Cluster Autoscaler with VCD Tanzu Kubernetes clusters

Guide for Customer users

vmware®

Table of contents

Introduction

Cluster Autoscaler is a tool that automatically adjusts the size of the Kubernetes cluster by adding or removing the worker plane node for one of the following conditions. Cluster Autoscaler offers the auto-scaling of the Kubernetes worker nodes without downtime to affecting services, cost effective, with redundancy and resilience, and less maintenance.

- Failure of a pod because of resource constraints
- Underutilized Worker nodes with running for an prolonged period of time, where pods can be placed on other existing nodes.

In this article we will understand how VCD customers can configure Autoscaler on Tanzu Kubernetes Cluster provisioned by Container Service Extension.

Container service extension manages VMware Cloud Director Platform's Tanzu Kubernetes Clusters lifecycle adhering to VCD's infrastructure SLAs, and multi-tenancy requirements, for more information, please refer to Official Documentation for more details.

Cluster Autoscaler configuration happens at cluster level settings. So, this article is targeted for customer organization's individual cluster owner, cluster admin and customer org admin roles.

Scope and pre-requisites

The following sections describes step-by-step guidance for Kubernetes Cluster owner, user role who's going to configure and manage Autoscaler configuration. We assume that a Tanzu Kubernetes Cluster is provisioned successfully by the user on VMware Cloud Director Tenant portal. We also assume that,

1. The user is familiar with VMware Cloud Director's Kubernetes Container UI plugin to download the 'kubeconfig' file.
2. User is familiar with *Kubernetes* and *Cluster API concepts* and comfortable modifying capiyaml for Autoscaler settings.
3. Customer has installed required tools such as kubectl, clusterctl, go-lang and Autoscaler on a system where user can access cluster using kubeconfig file,

Autoscaler with Tanzu Kubernetes Cluster

There are three main steps to deploy and realize Autoscaler on worker nodes as follows:

1. Prepare local system with pre-requisites.
2. Deploy Autoscaler on desired Kubernetes cluster.
3. Scale worker nodes based on requirements.

Step 1 – Prepare local system with required tools.

This step is targeted for cluster owners who'd like to manage cluster Autoscaler from their local end point using kubectl. Please refer to this *blog* to prepare system with clusterctl (section 'Generate capiyaml' payload).

Following steps are described for mac OS, but this can be done for any operating system by following documentation *here*.

- Install Golang:
- brew update && brew install golang
- GOOS=Darwin
- Install Autoscaler

  git clone *https://github.com/kubernetes/Autoscaler*

`cd Autoscaler/cluster-Autoscaler/`

- Execute make command to compile Autoscaler binary.

  `Make`

- Verify that cluster-Autoscaler-amd64 binaries are created.

Step 2 – Enable Autoscaler on desired Kubernetes Cluster

Please review cluster Autoscaler official documentation before configuring the parameters *here*.

