

Hands-on Crossplane with Argo CD: Multi-cloud Kubernetes Deployment

Report by: DucTruong Huynh

Abstract:

Focuses on a practical assignment report detailing the utilization of Crossplane and Argo CD for multi-cloud Kubernetes deployment. Cover the integration of these tools to provision infrastructure across Azure, AWS, GCP, and YC while adhering to GitOps principles for continuous delivery.

Table of content:

1. Before start	1
2. Building Kubernetes Cluster on Azure:	2
2.1 Prerequisite.....	2
2.2 Configuration:	3
2.3 Result.....	4
2. Building Kubernetes Cluster on AWS:	5
2.1 Prerequisite.....	5
2.2 Configuration:	7
2.3 Result.....	8
3. Building Kubernetes Cluster on GCP:.....	8
3.1 Prerequisite.....	8
3.2 Configuration:	9
3.3 Result:	10
4. Building Kubernetes Cluster on YC:.....	11
4.1 Prerequisite.....	11
4.2 Configuration:	13
4.3 Result:	14

1. Before start

- **Kubernetes Cluster:** Ensure you have a Kubernetes cluster deployed on your preferred cloud provider (Azure, AWS, GCP) or on-premises.
- CLI: Kubectl, helm
- **Argo CD Installation:**

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

Install Crossplane in ArgoCD:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: crossplane
  namespace: argocd
  finalizers: []
spec:
  destination:
    name: ''
    namespace: crossplane-system
    server: 'https://kubernetes.default.svc'
  source:
    path: ''
    repoURL: 'https://charts.crossplane.io/stable'
    targetRevision: 1.15.2
    chart: crossplane
  sources: []
  project: default
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
```

Then manage it such as another ArgoCD app

2. Building Kubernetes Cluster on Azure:

2.1 Prerequisite

To authenticate with crossplane, you need to provide a secret which will allow you authenticate to Azure. To do that, you can authen via AzCli:

```
az ad sp create-for-rbac
```

```
--sdk-auth
```

```
--role Owner
```

```
--scopes <subscriptionId>
```

Then store it as a json file, for example `azure-credentials.json`

Then, create secrets with kubectl, this one will be used to mapping to ProviderRef later:

for example:

```
kubectl create secret generic azure-secret -n crossplane-system --from-file=creds=azure-credentials.json
```

Then, later you can create Provider config:

```
:
apiVersion: azure.upbound.io/v1beta1
kind: ProviderConfig
metadata:
  name: azure-provider
  annotations:
    argocd.argoproj.io/sync-wave: "2"
    argocd.argoproj.io/sync-options: "SkipDryRunOnMissingResource=true"
spec:
  credentials:
    source: Secret
    secretRef:
      namespace: crossplane-system
      name: azure-secrets
      key: creds
```

2.2 Configuration:

Provider:

```
apiVersion: pkg.crossplane.io/v1
kind: Provider
metadata:
  name: provider-azure-network
  annotations:
    argocd.argoproj.io/sync-wave: "0"
    argocd.argoproj.io/sync-options: "SkipDryRunOnMissingResource=true"
spec:
  package: xpkg.upbound.io/upbound/provider-azure-network:v0.42.1
```

And

