

# Amazon Elastic Kubernetes Service Anywhere Bare Metal on Dell PowerFlex

January 2023

H19277

## White Paper

### Abstract

This white paper describes the use of Amazon Elastic Kubernetes Service Anywhere with bare metal on the Dell PowerFlex infrastructure to deliver a seamless and automated operations experience for PowerFlex infrastructure across cloud-native and traditional workloads.

Dell Technologies Solutions

PowerFlex Engineering

Validated

## Copyright

The information in this publication is provided as is. Dell Inc. makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose.

Use, copying, and distribution of any software described in this publication requires an applicable software license.

Copyright © 2023 Dell Inc. or its subsidiaries. Published in the USA 01/23 White Paper H19277.

Dell Inc. believes the information in this document is accurate as of its publication date. The information is subject to change without notice.

**Note:** This document may contain language from third-party content that is not under Dell's control and is not consistent with Dell's current guidelines for Dell's own content. When such third-party content is updated by the relevant third parties, this guide will be revised accordingly.

# Contents

Introduction .....	4
PowerFlex product overview .....	6
Solution architecture .....	9
Installing an Amazon EKS Anywhere Bare Metal cluster on Dell PowerFlex.....	11
Creating an Amazon EKS Anywhere Bare Metal cluster .....	13
Validating the cluster deployment .....	23
Installing the PowerFlex CSI driver.....	24
Deployment tests and validation.....	27
Amazon EKS Connector .....	34
Conclusion.....	46
Hardware qualification details.....	46
References.....	48

# Introduction

## Document purpose

This white paper describes the deployment of Amazon Elastic Kubernetes Service (EKS) Anywhere Bare Metal on PowerFlex and the integration of PowerFlex CSI driver to provision persistent PowerFlex storage for containerized applications that are deployed on this platform and validates the common tasks that are outlined in the Amazon EKS Anywhere cluster management documentation. This white paper also describes using the bare metal infrastructure to enable connectivity and portability across Amazon Web Services public cloud environments.

## Executive summary

Amazon EKS Anywhere Bare Metal is a deployment option for Amazon EKS. It enables customers to create and operate Kubernetes clusters on customer-managed infrastructure, supported by AWS.

Many organizations are looking to deploy and manage their Kubernetes clusters with ease and at scale. Amazon EKS Anywhere can be used to manage Kubernetes clusters along with the option to deploy on bare metal servers.

With Amazon EKS Anywhere, PowerFlex customers benefit from a flexible deployment solution that scales as needed with the ability to separate compute and storage growth, in a unified fabric architecture.

The PowerFlex family offers key value propositions for traditional and cloud-native workloads, deployment flexibility, linear scalability, predictable high performance, and enterprise-grade resilience. The PowerFlex Container Storage Interface (CSI) provides an interface to automatically create and manage persistent volumes for Kubernetes clusters in Dell PowerFlex software defined storage.

The combination of EKS Anywhere and PowerFlex brings organizations to the next step in their IT journey. They can provide infrastructure as code and empower their DevOps teams to be the innovation engine for their organization.

## We value your feedback

Dell Technologies and the authors of this document welcome your feedback on the solution and the solution documentation. Contact the Dell Technologies Solutions team by [email](#).

**Author:** Syed Abrar

**Contributors:** Anthony Foster, Kevin M Jones

**Content Editor:** Ripa Bhagawati

---

**Note:** For links to additional documentation for this solution, see the PowerFlex section on the [Dell Technologies Info Hub](#).

---

## Objective

PowerFlex is a fully integrated, preconfigured, and tested software-defined infrastructure system that is an ideal platform for Amazon EKS Anywhere. Amazon EKS Anywhere helps customers automate cluster management, reduce support costs, and eliminate the redundancy of using multiple open-source or third-party tools to manage Kubernetes clusters while driving operational simplicity through Kubernetes orchestration.

## Audience

This white paper is intended for IT administrators, customers, sales engineers, field consultants, and anyone that is interested in configuring and deploying a Kubernetes cluster with the PowerFlex CSI driver to dynamically provision persistent volumes (PV) in an Amazon EKS Anywhere managed Kubernetes cluster.

The audience of this white paper must have a working knowledge of containers, Kubernetes cluster, PowerFlex, and Amazon EKS Anywhere.

## Terminology

**Table 1. Terminology**

Term	Definition
AGR	Aggregate
CA	Certificate Authority
CSI	Container Storage Interface
CRD	Custom Resource Definition
CAPT	Cluster-API-Provider-Tinkerbell
CAPI	Cluster-API
EKS A	Elastic Kubernetes Service Anywhere
ITOM	IT Operations Management
LCM	Life Cycle Management
MDM	Meta Data Manager
PV	Persistent Volume
PVC	Persistent Volume Claim
SO	Storage only
CO	Compute only
SDC	Storage Data Client for PowerFlex
SDS	Storage Data Server for PowerFlex
SLA	Service Level Agreement
SSD	Solid-State Disk for PowerFlex
ToR	Top of Rack
VLAN	Virtual Local Area Network

# PowerFlex product overview

## Dell PowerFlex family

Dell PowerFlex is a software-defined infrastructure platform that reduces operational and infrastructure complexity. It empowers organizations to move faster by delivering flexibility, elasticity, and simplicity with extraordinary predictable performance for mission-critical workloads, while also providing resiliency at scale. The Dell PowerFlex family provides a foundation that combines compute and high-performance storage resources in a managed, unified fabric.

## PowerFlex delivers transformational value

### Delivers stringent SLAs effortlessly

PowerFlex uses software to unlock the full potential of rapid advances in industry-standard hardware and deliver extreme SLA outcomes. PowerFlex aggregates resources across a broad set of nodes, unlocking massive input, output, and throughput performance while minimizing the latency. Its self-balancing architecture eliminates any hotspots and ensures consistency and simplicity over time. You can scale the system while linearly scaling performance from a minimum of four nodes to thousands of nodes, on-demand and without any disruption. And with its self-healing architecture, PowerFlex can handle outages, upgrades, and maintenance without downtime resulting in 99.9999 <sup>1</sup>percent availability.

### Flexible and dynamic architecture

Dell PowerFlex delivers transformational agility that enables organizations to rapidly respond to changing business needs. It offers flexibility to mix and match storage, compute, and hyperconverged nodes in a dynamic deployment, allowing you to scale storage and compute resources together or independently, one node at a time.

### Shared platform for heterogeneous workloads

Dell PowerFlex supports a broad range of operating environments simultaneously such as bare metal operating systems, hypervisors, and container platforms with a unified underlying infrastructure platform and management. You can also support heterogeneous workloads with varying requirements on a flexible shared infrastructure platform and modernize your application architectures on your schedule.

### Extensive automation for predictability and simpler workflows

Dell PowerFlex offers full-stack IT Operations Management (ITOM) and Life Cycle Management (LCM) with Dell PowerFlex Manager. It provides extensive automation capabilities with Dell PowerFlex Manager REST APIs and custom Ansible modules to integrate with your infrastructure, application, and DevOps workflows. Dell PowerFlex Manager enables automated deployments and expansions with minimal time for the IT team, letting them focus on other strategic initiatives.

