의 Application yaml



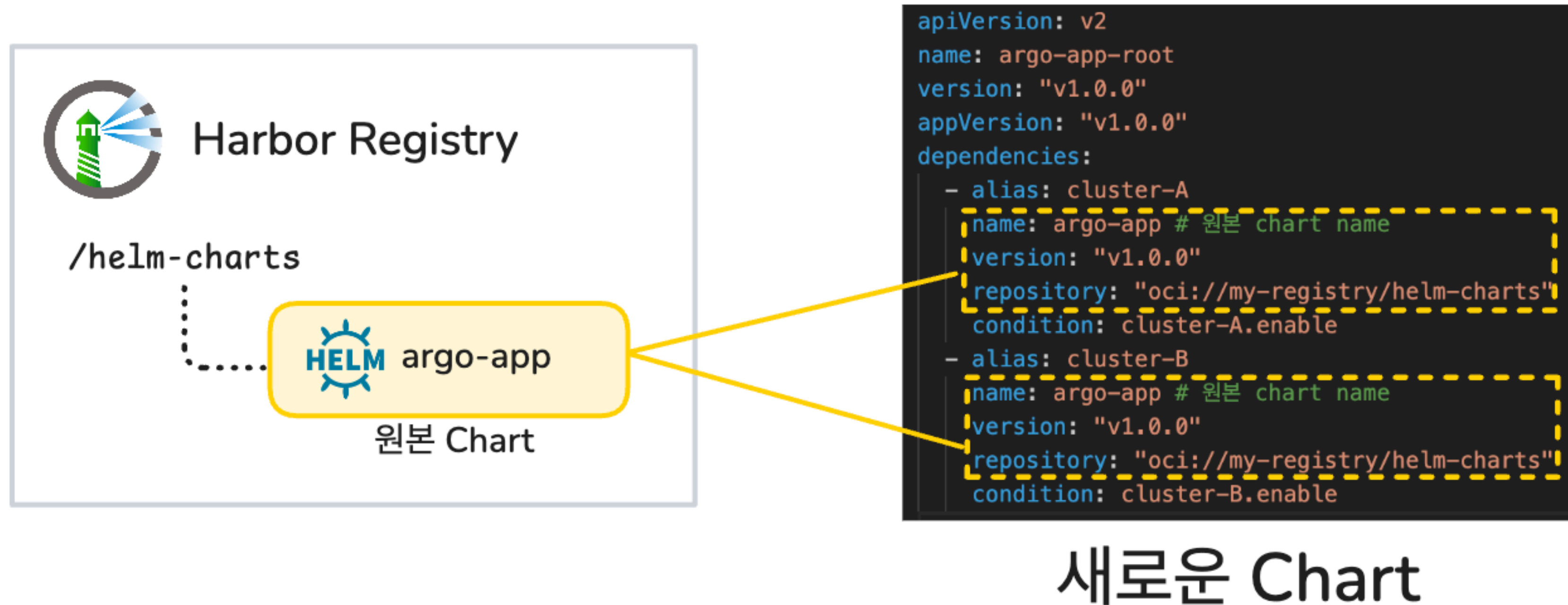
```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  {{- with .Values.additionalAnnotations }}
  annotations:
    {{- range $key, $value := . }}
    {{ $key }}: {{ $value | quote }}
    {{- end }}
  {{- end }}
  labels:
  {{- end }}
  {{- with .Values.additionalLabels }}
  {{- toYaml . | nindent 4 }}
  {{- end }}
  name: {{ .Values.name }}-{{ .Values.namespace | default "default" }}
  namespace: {{ .Values.namespace | default $argocdNamespace }}
  {{- with .Values.finalizers }}
  finalizers:
    {{- toYaml . | nindent 4 }}
  {{- end }}
spec:
  project: {{ .Values.project | default $project }}
  source:
    {{- with .Values.source | default $source }}
    repoURL: {{ .repoURL | default $source.repoURL }}
    targetRevision: {{ .targetRevision | default $source.targetRevision }}
    path: {{ .path | default $source.path }}
    {{- with .kustomize | default $source.kustomize }}
    kustomize:
      {{- end }}
    destination:
      {{- with .Values.destination | default $destination }}
      namespace: {{ .namespace | default $destination.namespace }}
      server: {{ .server | default $destination.server }}
      {{- end }}
    {{- with .Values.syncPolicy | default $syncPolicy }}
    syncPolicy:
      {{- end }}
    {{- with .Values.ignoreDifferences }}
    ignoreDifferences:
      {{- end }}
    {{- with .Values.info }}
    info:
      {{- end }}
  {{- end }}
```