1. Download Kubeconfig and set the context to deploy cluster Autoscaler.
2. Create Kubernetes deployment file sample spec as follows:

```
apiVersion: v1
kind: ServiceAccount
metadata:
labels:
k8s-addon: cluster-autoscaler.addons.k8s.io
k8s-app: cluster-autoscaler
name: cluster-autoscaler
namespace: kube-system

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
name: cluster-autoscaler
labels:
k8s-addon: cluster-autoscaler.addons.k8s.io
k8s-app: cluster-autoscaler
rules:
- apiGroups: [""]
resources: ["events", "endpoints"]
verbs: ["create", "patch"]
- apiGroups: [""]
resources: ["pods/eviction"]
verbs: ["create"]
- apiGroups: [""]
resources: ["pods/status"]
verbs: ["update"]
- apiGroups: [""]
resources: ["endpoints"]
resourceNames: ["cluster-autoscaler"]
verbs: ["get", "update"]
- apiGroups: [""]
resources: ["nodes"]
verbs: ["watch", "list", "get", "update"]
- apiGroups: ["cluster.x-k8s.io"]
resources: ["machinedeployments", "machines", "machinesets", "machinedeployments/scale"]
verbs: ["watch", "list", "get", "update"]
- apiGroups: [""]
resources:
- "pods"
- "services"
```

```
- "replicationcontrollers"
- "persistentvolumeclaims"
- "persistentvolumes"
verbs: ["watch", "list", "get"]
- apiGroups: ["extensions"]
resources: ["replicasets", "daemonsets"]
verbs: ["watch", "list", "get"]
- apiGroups: ["policy"]
resources: ["poddisruptionbudgets"]
verbs: ["watch", "list"]
- apiGroups: ["apps"]
resources: ["statefulsets", "replicasets", "daemonsets"]
verbs: ["watch", "list", "get"]
- apiGroups: ["storage.k8s.io"]
resources: ["storageclasses", "csinodes"]
verbs: ["watch", "list", "get"]
- apiGroups: ["batch", "extensions"]
resources: ["jobs"]
verbs: ["get", "list", "watch", "patch"]
- apiGroups: ["coordination.k8s.io"]
resources: ["leases"]
verbs: ["create"]
- apiGroups: ["coordination.k8s.io"]
resourceNames: ["cluster-autoscaler"]
resources: ["leases"]
verbs: ["get", "update"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
name: cluster-autoscaler
namespace: kube-system
labels:
k8s-addon: cluster-autoscaler.addons.k8s.io
k8s-app: cluster-autoscaler
rules:
- apiGroups: [""]
resources: ["configmaps"]
verbs: ["create","list","watch"]
- apiGroups: [""]
resources: ["configmaps"]
resourceNames: ["cluster-autoscaler-status", "cluster-autoscaler-priority-expander"]
verbs: ["delete", "get", "update", "watch"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
name: cluster-autoscaler
labels:
k8s-addon: cluster-autoscaler.addons.k8s.io
k8s-app: cluster-autoscaler
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: cluster-autoscaler
```

```
subjects:
- kind: ServiceAccount
name: cluster-autoscaler
namespace: kube-system

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
name: cluster-autoscaler
namespace: kube-system
labels:
k8s-addon: cluster-autoscaler.addons.k8s.io
k8s-app: cluster-autoscaler
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: Role
name: cluster-autoscaler
subjects:
- kind: ServiceAccount
name: cluster-autoscaler
namespace: kube-system

---
apiVersion: apps/v1
kind: Deployment
metadata:
name: cluster-autoscaler
namespace: kube-system
labels:
app: cluster-autoscaler
spec:
replicas: 1
selector:
matchLabels:
app: cluster-autoscaler
template:
metadata:
labels:
app: cluster-autoscaler
annotations:
cluster-autoscaler.kubernetes.io/safe-to-evict: 'false'
spec:
serviceAccountName: cluster-autoscaler
containers:
- image: k8s.gcr.io/autoscaling/cluster-autoscaler:v1.20.0
name: cluster-autoscaler
resources:
limits:
cpu: 100m
memory: 500Mi
requests:
cpu: 100m
memory: 500Mi
command:
- ./cluster-autoscaler
```

```
- --v=4
- --stderrthreshold=info
- --cloud-provider=clusterapi
- --expendable-pods-priority-cutoff=-10
- --scale-down-delay-after-delete=10s
- --scale-down-delay-after-add=10s
- --scale-down-delay-after-failure=10s
- --expander=least-waste
- --node-group-auto-discovery=clusterapi:namespace=k8s01-ns
- --balance-similar-node-groups
- --skip-nodes-with-system-pods=false
```

Deploy Cluster Autoscaler by applying the above created file with kubectl command as follows:

```
kubectl  apply -f cluster-autoscaler.yaml
```

3. Verify the cluster Autoscaler pod is running under kube-system namespace as follows

```
kubectl get sa cluster-autoscaler -n kube-system
NAME             SECRETS   AGE
cluster-autoscaler   1      31m
kubectl get deployment cluster-autoscaler -n kube-system
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
cluster-autoscaler   1/1    1         1         13m

kubectl get pod cluster-autoscaler-654889dc79-gssl4 -n kube-system
NAME                      READY   STATUS    RESTARTS   AGE
cluster-autoscaler-654889dc79-gssl4   1/1    Running   0       13m
```

Step 3 - Scale worker nodes using Autoscaler

For this section we will use VCD api to collect the desired cluster information using VCD API as cluster author/user role (same user in step 2)

1. Get cluster info using following VCD API call:
2. Copy content of capiYaml to text, convert it to yaml file.
3. Modify machinedeployment section to add Autoscaler to discover worker node.