### Broad ecosystem of workload solutions

Dell PowerFlex is designed to deliver superior outcomes at any scale for the most demanding mission-critical environments. It is optimized for a wide range of validated workload solutions ranging from traditional relational databases and modern cloud-native NoSQL databases to throughput-intensive analytics workloads.

---

<sup>1</sup> Workload performance claims based on internal Dell testing. (REF-0000470)

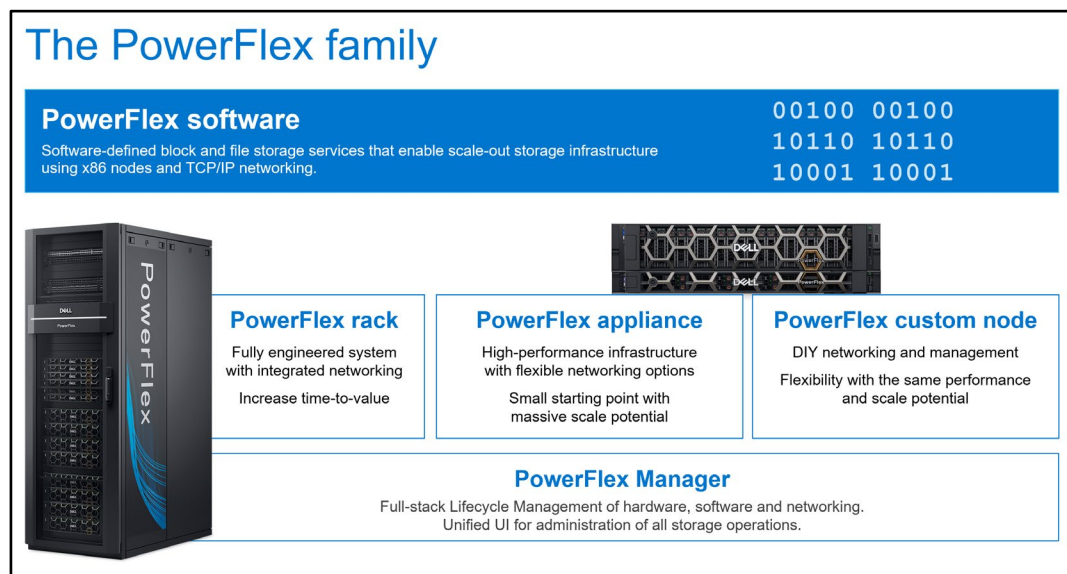


Figure 1. PowerFlex family

### Dell PowerFlex software

The PowerFlex software provides software-defined storage services. It delivers high performance, highly resilient block storage service that can scale to thousands of nodes.

### Dell PowerFlex consumption options

Dell PowerFlex is available with multiple consumption options to help you meet your project and data center requirements. Dell PowerFlex appliance and Dell PowerFlex rack provide customers with the flexibility to choose a deployment option to meet their exact requirements.

#### Dell PowerFlex rack

Dell PowerFlex rack is a fully engineered system, with integrated networking that enables customers to simplify deployments and accelerate time to value. The preconfigured and flexible network options, ingrained security controls, end-to-end life cycle management, secure call home, and single call support are some of the many key features that help customers streamline day-2 operation, deliver agility and resilience, and maintain a security posture.

#### Dell PowerFlex appliance

Dell PowerFlex appliance offers customers a smaller configuration of four nodes, while enabling them to use their existing network infrastructure.

With Dell PowerFlex, customers deploy to match their initial needs and expand with massive scale potential, without having to compromise on performance and resiliency.

### Flexible consumption-based billing options

Dell PowerFlex is available through APEX custom solutions by the APEX Flex on Demand and APEX Data center Utility for customers looking to adopt consumption-based OpEx models.

### **APEX Flex on Demand**

APEX Flex on Demand allows you to pay for technology as you use it and provides immediate access to buffer capacity. Your payment adjusts to match your usage.

### **APEX Data Center Utility**

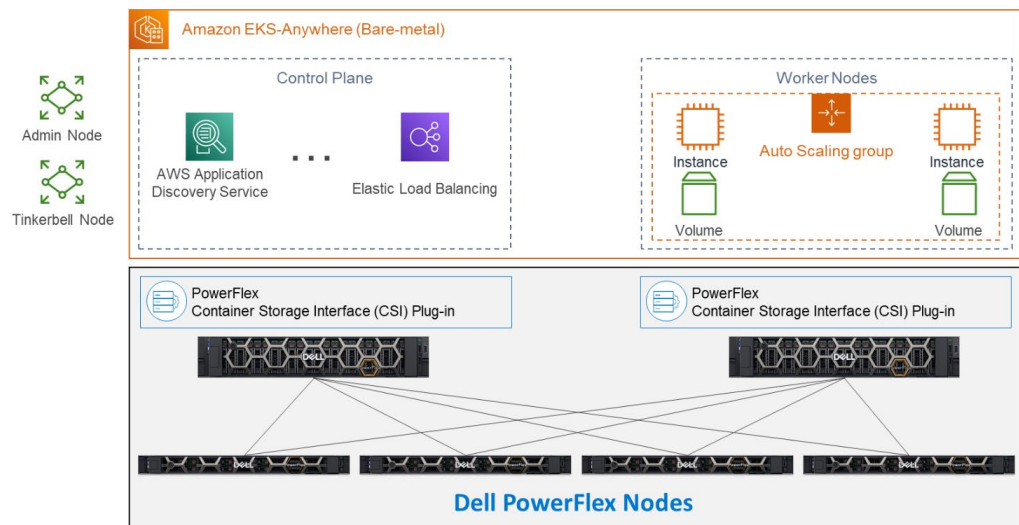
APEX Data Center Utility provides the leading product portfolio from Dell Technologies that is coupled with your choice of professional services and support to manage your data center and its operations. A single invoice provides monthly payments that are based on a predictable rate and vary based on your usage.



## Solution architecture

This section describes how Amazon EKS Anywhere Bare Metal is deployed on a Dell PowerFlex bare metal server. This solution can be deployed on other Dell PowerFlex family products as well.

**Note:** The solution that is described here was validated in the Dell engineering lab with the hardware specification that is provided in the configuration details section.



**Figure 2. Bare Metal architecture of the Amazon EKS Anywhere environment on Dell PowerFlex**

The bottom portion of the figure consists of PowerFlex – storage-only nodes. The center of the diagram consists of the hosts that are used for the control plane and worker nodes. These nodes are PowerFlex – compute-only nodes. On the left of the diagram are the admin and Tinkerbell nodes that are used for administration of the environment. At the upper left, we have the control plane that provides operational control and orchestration. The worker node, at the upper right, handle the workloads.

Each storage node contains five 1.4TB SAS SSD drives and two 25GbE NVIDIA Mellanox network links. For the validation, four PowerFlex storage nodes were used to provide full storage redundancy.

Amazon EKS Anywhere tinkerbell process installs the Ubuntu operating system on the two compute nodes through iPXE boot process. After Ubuntu installation, SDC (Storage Data Client) and CSI (container storage interface) is installed to provide a storage provisioning interface.