Helm으로 하나씩 다 템플릿화!!



Helm으로 구현하는 App of Apps 패턴

Sub Chart 형식으로 원본 Chart 재사용 (1)



Helm으로 구현하는 App of Apps 패턴

Sub Chart 형식으로 원본 Chart 재활용 (2)

common-values.yaml

```
global:
  project: dev
  argocdNamespace: argocd
source:
  repoURL: "https://github.com/app-1/app-1.git"
  targetRevision: main
destination:
  namespace: dev
cluster-A:
  destination:
    server: https://cluster-A:6443
cluster-B:
  destination:
    server: https://cluster-B:6443
```

공통 사용할 values
(환경, repoURL, endpoint 등)

svc-a.yaml

```
global:
  name: svc-a
  labels: ...
source:
  path: dev/svc-a
cluster-A:
  enable: true
  additionalLabels: {}
  syncPolicy:
    automated:
      prune: true
cluster-B:
  enable: false
  additionalLabels: {}
  syncPolicy: {}
```

svc-b.yaml

```
global:
  name: svc-b
  labels: ...
source:
  path: dev/svc-b
cluster-A:
  enable: true
  additionalLabels: {}
  syncPolicy:
    automated:
      prune: true
cluster-B:
  enable: false
  additionalLabels: {}
  syncPolicy: {}
```

svc-c.yaml

```
global:
  name: svc-c
  labels: ...
source:
  path: dev/svc-c
cluster-A:
  enable: true
  additionalLabels: {}
  syncPolicy:
    automated:
      prune: true
cluster-B:
  enable: false
  additionalLabels: {}
  syncPolicy: {}
```

서비스 개별 values
(name, 라벨, syncPolicy 등)

Helm으로 구현하는 App of Apps 패턴

Sub Chart 형식으로 원본 Chart 재사용 (3)