```
apiVersion: pkg.crossplane.io/v1
kind: Provider
metadata:
  name: provider-azure-containerservice
  annotations:
    argocd.argoproj.io/sync-wave: "6"
    argocd.argoproj.io/sync-options: "SkipDryRunOnMissingResource=true"
spec:
  package: xpkg.upbound.io/upbound/provider-azure-containerservice:v1.0.1
```

Reference, read further:

<https://marketplace.upbound.io/providers/upbound/provider-azure-network/v1.1.0>

<https://marketplace.upbound.io/providers/upbound/provider-azure-containerservice/v1.1.0>

Application:

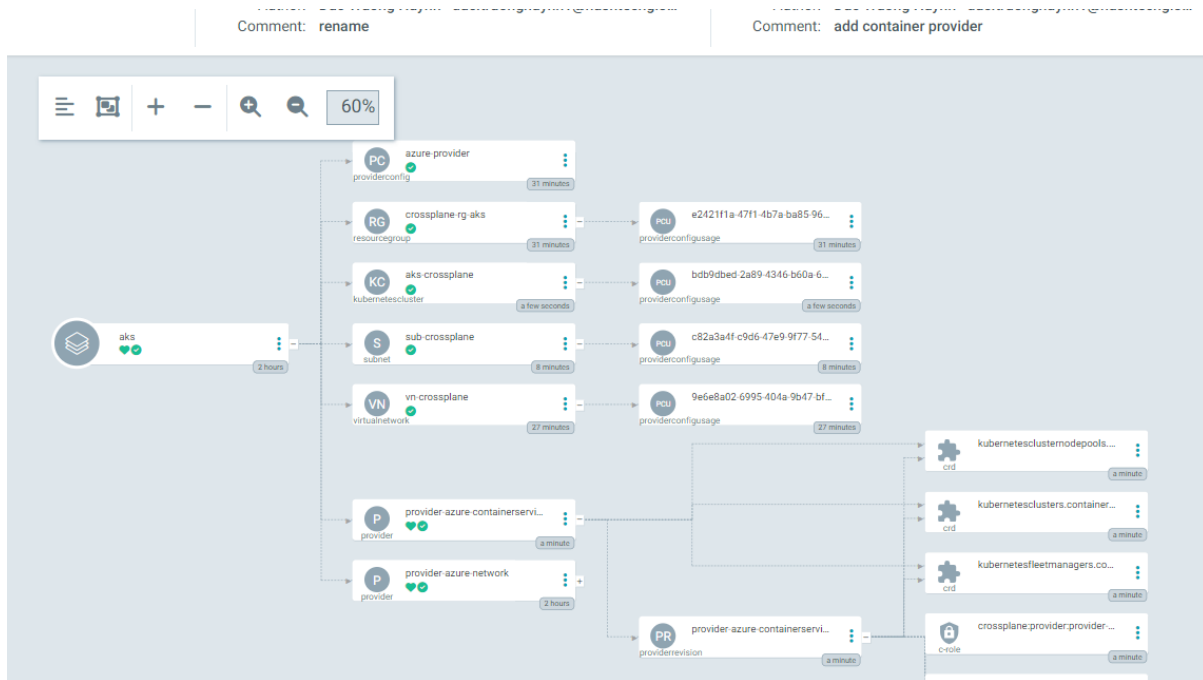
Reference to github: <https://github.com/huynhduc0/argo-demo/tree/master/aks/crossplane>

With application:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: aks
  namespace: argocd
spec:
  destination:
    name: ''
    namespace: dev
    server: 'https://kubernetes.default.svc'
  source:
    path: aks/crossplane
    repoURL: 'https://github.com/huynhduc0'
    targetRevision: HEAD
  sources: []
  project: default
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
```

2.3 Result

After run, we can see our AKS infrastructure as application struct below:



And Resource created on Azure:

crossplane-rg-aks Resource group

Search + Create Manage view Delete resource group Refresh Export to CSV Open query Assign tags ...

Overview

- Activity log
- Access control (IAM)
- Tags
- Resource visualizer
- Events

Settings

- Deployments
- Security
- Deployment stacks
- Policies
- Properties
- Locks

Cost Management

- Cost analysis
- Cost alerts (preview)

Essentials [JSON View](#)

Resources Recommendations

Filter for any field... Type equals **all** Location equals **all** Add filter

Showing 1 to 2 of 2 records. ☐ Show hidden types No grouping List view

Name	Type	Location
aks-crossplane	Kubernetes service	Central US
vn-crossplane	Virtual network	Central US