This is a two-layer architecture where compute and storage nodes are separated. It is important to note that there is no hypervisor installed.

---

**Note:** The validation was performed in a nonproduction environment. For more information about the minimum hardware requirements, refer to this link:

<https://anywhere.eks.amazonaws.com/docs/reference/baremetal/bare-prereq/>

---

Using a two-layer architecture makes it possible to scale resources independently as needed in the environment for optimal resource utilization. Hence, if more storage is needed, it can be scaled independently of compute. Additional compute capacity can be added to the environment as needed.

Outside the Amazon EKS Anywhere instance, there are two physical nodes. Both are central to building the control plane and worker nodes. The admin node is used to control the Amazon EKS Anywhere instance and serves as a portal to upload inventory information to the Tinkerbell node. The Tinkerbell node serves as the infrastructure services stack and is key to the provisioning and PXE booting of the bare metal servers.

The installation process requires you to create a data center hardware configuration file with the details pertaining to your physical server hardware in csv format. Target cluster creation process needs the input file to be in YAML format. The cluster config file is generated from the information provided in the hardware CSV file. With the information gathered from the cluster specification and hardware CSV file, the three custom resource definitions (CRD's) are created. These include

- Hardware custom resources
- Template custom resources
- Workflow custom resources

The Tinkerbell process creates a local `kind` cluster on the admin host to install the Cluster-API (CAPI) and the Cluster-API-Provider-Tinkerbell (CAPT).

For more information about the booting and deployment process, see <https://anywhere.eks.amazonaws.com/docs/reference/baremetal/netbooting-and-tinkerbell/>

With the base control environment operation, CAPI creates the control plane and worker node resources, and CAPT maps and powers on the corresponding bare metal servers. The bare metal servers PXE boot from the Tinkerbell node to run corresponding roles. Kubernetes management resources are transferred from the bootstrap cluster to the target Amazon EKS Anywhere workload cluster.

The `kind` cluster is then deleted from the admin machine. This creates both the control plane and worker nodes. SDC drivers are installed on the worker nodes along with the Dell CSI plug-in for PowerFlex. Workloads can be deployed on the worker nodes as needed.

# Installing an Amazon EKS Anywhere Bare Metal cluster on Dell PowerFlex

## Overview

This section describes the steps to install Amazon EKS Anywhere Bare Metal cluster on Ubuntu and to configure the PowerFlex CSI driver on a PowerFlex cluster.

## Requirements

The following table describes the prerequisites for building an Amazon EKS Anywhere Bare Metal cluster:

**Table 2. Requirements for Amazon EKS Anywhere cluster on Dell PowerFlex**

Name	Description	Reference
Tinkerbell	Used for Bare metal provisioning and management.	<a href="https://tinkerbell.org/">https://tinkerbell.org/</a>
Ubuntu	Admin node All the operations and management of Amazon EKS Anywhere is managed from the admin node.	
Eksctl-Anywhere	Installs Eksctl -Anywhere You can install eksctl and eksctl-anywhere using homebrew package manager.	<ul style="list-style-type: none"> <li>• <a href="https://brew.sh/">https://brew.sh/</a></li> <li>• <a href="https://docs.aws.amazon.com/eks/latest/userguide/eksctl-anywhere.html">brew install aws/tap/eks-anywhere</a></li> </ul>
Docker Compose	Installs Docker Composer	
Docker CE	Installs Docker CE	<a href="https://docs.docker.com/engine/install/ubuntu/">https://docs.docker.com/engine/install/ubuntu/</a>
Kubectll	Kubectll to interact with Kubernetes cluster.	<a href="https://kubernetes.io/docs/tasks/tools/install-kubectll/">https://kubernetes.io/docs/tasks/tools/install-kubectll/</a>
Ubuntu	Ensures that the nodes are accessed using SSH and the required ports are opened before the installation.	
Dell PowerFlex CSI	Dell PowerFlex CSI storage orchestration	<a href="https://github.com/dell/csi-powerflex">https://github.com/dell/csi-powerflex</a>

## Set up Ubuntu administrative machine

Install the docker containers on the administrative machine by following the procedure described in this link : <https://docs.docker.com/engine/install/ubuntu/>

## Install Amazon EKS Anywhere

Install the Amazon EKS Anywhere bundle on the administrative machine by following the procedure in the following link:

<https://anywhere.eks.amazonaws.com/docs/getting-started/install/>

## Install Tinkerbell

To install and set up the Tinkerbell stack, you will need to have the administrative machine and the bare metal servers on the same network:

1. Extract the given tinkerbell stack tar:  

```
tar -xzf tinkerbell-stack.tar.gz
```
2. Change the directory to the extracted bundle:  

```
cd tinkerbell-stack
```
3. Set the `TINKERBELL_HOST_IP` env var. This is usually the IP of the machine on which Tinkerbell infrastructure runs.  

```
export TINKERBELL_HOST_IP= XXX.XXX.XXX.XXX
```

Run `docker-compose up`

```
docker-compose up -d
```

### Notes:

- The Tinkerbell IP address is required and identifies the Tinkerbell service. This IP address must be on the same network as the physical hosts that are being used to provide both the control plane and worker nodes and must be a unique IP in the network range that does not conflict with other IPs. This allows the Tinkerbell services to move from the admin machine and run on the target server. This IP address makes it possible for the stack to be used for future provisioning needs.
- The setup commands that are described in this document are from the beta version of Amazon EKS-A. Some of the setup commands have changed in the GA version. The new commands are incorporated in the document; however, you may still see screenshots referring to old commands. Hence, it is recommended to check the latest documentation.

# Creating an Amazon EKS Anywhere Bare Metal cluster

## Overview

Use the `eks-a` command to create your cluster based on the configuration from your admin machine.

## Cluster creation steps

The following steps describe the creation of a cluster:

1. Capture the information about the physical hosts to be used as follows:

```
iDRAC: 10.X.X.X
Slot/Embedded Port  MAC                               Switch ID
Switch Port
Slot 2              1                               0C:42:A1:8C:F4:F0
0c:29:ef:eb:1a:00   ethernet1/1/5
Slot 2              2                               0C:42:A1:8C:F4:F1
c0:3e:ba:d9:6b:00   ethernet1/1/15
Slot 3              1                               0C:42:A1:8C:F2:20
c0:3e:ba:d9:6b:00   ethernet1/1/5
Slot 3              2                               0C:42:A1:8C:F2:21
0c:29:ef:eb:1a:00   ethernet1/1/15
iDRAC: 10.X.X.X
Slot/Embedded Port  MAC                               Switch ID
Switch Port
Slot 2              1                               0C:42:A1:8C:E4:58
0c:29:ef:eb:1a:00   ethernet1/1/6
Slot 2              2                               0C:42:A1:8C:E4:59
c0:3e:ba:d9:6b:00   ethernet1/1/16
Slot 3              1                               0C:42:A1:8C:E4:50
c0:3e:ba:d9:6b:00   ethernet1/1/6
Slot 3              2                               0C:42:A1:8C:E4:51
0c:29:ef:eb:1a:00   ethernet1/1/16
```

2. Create a hardware CSV file: Create a hardware CSV file that contains an entry for every physical machine that will be added during cluster creation. For more information, see <https://aws.amazon.com/blogs/containers/getting-started-with-eks-anywhere-on-bare-metal/>.
3. Edit the `$CLUSTER_NAME.yaml` file to enter the values for your environment. Cluster names must start with an upper or lowercase letter and can contain only letters, numbers, and dashes, as shown in the following example:

```
eksctl anywhere generate clusterconfig $CLUSTER_NAME --provisioner
tinkerbell > clusterconfig.yaml
```

4. Use the `ssh-keygen` command to create the private and public keys for the `eksadmin` user to allow SSH connectivity to the Amazon EKS-A hosts:

```
$ ssh-keygen
```

For example, See the following sample YAML file:

---

**Note:** Neither Ubuntu nor RHEL OS images for Bare Metal Amazon EKS Anywhere are generally available . However, you can see [Building node images](#) for information on how to build Amazon EKS Anywhere images of these Linux distributions.