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

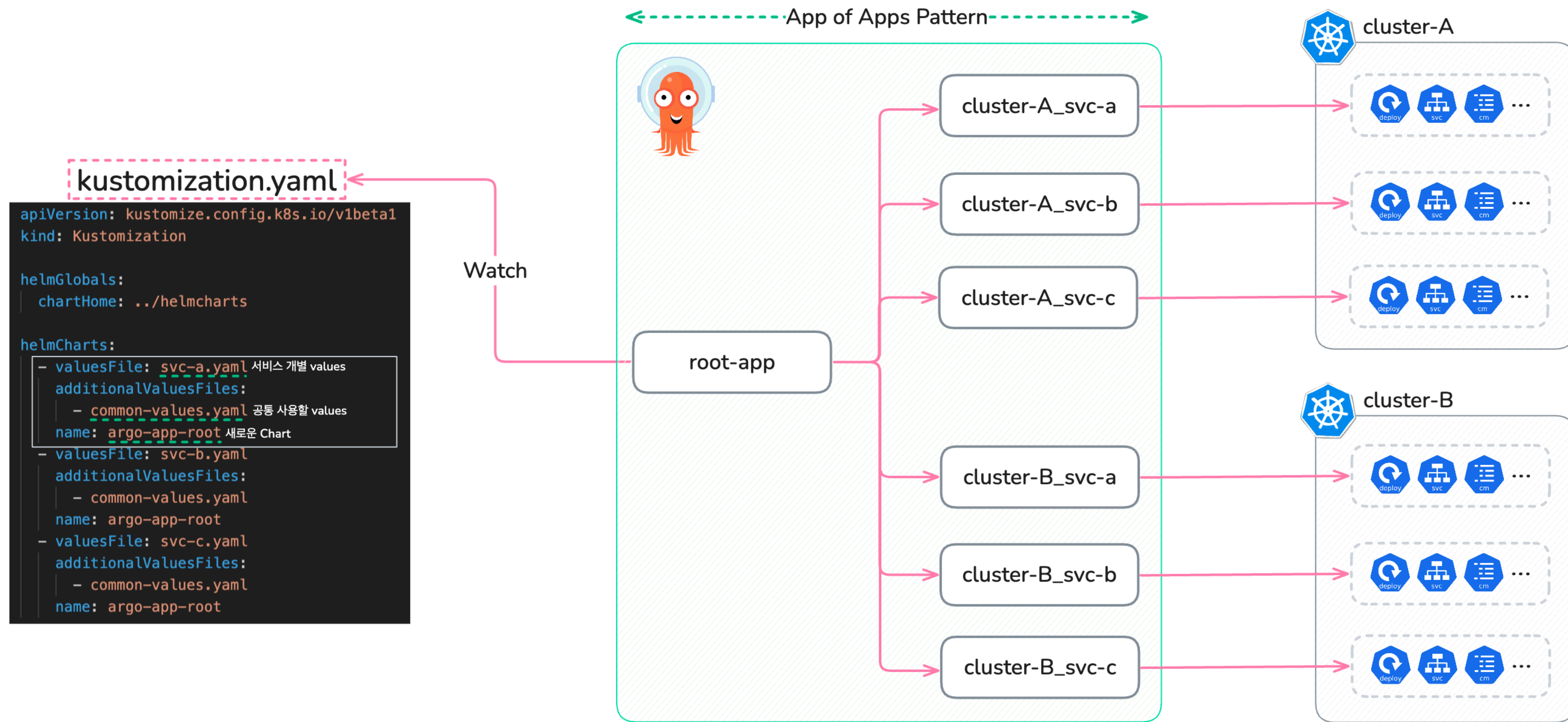
helmGlobals:
  chartHome: ../helmcharts

helmCharts:
  - valuesFile: svc-a.yaml 서비스 개별 values
    additionalValuesFiles:
      - common-values.yaml 공통 사용할 values
    name: argo-app-root 새로운 Chart
  - valuesFile: svc-b.yaml
    additionalValuesFiles:
      - common-values.yaml
    name: argo-app-root
  - valuesFile: svc-c.yaml
    additionalValuesFiles:
      - common-values.yaml
    name: argo-app-root
```

kustomization.yaml

Helm으로 구현하는 App of Apps 패턴

Sub Chart 형식으로 원본 Chart 재활용 (4)



Helm으로 구현하는 App of Apps 패턴

요약

Application 리소스를 helm과 kustomize를 혼합하여 배포

- **Helm:** Application의 정보를 템플릿화, 추상화, 표준화하기 위해 사용
- **Kustomize:** 수많은 Helm Chart들을 하나의 App으로 엮어주기 위해 사용

만들기까지 절차는 좀 있지만, 써보니 날것의 yaml보다 훨씬 좋음 👍

- monorepo 방식으로 배포 manifest를 관리하는 환경에 잘 맞았음
- 직관적인 추가 방법 (특정 위치에 특정 네이밍으로 values.yaml을 작성하고, kustomize에 추가하면 됨)
- 환경별 전체 Application을 일괄로 변경할 때 유리 (repoURL, 라벨, 클러스터, syncPolicy 등 일괄 변경)

Lesson Learned

자동화와 오퍼레이터는 강력한 툴이지만, 항상 나이스한건 아님

- Best Practice와 실제 운영 상황에는 간극이 있다 → 이런 간극에서 타협점을 찾는게 중요
- 현재 상황에 적합한 구조를 찾거나, 구조를 만들어 나가는 것이 중요

시행착오는 늘 있음.

이걸 토대로 빠르게 문제점을 고치고 더 나은 구조로 바꿔 나가는 것에 집중하자 🛠️

End Of Document