   Existing machinedeployment metadata would look like –

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: api1-md-0
  namespace: default
```

After modification:

```
apiversion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: jarvis-02-worker-node-pool-1
  namespace: jarvis-02-ns
  cluster.x-k8s.io/cluster-api-Autoscaler-node-group-max-size: "5"
  cluster.x-k8s.io/cluster-api-Autoscaler-node-group-min-size: "1"
```

4. User can enable Autoscaler on multiple worker node pools by modifying its relevant node pool machine deployment config sections as described in step 3.
5. As seen in above steps, in Machine Deployment, we configure maximum and minimum sizes. The TKG cluster also has a deployment called "RDE Projector" to maintain desired worker nodes set by VCD UI/API "MachineDeployment.spec.repllicas". The replicas configuration ensures the worker node for this specific worker node pool, on VCD customer organization. When user attempts to manage the same worker node pool using VCD UI/API and cluster Autoscaler, the outcome may become unexpected as replicas filed enforcement can overwrite Autoscaler's settings on the same worker-node pool. This behavior may result in indeterministic no. of worker nodes or continuous deletion/creation of worker nodes. This behavior is expected and must be thought out by cluster user on management best practices. To overcome this situation, remove the replica field from the RDE API payload as follows:
6. To remove conflict from CSE's RDE projector component, delete replicas field from the machine deployment section of the "RDE.entity.spec.capiYaml" follows:

   Before:

   spec:

   clusterName: k8s01

   replicas: 1

   template:

   After:

   spec:

   clusterName: k8s01

   ~~**replicas: 1**~~

   template: 1

7. modify the cluster capiyaml using PUT call as described *here* in section 'Resize a Cluster'. We can also convert jsonstring of the capiyaml output by using command: `yq -P e capi.yaml | sed ':a;N;$!ba;s/\n/\\n/g' | sed 's/$/\\n/' > capi-new.txt`

Design/Operation Considerations with Autoscaler:

1. Kubernetes cluster configuration can be managed in many ways – kubectl. Clusterctl; In this case by Autoscaler. Tenant user needs to understand it is very easy to overwrite cluster properties by using multiple tools. For example, if user is managing worker node for a specific worker node pool using cluster Autoscaler, user must refrain from managing worker node pool count using VCD UI/API to

scale the worker node pool. Another example is, if user wants to manually scale worker node pool after Autoscaler is enabled, user needs to follow the steps below:

- Disable Autoscaler
- Update no. of worker nodes from VCD UI or API and re-insert replicas field as follows in the machineDeployment as follows:

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: MachineDeployment
metadata:
  name: api4-md-0
  namespace: api4-ns
spec:
  clusterName: api4
  replicas: 1
  selector:
    matchLabels: null
```

2. The minimum supported worker node count is one. Scale from Zero with Autoscaler is not supported.
3. CSE provisions self-managing Tanzu Kubernetes clusters. Each cluster is its own management cluster to perform scale, upgrade, update cluster operations.

   Optionally User can also treat CSE created cluster as management cluster to create another workload cluster using clusterapi. In the latter case, refer to cluster auto scale for workload and management clusters *here*

References:

1. Kubernetes Autoscaler with Cluster API
2. Autoscaler Frequently Asked Questions

About the Authors

Rohan Mukesh is a Sr. Staff Solutions Architect at VMware in Modern Application and Management Business unit.

Sachi Bhatt is a Staff Technical Product Manager at VMware in Cloud Services Business Unit.

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 vmware.com Item No: vmw-wp-tech-temp-a4-word-101-proof 6/20