---

```
apiVersion: anywhere.eks.amazonaws.com/v1alpha1
kind: Cluster
metadata:
  name: eksadmin-18990
spec:
  clusterNetwork:
    cniConfig:
      cilium: {}
    pods:
      cidrBlocks:
        - 192.168.0.0/16
    services:
      cidrBlocks:
        - 10.96.0.0/12
  controlPlaneConfiguration:
    count: 3
    endpoint:
      host: "192.168.151.80"
    machineGroupRef:
      kind: TinkerbellMachineConfig
      name: eksadmin-18990-cp
    datacenterRef:
      kind: TinkerbellDatacenterConfig
      name: eksadmin-18990
  kubernetesVersion: "1.22"
  managementCluster:
    name: eksadmin-18990
  workerNodeGroupConfigurations:
    - count: 1
      machineGroupRef:
        kind: TinkerbellMachineConfig
        name: eksadmin-18990
      name: md-0
```

```

---
apiVersion: anywhere.eks.amazonaws.com/v1alpha1
kind: TinkerbellDatacenterConfig
metadata:
  name: eksadmin-18990
spec:
  tinkerbellaCertURL: "http://192.168.151.58:42114/cert"
  tinkerbellaGRPCAuth: "192.168.151.58:42113"
  tinkerbellaIP: "192.168.151.58"
  tinkerbellaPBnJGRPCAuth: "192.168.151.58:50051"

---
apiVersion: anywhere.eks.amazonaws.com/v1alpha1
kind: TinkerbellMachineConfig
metadata:
  name: eksadmin-18990-cp
spec:
  osFamily: ubuntu
  templateRef:
    kind: TinkerbellTemplateConfig
    name: eksadmin-18990
  users:
    - name: ec2-user
      sshAuthorizedKeys:
        - ssh-rsa
        AAAAB3xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
---
apiVersion: anywhere.eks.amazonaws.com/v1alpha1
kind: TinkerbellMachineConfig
metadata:
  name: eksadmin-18990
spec:
  osFamily: ubuntu
  templateRef:

```



```

kind: TinkerbellTemplateConfig
name: eksadmin-18990
users:
- name: ec2-user
  sshAuthorizedKeys:
  - ssh-rsa
  AAAAB3xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  ---
apiVersion: anywhere.eks.amazonaws.com/v1alpha1
kind: TinkerbellTemplateConfig
metadata:
  name: eksadmin-18990
spec:
  template:
    global_timeout: 6000
    id: ""
    name: eksadmin-18990
    tasks:
    - actions:
      - environment:
        COMPRESSED: "true"
        DEST_DISK: /dev/sda
        IMG_URL: https://anywhere-
assets.eks.amazonaws.com/releases/bundles/8/artifacts/raw/1-
22/ubuntu-v1.22.6-eks-d-1-22-4-eks-a-8-amd64.gz
        image: image2disk:v1.0.0
        name: stream-image
        timeout: 360
      - environment:
        BLOCK_DEVICE: /dev/sda1
        CHROOT: "y"
        CMD_LINE: apt -y update && apt -y install openssl
        DEFAULT_INTERPRETER: /bin/sh -c
        FS_TYPE: ext4
        image: cexec:v1.0.0

```

```

name: install-openssl
timeout: 90
- environment:
  CONTENTS: |
    network:
      version: 2
      renderer: networkd
      ethernets:
        eno1:
          dhcp4: true
  DEST_DISK: /dev/sda1
  DEST_PATH: /etc/netplan/config.yaml
  DIRMODE: "0755"
  FS_TYPE: ext4
  GID: "0"
  MODE: "0644"
  UID: "0"
image: writefile:v1.0.0
name: write-netplan
timeout: 90
- environment:
  CONTENTS: |
    datasource:
      Ec2:
        metadata_urls: ["http://192.168.151.58:50061"]
        strict_id: false
    system_info:
      default_user:
        name: tink
        groups: [wheel, adm]
        sudo: ["ALL=(ALL) NOPASSWD:ALL"]
        shell: /bin/bash
      manage_etc_hosts: localhost
      warnings:

```

```

        dsid_missing_source: off
        DEST_DISK: /dev/sda1
        DEST_PATH: /etc/cloud/cloud.cfg.d/10_tinkerbelle.cfg
        DIRMODE: "0700"
        FS_TYPE: ext4
        GID: "0"
        MODE: "0600"
        UID: "0"
        image: writefile:v1.0.0
        name: add-tink-cloud-init-config
        timeout: 90
- environment:
    CONTENTS: |
        datasource: Ec2
        DEST_DISK: /dev/sda1
        DEST_PATH: /etc/cloud/ds-identify.cfg
        DIRMODE: "0700"
        FS_TYPE: ext4
        GID: "0"
        MODE: "0600"
        UID: "0"
        image: writefile:v1.0.0
        name: add-tink-cloud-init-ds-config
        timeout: 90
- environment:
    BLOCK_DEVICE: /dev/sda1
    FS_TYPE: ext4
    image: kexec:v1.0.0
    name: kexec-image
    pid: host
    timeout: 90
    name: eksadmin-18990
    volumes:
    - /dev:/dev

```

```
- /dev/console:/dev/console
- /lib/firmware:/lib/firmware:ro
worker: '{{.device_1}}'
version: "0.1"
```

For more information about the cluster specification, see

<https://anywhere.eks.amazonaws.com/docs/reference/clusterspec/baremetal/>

5. After updating the YAML file, apply the `EKS-A` command to create the cluster.
6. Run the following command to start the deployment on the admin node:

```
eksctl anywhere create cluster --file clusterconfig.yaml
--hardware-csv hardware.csv
```