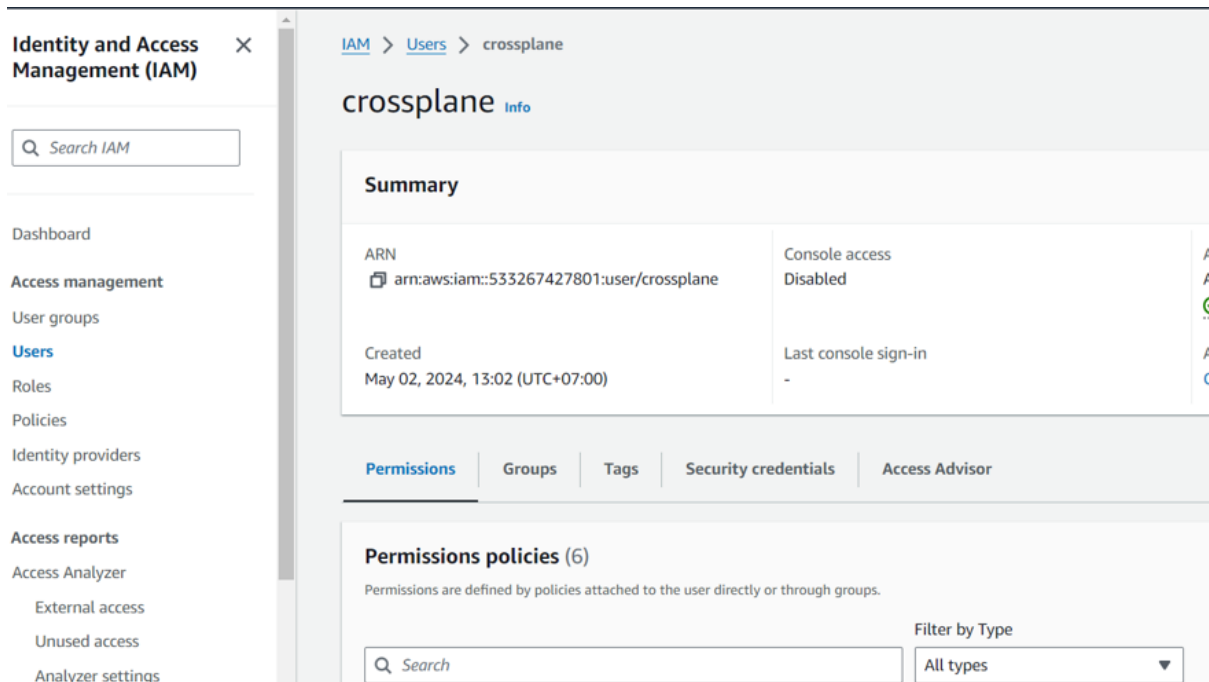
< Previous Page 1 of 1 Next >

[Give feedback](#)

2. Building Kubernetes Cluster on AWS:

2.1 Prerequisite

Same to Azure, we also need create a secrets, however, json format will not be use. Login to your AWS account and get AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY, grant nessessary permission such as EKS, VPC...



then create a file, such as `aws-credentials.txt`:

`AWS_ACCESS_KEY_ID=<your key>`

`AWS_SECRET_ACCESS_KEY= <your key>`

then, create with `kubectl`:

`kubectl create secret generic aws-secret -n crossplane-system --from-file=creds=aws-credentials.txt`

And your provider can be:

```
apiVersion: aws.upbound.io/v1beta1
kind: ProviderConfig
metadata:
  name: aws-provider
  annotations:
    argocd.argoproj.io/sync-wave: "1"
    argocd.argoproj.io/sync-options: "SkipDryRunOnMissingResource=true"
spec:
  credentials:
    source: Secret
    secretRef:
      namespace: crossplane-system
      name: aws-creds
      key: creds
```