The following example shows the cluster creation:

```

$ sudo eksctl anywhere create cluster --hardware-csv
hardware.csv -f eksadmin29047.yaml
Warning: The recommended number of control plane nodes is 3
or 5
Warning: The recommended number of control plane nodes is 3
or 5
Performing setup and validations
hook path override set  {"path":
"http://10.x.x.x:20000/images/hook"}
✓ Tinkerbell Provider setup is valid
✓ Validate certificate for registry mirror
✓ Validate authentication for git provider
✓ Create preflight validations pass
Creating new bootstrap cluster
Provider specific pre-capi-install-setup on bootstrap
cluster
Installing cluster-api providers on bootstrap cluster
Provider specific post-setup
Creating new workload cluster
Installing networking on workload cluster
Creating EKS-A namespace
Installing cluster-api providers on workload cluster
Installing EKS-A secrets on workload cluster
Installing resources on management cluster
Moving cluster management from bootstrap to workload
cluster
Installing EKS-A custom components (CRD and controller) on
workload cluster
Installing EKS-D components on workload cluster
Creating EKS-A CRDs instances on workload cluster
Installing GitOps Toolkit on workload cluster
GitOps field not specified, bootstrap flux skipped
Writing cluster config file
Deleting bootstrap cluster
🎉 Cluster created!
-----

The Amazon EKS Anywhere Curated Packages are only available
to customers with the
Amazon EKS Anywhere Enterprise Subscription
-----

Enabling curated packages on the cluster
Installing helm chart on cluster      {"chart": "eks-
anywhere-packages", "version": "0.2.16-eks-a-21"}
⚠ Unable to create credentials for curated packages:
{"warning": "environment variables
EKSA_AWS_SECRET_ACCESS_KEY and EKSA_AWS_ACCESS_KEY_ID not
provided"}

```

## 7. Run the following command to Export kubeconfig:

Amazon Elastic Kubernetes Service Anywhere Bare Metal on Dell PowerFlex

White Paper

```
export KUBECONFIG=$CLUSTER_NAME/$CLUSTER_NAME.kubeconfig
```

8. After the deployment of the Amazon EKS Anywhere Bare Metal cluster, verify the cluster status by running the following command:

```
kubectl get po -A -l control-plane=controller-manager
```

The following example shows the cluster status:

```
$ kubectl get po -A -l control-plane=controller-manager
```

NAMESPACE	NAME
READY	STATUS
RESTARTS	AGE
capi-kubeadm-bootstrap-system	capi-kubeadm-bootstrap-controller-
manager-776b89dcdf-z5681	Running 0 6d17h
capi-kubeadm-control-plane-system	capi-kubeadm-control-plane-
controller-manager-7966d44fc-tzjpl	1/1 Running 0 6d17h
capi-system	capi-controller-manager-ffcf6f4f6-
c9dbg	1/1 Running 0 6d17h
capt-system	capt-controller-manager-65bffcdbc5-
p7w8k	1/1 Running 0 6d17h
dell-csi-operator	dell-csi-operator-controller-
manager-64c68f44dc-jmzc6	1/1 Running 0 6d14h
eks-a-packages	eks-anywhere-packages-55c5797f9-
trgp6	1/1 Running 0 6d17h
eks-a-system	rufio-controller-manager-7ccbf44cb-
f9mpt	1/1 Running 0 6d17h
etcdadm-bootstrap-provider-system	etcdadm-bootstrap-provider-
controller-manager-6c6f479f84-ctgmr	1/1 Running 0 6d17h
etcdadm-controller-system	etcdadm-controller-controller-
manager-9dbb4669b-8j8vq	1/1 Running 0 6d17h

## Validating the cluster deployment

The following steps describe the validation of the cluster deployment:

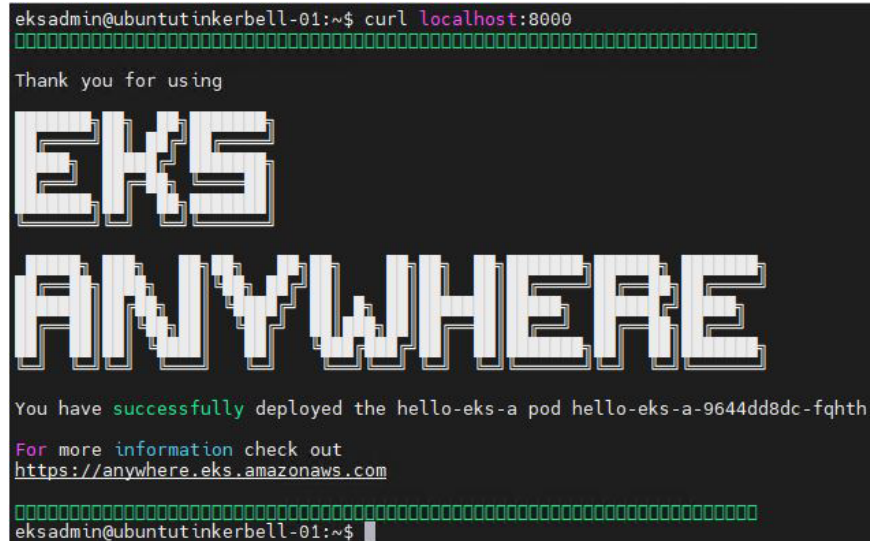
1. Run the following command to verify that your cluster is working properly:  

```
kubectl apply -f  
"https://anywhere.eks.amazonaws.com/manifests/hello-eks-a.yaml"
```
2. Run the following command to see the new pod running in your cluster:  

```
kubectl get pods -l app=hello-eks-a
```
3. There is a default web page that is served from the container. Run the following command to forward the deployment port to your local machine:  

```
kubectl port-forward deploy/hello-eks-a 8000:80
```
4. Run the following command to open a browser on your admin system or use `curl localhost:8000` to view the page example application, as shown in the following figure:

```
curl localhost:8000
```



```
eksadmin@ubuntutinkerbell-01:~$ curl localhost:8000  
████████████████████████████████████████████████████████████████████████████████  
Thank you for using  
EKS  
ANYWHERE  
You have successfully deployed the hello-eks-a pod hello-eks-a-9644dd8dc-fqth  
For more information check out  
https://anywhere.eks.amazonaws.com  
████████████████████████████████████████████████████████████████████████████████  
eksadmin@ubuntutinkerbell-01:~$
```

Figure 3. Hello EKS-Anywhere pod

# Installing the PowerFlex CSI driver

## Overview

The CSI driver for PowerFlex is a plug-in that is installed into Kubernetes to enable the automated provisioning of persistent storage using a storage class from the underlying PowerFlex storage system. The CSI driver for PowerFlex and Kubernetes communicates using the CSI protocol. The CSI driver for PowerFlex supports PV capabilities, dynamic and static PV provisioning, and snapshot capabilities.

CSI drivers can be installed using two options:

- Installation of CSI Drivers using Helm
- Installation of CSI drivers using Dell CSI Operator

For this document, we have considered using CSI drivers using Helm.

The [Helm chart](#) can be used to install the CSI driver for Dell PowerFlex using a shell script. This script installs the CSI driver container image along with the required Kubernetes sidecar containers.

The controller section of the Helm chart installs the following components in a single deployment in the `vxflexos` namespace:

- CSI driver for Dell PowerFlex
- Kubernetes Provisioner that provisions the volumes
- Kubernetes Attacher that attaches the volumes to the containers
- Kubernetes Snapshotter that provides snapshot support
- Kubernetes External Resizer that resizes the volume

The node section of the Helm chart installs the following components in a daemon set in the `vxflexos` namespace:

- CSI driver for Dell PowerFlex
- Kubernetes Node Registrar that handles the driver registration

## Prerequisites

Before installing the CSI Driver for PowerFlex, ensure that you perform the following tasks:

- Install Helm 3.
- Enable Zero Padding on PowerFlex.
- Install PowerFlex SDC on all Kubernetes nodes.