2.2 Configuration:

Provider needed, use also use another once for each service, or for all:

```
apiVersion: pkg.crossplane.io/v1
kind: Configuration
metadata:
  name: configuration-aws-eks
  annotations:
    argocd.argoproj.io/sync-wave: "0"
    argocd.argoproj.io/sync-options: "SkipDryRunOnMissingResource=true"
spec:
  package: xpkg.upbound.io/upbound/configuration-aws-eks:v0.7.0
```

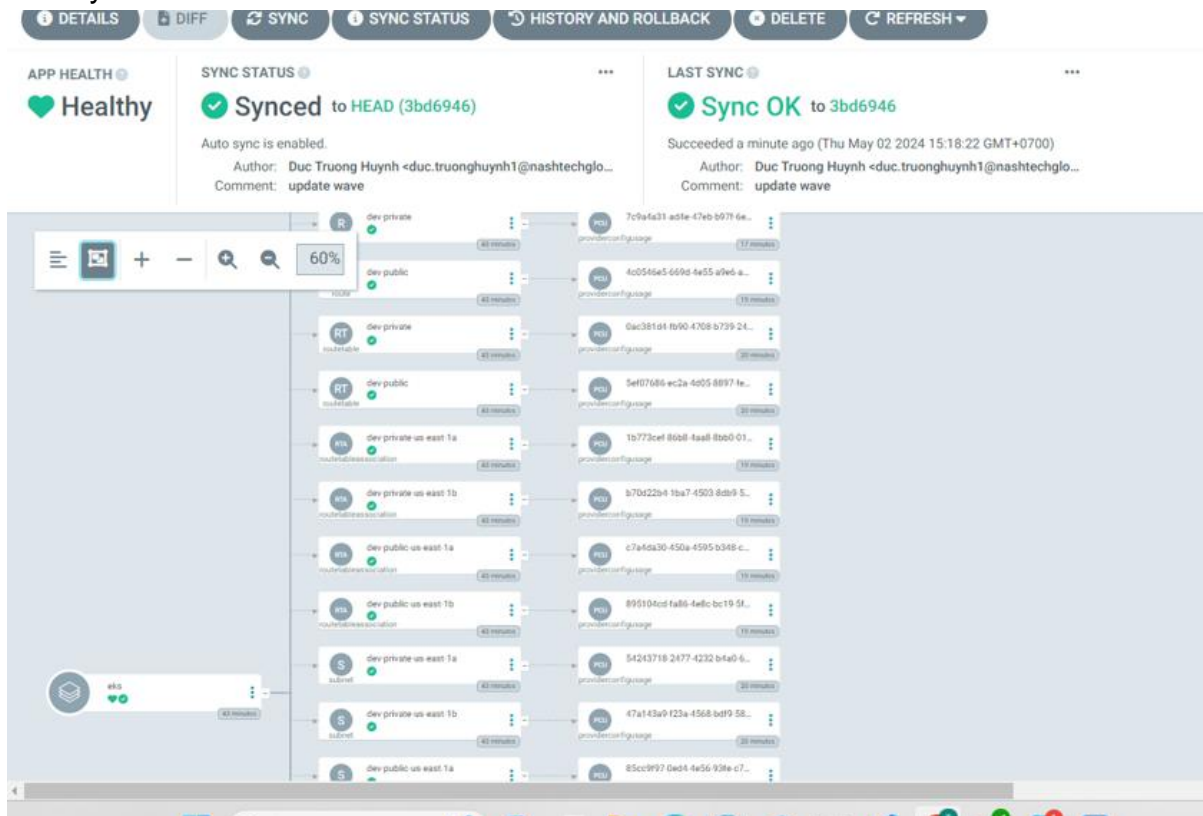
Reference to github: <https://github.com/huynhduc0/argo-demo/tree/master/eks/crossplane>