For information about SDC installation, see <https://dell.github.io/csm-docs/docs/csdriver/installation/helm/powerflex/#manual-sdc-deployment>.

- Create a Kubernetes secret for PowerFlex credentials:

```
kubectl create secret generic vxflexos-config -n vxflexos --from-file=config=samples/config.yaml
```

For more information about the Kubernetes secret for PowerFlex credentials, see <https://dell.github.io/csm-docs/docs/csdriver/installation/helm/powerflex/>.



- Get the Helm values for the PowerFlex system name or ID, default gateway, and MDM IP addresses, and default storage pool.

For more information, see

<https://dell.github.io/csm-docs/docs/csidriver/installation/helm/powerflex/#install-the-driver>.

- Ensure that the volume snapshot requirements are met.

For information about volume snapshot, see <https://dell.github.io/csm-docs/docs/csidriver/installation/helm/powerflex/#volume-snapshot-crds>

## Installing the PowerFlex CSI driver

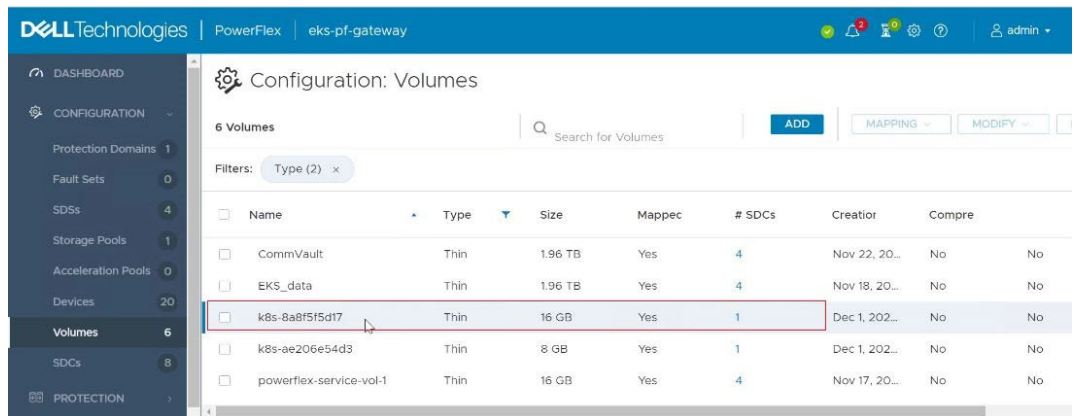
To install the PowerFlex CSI driver, follow these steps:

1. Run the following command to download the installation source files from GitHub:  
`$ git clone https://github.com/dell/csi-vxflexos`
2. Run the following command to create a namespace called `vxflexos`:  
`$ kubectl create namespace vxflexos`
3. Run the following script to collect information from the PowerFlex SDC:  
`get_vxflexos_info.sh`.
4. Copy the `csi-vxflexos/values.yaml` script into a `myvalues.yaml` file in the same directory as the `csi-install.sh` script.
5. Edit the `myvalues.yaml` file to set the parameters such as file system types, volume name prefix, and controller count for the installation.
6. Create a `config.json` file for driver configuration. This file contains information such as the PowerFlex system IP details and credentials.
7. Run the following command to proceed with the installation:  
`$ sh csi-install.sh --namespace vxflexos -values myvalues.yaml`
8. Run the following command to check the `vxflexos` namespace for running pods to verify it is deployed correctly:  
`$ kubectl get pods -n vxflexos`  
For more information about CSI driver installation, see [GitHub](#).
9. Storage classes can be created through `kubectl` commands. For more information about creating storage classes, see [Storage Classes](#).  
`$ kubectl apply -f storageclass.yaml`  
`$ kubectl apply -f storageclass-xfs.yaml`
10. Ensure that the PowerFlex CSI driver is running on the Amazon EKS-A Cluster, as shown in the following status example:

```
eksadmin@ubuntu-tinkerbell-01:~/csi-powerflex/dell-csi-helm-installer$ kubectl get pods -n vxflexos
NAME                                READY   STATUS    RESTARTS   AGE
vxflexos-controller-57b5f5588b-b9bkr 5/5     Running   8 (73s ago) 95s
vxflexos-node-h4mqb                   2/2     Running   0           95s
eksadmin@ubuntu-tinkerbell-01:~/csi-powerflex/dell-csi-helm-installer$
```

Figure 4. CSI pod status

11. Test the deployment workflow of a simple pod on PowerFlex storage. For more information, see [Test PowerFlex CSI Driver](#).
12. Ensure that the persistent volume is created in the PowerFlex cluster using the PowerFlex UI, as shown in the following figure:



**Figure 5. PowerFlex UI**

## Deployment tests and validation

### Overview

This section provides a detailed description of the tests that were performed to validate the Amazon EKS Anywhere cluster hosted on PowerFlex.

### Use case 1: Provision PVC volumes with PowerFlex CSI driver

This use case describes provisioning of the PVC volumes with the PowerFlex CSI driver:

- The CSI Functionality test scripts are used for this test case. When the CSI driver is deployed using Helm, it provides several test scripts which can be used to ensure that it is functioning correctly.
- The tests that are run in the following example, creates two PVs of 16 GB each. To monitor the results, view the volume section of the PowerFlex GUI. The following YAML sample is used for deploying a Pod with a Persistent Volume Claim (PVC).

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvol0
  namespace: helmtest-vxflexos
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 16Gi
  storageClassName: vxflexos
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvol1
  namespace: helmtest-vxflexos
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 16Gi
  storageClassName: vxflexos-xfs
```

- The `kubectl get pvc` command retrieves the status of the claim, as shown in the following sample:

```
$ kubectl get pvc -n helmtest-vxflexos
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
pvol0	Bound	k8s-2cc5c6df9d	16Gi	RWO	vxflexos	22d
pvol1	Bound	k8s-9b25ab5be5	16Gi	RWO	vxflexos-xfs	22d

- Verify the volumes in the PowerFlex Storage UI, as shown in the following figure:

Name	Type	Size	Mappe	# SDCs	Creatio	Comprn	R
k8s-2cc5c6df9d	Thin	16 GB	Yes	1	Jun 20, 2...	No	No
k8s-8a8f5f5d7	Thin	16 GB	Yes	1	Dec 1, 20...	No	No
k8s-9b25ab5be5	Thin	16 GB	Yes	1	Jun 20, 2...	No	No
k8s-ae206e54d3	Thin	8 GB	Yes	1	Dec 1, 20...	No	No

Figure 6. Dell PowerFlex storage UI

## Use case 2: Extend PVC volumes with PowerFlex CSI

This use case describes the extension of PVC volumes with PowerFlex CSI:

- A Pod with a PVC volume of 8 GB is created In the lab environment.
- PVC is extended with the `kubectl patch pvc` command.
- The activity is monitored from the **Volumes** section of the PowerFlex UI, as shown in the following figure:

Name	Type	Size	Mappec	# SDCs	Creator	Compre
CommVault	Thin	1.96 TB	Yes	4	Nov 22, 20...	No
EKS_data	Thin	1.96 TB	Yes	4	Nov 18, 20...	No
k8s-8a8f5f5d7	Thin	8 GB	Yes	1	Dec 1, 202...	No
k8s-ae206e54d3	Thin	8 GB	Yes	1	Dec 1, 202...	No
powerflex-service-vol-1	Thin	16 GB	Yes	4	Nov 17, 20...	No

Figure 7. PowerFlex PVC volume

- The `kubectl get pvc` command provides the status of the claim and the `kubectl patch pvc` command extends the claim, as shown in the following example:

```
$ kubectl get pvc
NAME                                STATUS      VOLUME             CAPACITY
ACCESS                              MODES      STORAGECLASS        AGE
mysql-pv-claim                      Bound      k8s-8asf515d17     8Gi
RWO                                 vxflexos   129m
mysql-pv-claim-snap1                Bound      k8s-ae206454d3     8Gi
RWO                                 vxflexos   133m
$ kubectl patch pvc mysql-pv-claim -p
'{"resources":{"requests":{"storage":"16Gi"}}}'
persistentvolumeclaim/mysql-pv-claim patched
```

- Verify the PVC expansion on the PowerFlex Storage UI, as shown in the following figure:

Name	Type	Size	Mappec	# SDCs	Creator	Compre
CommVault	Thin	1.96 TB	Yes	4	Nov 22, 20...	No
EKS_data	Thin	1.96 TB	Yes	4	Nov 18, 20...	No
k8s-8a8f515d17	Thin	16 GB	Yes	1	Dec 1, 2022...	No
k8s-ae206e54d3	Thin	8 GB	Yes	1	Dec 1, 2022...	No
powerflex-service-vol-1	Thin	16 GB	Yes	4	Nov 17, 20...	No

Figure 8. Dell PowerFlex storage volume

### Use case 3: Snapshot with PowerFlex CSI

This use case describes how to take a snapshot with PowerFlex CSI:

- The CSI Functionality test scripts are used for this test case. When the CSI driver is deployed using Helm, it provides several test scripts which can be used to ensure that it is functioning correctly.
- The tests run in the following example, creates two persistent volumes and creates snapshots then deletes snapshots of the volumes. The following YAML file is used to deploy the Pod with the PVC:

```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: vxflextest
  namespace: helmtest-vxflexos
---
kind: StatefulSet
apiVersion: apps/v1
metadata:
  name: vxflextest
  namespace: helmtest-vxflexos
spec:
  selector:
    matchLabels:
      app: vxflextest
  serviceName: 2vols
  template:
    metadata:
      labels:
        app: vxflextest
    spec:
      serviceAccount: vxflextest
      containers:
        - name: test
          image: docker.io/centos:latest
          command: [ "/bin/sleep", "3600" ]
          volumeMounts:
            - mountPath: "/data0"
              name: pvol0
            - mountPath: "/data1"
              name: pvol1
      volumes:
        - name: pvol0
          persistentVolumeClaim:
            claimName: pvol0
        - name: pvol1
          persistentVolumeClaim:
            claimName: pvol1

```

The following YAML file is used to deploy the Pod 1 with the PVC:

```

apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvol0-snap1
  namespace: helmtest-vxflexos
spec:
  volumeSnapshotClassName: vxflexos-snapclass
  source:
    persistentVolumeClaimName: pvol0

```

The following YAML file is used to provision the PVC:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: pvol0-snap2
  namespace: helmtest-vxflexos
spec:
  volumeSnapshotClassName: vxflexos-snapclass
  source:
    persistentVolumeClaimName: pvol0
```

The following script is used for the Snapshot test.

```
#!/bin/bash
NS=helmtest-vxflexos
source ./common.bash

echo "creating snap1 of pvol0"
kubectl create -f snap1.yaml
sleep 10
kubectl get volumesnapshot -n ${NS}
kubectl describe volumesnapshot -n ${NS}
sleep 10
echo "creating snap2 of pvol0"
kubectl create -f snap2.yaml
sleep 10
kubectl describe volumesnapshot -n ${NS}
sleep 10
echo "deleting snapshots..."
kubectl delete volumesnapshot pvol0-snap1 -n ${NS}
sleep 10
kubectl delete volumesnapshot pvol0-snap2 -n ${NS}
sleep 10
kubectl get volumesnapshot -n ${NS}
```

The following sample shows the output of the test:

```

$ bash ./snaptest.sh
creating snap1 of pvol0
volumesnapshot.snapshot.storage.k8s.io/pvol0-snap1 created
NAME          READYTOUSE  SOURCEPVC  SOURCESNAPSHOTCONTENT
RESTORESIZE   SNAPSHOTCLASS  SNAPSHOTCONTENT
CREATIONTIME  AGE
pvol0-snap1          pvol0
vxflexos-snapclass          10s
Name:          pvol0-snap1
Namespace:     helmtest-vxflexos
Labels:        <none>

Annotations:   <none>
API Version:   snapshot.storage.k8s.io/v1
Kind:          VolumeSnapshot
Metadata:
  Creation Timestamp:  2022-07-13T15:51:57Z
  Generation:         1
  Managed Fields:
    API Version:  snapshot.storage.k8s.io/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:spec:
        .:
        f:source:
          .:
          f:persistentVolumeClaimName:
          f:volumeSnapshotClassName:
      Manager:      kubect1-create
      Operation:    Update
      Time:         2022-07-13T15:51:57Z
      Resource Version:  7151834
      UID:           fe7fc0f8-120b-4404-bca9-f79ededc1cb
  Spec:
    Source:
      Persistent Volume Claim Name:  pvol0
      Volume Snapshot Class Name:    vxflexos-snapclass
  Events:                           <none>
creating snap2 of pvol0
volumesnapshot.snapshot.storage.k8s.io/pvol0-snap2 created
Name:          pvol0-snap1
Namespace:     helmtest-vxflexos
Labels:        <none>
Annotations:   <none>
API Version:   snapshot.storage.k8s.io/v1
Kind:          VolumeSnapshot
Metadata:
  Creation Timestamp:  2022-07-13T15:51:57Z
  Generation:         1
  Managed Fields:
    API Version:  snapshot.storage.k8s.io/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:spec:
        .:
        f:source:
          .:

```



```

      f:persistentVolumeClaimName:
      f:volumeSnapshotClassName:
    Manager:      kubect1-create
    Operation:    Update
    Time:         2022-07-13T15:51:57Z
    Resource Version: 7151834
    UID:          fe7fc0f8-120b-4404-bca9-f79ededc1cb
  Spec:
    Source:
      Persistent Volume Claim Name:  pvol0
      Volume Snapshot Class Name:    vxflexos-snapclass
  Events:      <none>