With application:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: eks
  namespace: argocd
spec:
  destination:
    name: ''
    namespace: dev
    server: 'https://kubernetes.default.svc'
  source:
    path: eks/crossplane
    repoURL: 'https://github.com/huynhduc0'
    targetRevision: HEAD
  sources: []
  project: default
  syncPolicy:
    automated:
      prune: true
      selfHeal: false
    syncOptions:
      - CreateNamespace=true
```

2.3 Result

Then you can see infrastructure:

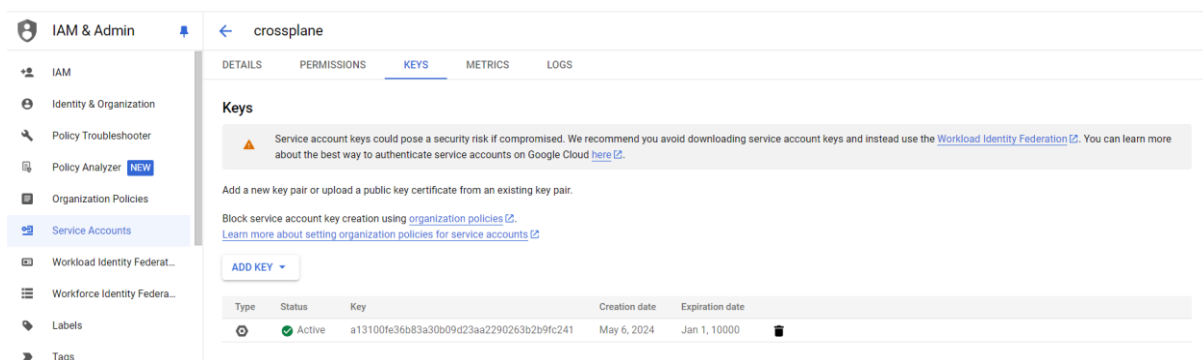


3. Building Kubernetes Cluster on GCP:

3.1 Prerequisite

Create a Service Account with GKE permission,

Then create key and download secrets:



Then run:

```
kubectl create secret generic gcp-secret -n crossplane-system --from-file=creds=<downloaded file.json>
```

Then, later you can create Provider config:


```

:
apiVersion: gcp.upbound.io/v1beta1
kind: ProviderConfig
metadata:
  name: {{.Values.provider.name}}
  annotations:
    argocd.argoproj.io/sync-wave: "1"
    argocd.argoproj.io/sync-options: "SkipDryRunOnMissingResource=true"
spec:
  projectID: arcane-geode-422508-p4
  credentials:
    source: Secret
    secretRef:
      namespace: crossplane-system
      name: gcp-secret
      key: creds

```

3.2 Configuration:

Provider:

```

apiVersion: pkg.crossplane.io/v1
kind: Provider
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "0"
    argocd.argoproj.io/sync-options: "SkipDryRunOnMissingResource=true"
  name: provider-gcp-compute
spec:
  package: xpkg.upbound.io/upbound/provider-gcp-compute:v1.1.0
---
apiVersion: pkg.crossplane.io/v1
kind: Provider
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "0"
    argocd.argoproj.io/sync-options: "SkipDryRunOnMissingResource=true"
  name: provider-gcp-container
spec:
  package: xpkg.upbound.io/upbound/provider-gcp-container:v1.1.0

```

Reference, read further:

<https://marketplace.upbound.io/providers/upbound/provider-gcp-compute/v1.1.0>

<https://marketplace.upbound.io/providers/upbound/provider-gcp-container/v1.1.0>

Application:

Reference to github:

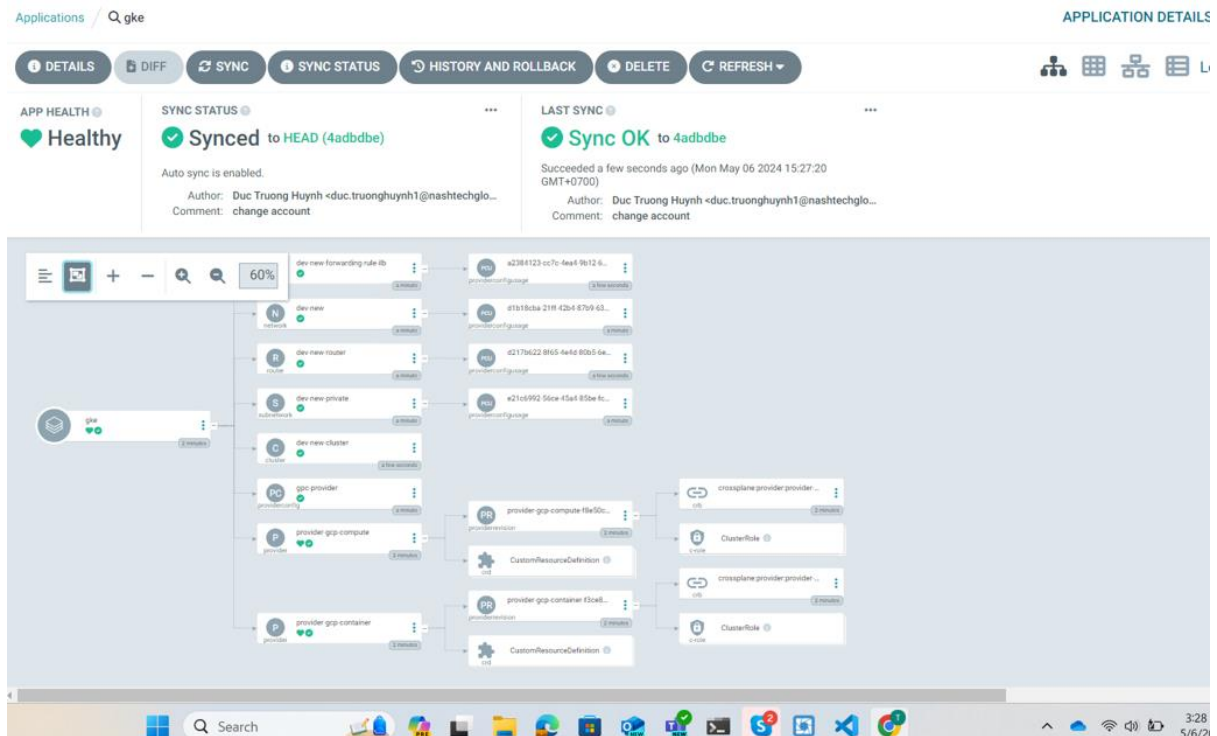
<https://github.com/huynhduc0/argo-demo/tree/master/gke/crossplane>

With application:

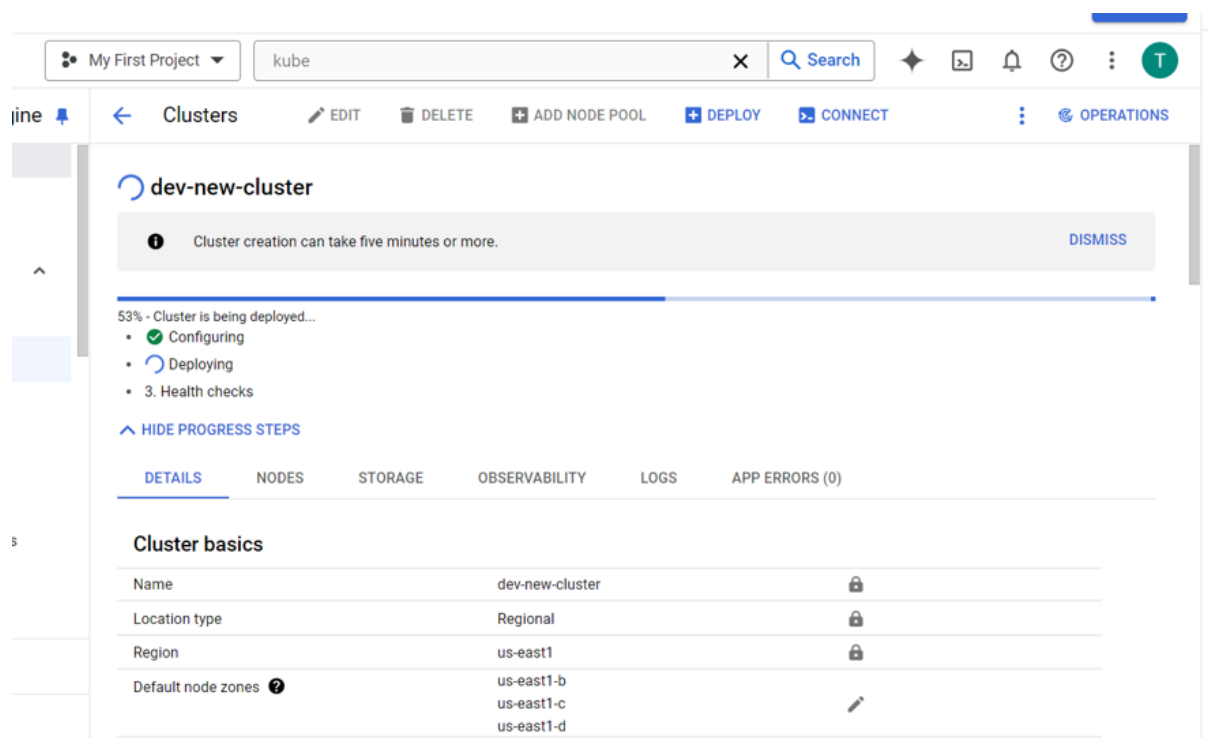
```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: gke
  namespace: argocd
spec:
  destination:
    name: ''
    namespace: dev3
    server: 'https://kubernetes.default.svc'
  source:
    path: gke/crossplane
    repoURL: 'https://github.com/huynhduc0/argo-demo/'
    targetRevision: HEAD
  helm:
    valueFiles:
      - values.yaml
  sources: []
  project: default
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
```

3.3 Result:

Then you can see infrastructure:



And on Google Cloud Console:



4. Building Kubernetes Cluster on YC:

4.1 Prerequisite

Create a Service Account with MKS (Manage Kubenertes Service) permission,

Then create key and download secrets:

Folder dashboard Service accounts Subscriptions Service notifications Access bindings Operations

Filter by name or ID

Name	Roles in folder	Date created	
crp	k8s.admin admin vpc.privateAdmin vpc.publicAdmin k8s.cluster-api.cluster-admin k8s.clusters.agent vpc.admin	05/07/2024, at 14:03	...

+ Create new key Edit account Delete account

Create static access key
For Object Storage, Message Queue,
and access to YDB via DocumentAPI

Create API key
For simplified authentication instead
of IAM tokens

Create authorized key
To request IAM tokens and YDB access
via the API

Then run:

```
kubectl create secret generic yc-secret -n crossplane-system --from-file=creds=<downloaded file.json>
```

Then, later you can create Provider config:

```
:
apiVersion: yandex-cloud.upjet.crossplane.io/v1beta1
kind: ProviderConfig
metadata:
  name: yc-provider
  annotations:
    argocd.argoproj.io/sync-wave: "1"
    argocd.argoproj.io/sync-options: "SkipDryRunOnMissingResource=true"
spec:
  credentials:
    source: Secret
    secretRef:
      namespace: crossplane-system
      name: yc-secret
      key: creds
```

4.2 Configuration:

Provider:

```
apiVersion: pkg.crossplane.io/v1
kind: Provider
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "0"
    argocd.argoproj.io/sync-options: "SkipDryRunOnMissingResource=true"
  name: provider-upjet-yc
spec:
  package: xpkg.upbound.io/tages/provider-upjet-yc:v1.1.0
```

Reference, read further:

<https://marketplace.upbound.io/providers/tages/provider-upjet-yc/v1.1.0>

Application:

Reference to github:

<https://github.com/huynhduc0/argo-demo/tree/master/yck/crossplane>

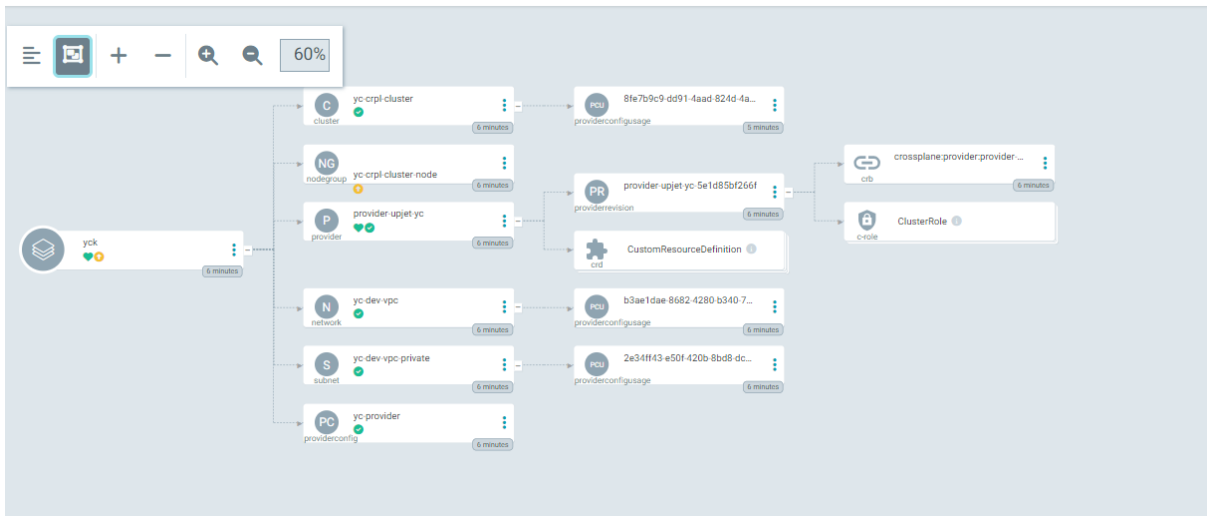
With application:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: yck
  namespace: argocd
spec:
  destination:
    name: ''
    namespace: dev3
    server: 'https://kubernetes.default.svc'
  source:
    path: yck/crossplane
    repoURL: 'https://github.com/huynhduc0/argo-demo/'
    targetRevision: HEAD
    helm:
      valueFiles:
        - values.yaml
  sources: []
  project: default
  syncPolicy:
```

```
automated:
  prune: true
  selfHeal: true
syncOptions:
  - CreateNamespace=true
```

4.3 Result:

Then you can see infrastructure:



And on Yandex Cloud Console:

[illegible]

yc-crpl-cluster

Cluster

Overview

Nodes manager

Workload

Storage

Network

Configuration

Access management

Namespaces

Events

Custom resources PREVIEW

Logs

Operations

Workload

Pods

Deployments

DaemonSets

StatefulSets

HPA

Jobs

Cron jobs

Name

Namespace

Controller type

Name	Namespace	Status	Node	Controller type	Controller name	Restarts	CPU	RAM	Storage
coredns-57b57bfc5b-zlb6z	kube-system	Pending	—	Deployment	coredns	0	100 / — mCPU	128 / 128 MB	— / — MB
kube-dns-autoscaler-bd7cc5977-6jdc9	kube-system	Pending	—	Deployment	kube-dns-autoscaler	0	20 / — mCPU	10 / — MB	— / — MB
metrics-server-54cb698b7f-22t4p	kube-system	Pending	—	Deployment	metrics-server	0	5 / 100 mCPU	50 / 300 MB	— / — MB

FIN.