Name:      pvol0-snap2
Namespace: helmtest-vxflexos
Labels:    <none>
Annotations: <none>
API Version: snapshot.storage.k8s.io/v1
Kind:      VolumeSnapshot
Metadata:
  Creation Timestamp: 2022-07-13T15:52:17Z
  Generation:         1
  Managed Fields:
    API Version: snapshot.storage.k8s.io/v1
    Fields Type: FieldsV1
    fieldsV1:
      f:spec:
        .:
        f:source:
          .:
          f:persistentVolumeClaimName:
          f:volumeSnapshotClassName:
        Manager:      kubect1-create
        Operation:    Update
        Time:         2022-07-13T15:52:17Z
        Resource Version: 7151901
        UID:          b915f17e-5f15-48ab-a4c7-25c3901e97ca
  Spec:
    Source:
      Persistent Volume Claim Name:  pvol0
      Volume Snapshot Class Name:    vxflexos-snapclass
  Events:      <none>
deleting snapshots...
volumesnapshot.snapshot.storage.k8s.io "pvol0-snap1" deleted
volumesnapshot.snapshot.storage.k8s.io "pvol0-snap2" deleted
No resources found in helmtest-vxflexos namespace.

```

# Amazon EKS Connector

## Overview

Amazon Elastic Kubernetes Services (EKS) Connector is a new capability from Amazon Web Services that can be used to connect any Kubernetes cluster to the EKS console. Amazon EKS Connector can be used to get a unified view of the Kubernetes cluster in the environment. The user can connect any Kubernetes cluster including EKS Anywhere running in on-premises, self-managed Kubernetes clusters on EC2, and other Kubernetes clusters running outside of AWS to the EKS console.

This feature enables the customers to view all the connected Kubernetes clusters and provides a holistic view of the resources including Compute CPU, memory, pods, and so on. Customers can monitor the performance and utilization of their resources across the Kubernetes environment.

The Kubernetes cluster can be registered with the Amazon EKS Connector in multiple ways using AWS CLI, SDK, eksctl, or the AWS management console.

For more information, see [Amazon EKS Connector](#)

## Connecting to an EKS Anywhere cluster to AWS EKS Connector

The following steps describe how to connect an on-premises Amazon EKS Anywhere cluster that is deployed with PowerFlex storage to the AWS EKS Connector.

### Step 1- Preparations

Before getting started the EKS Connector registration, the necessary roles and policies in AWS IAM need to be created.

- a. Log in to your Amazon account using the administrator with enough permissions to manage EKS and IAM resources. You must create the following two roles for this connection to work:
  - i Service-linked role for Amazon EKS
  - ii EKS-Connector-agent role

**Service-linked role** – This role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services.

**EKS-connector-agent Role** – This IAM role allows the EKS connector agent to interact with AWS Systems Manager Agent (SSM) service. This IAM role is used by the EKS Connector agent on the Kubernetes cluster to connect to the SSM service on AWS.

- b. Create a policy document to associate it with the role. The permission that is associated with this policy enables the connector to talk back with the SSM service. When you navigate through the EKS console, The console goes through EKS and SSM service until it finally communicates with the agent running inside your cluster.

For more information about creating the role and policy, see the [appendix section](#).

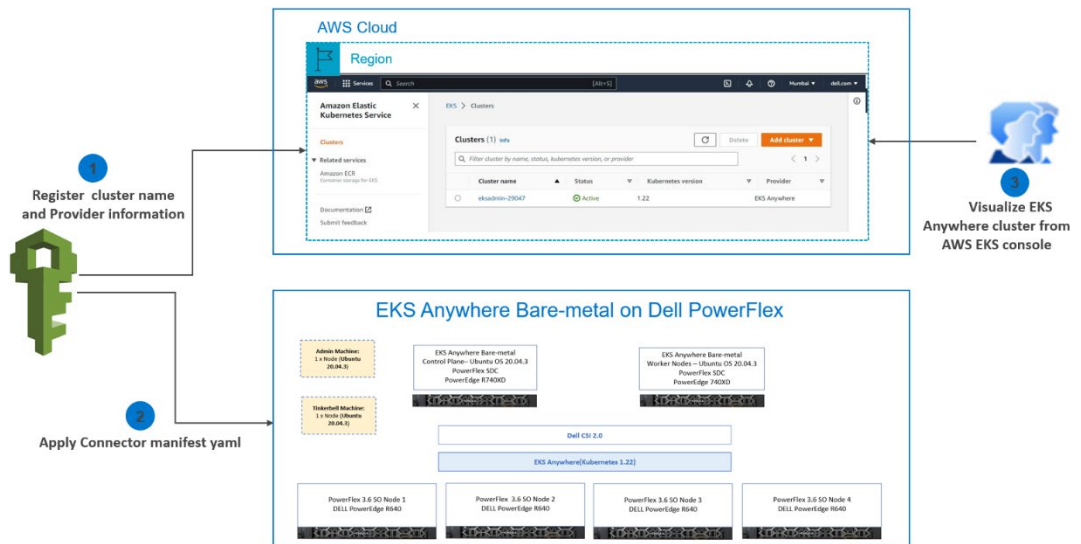


Figure 9. Amazon EKS Connector

## Step 2 - EKS Connector Cluster Registration

The cluster registration process involves two steps:

1. Register the cluster with Amazon EKS.
2. Applying a connector YAML manifest file in the target cluster to enable connectivity.

### Register the cluster with Amazon EKS

1. The EKS console in AWS includes a register option along with the create cluster option. Open the EKS console and go to the **Clusters** section. Select the **Register** from the **AddCluster** dropdown.

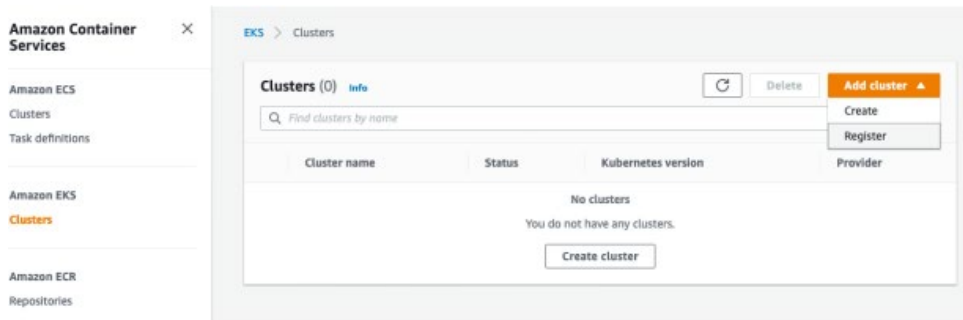


Figure 10. Registering EKS Connector

2. Enter the details in the Register Cluster form as shown in the following figure:
  - a. Define a name for the cluster.
  - b. Select the Provider as EKS Anywhere.
  - c. Select the EKS Connector Role to provide Kubernetes control plane to create the resources on your behalf. If the role is precreated, it is loaded, and ready to select.
  - d. Click **Register cluster**.

## Register cluster

### Cluster configuration

**Name**  
Enter a unique name for this cluster. This property cannot be changed after the cluster is created.

**Provider** [Info](#)  
Please select provider that hosts this cluster

### Connection configuration

**EKS connector role** [Info](#)  
Select the IAM role with which the EKS Connector uses to authenticate with AWS. To create a new role, go to the [IAM console](#).

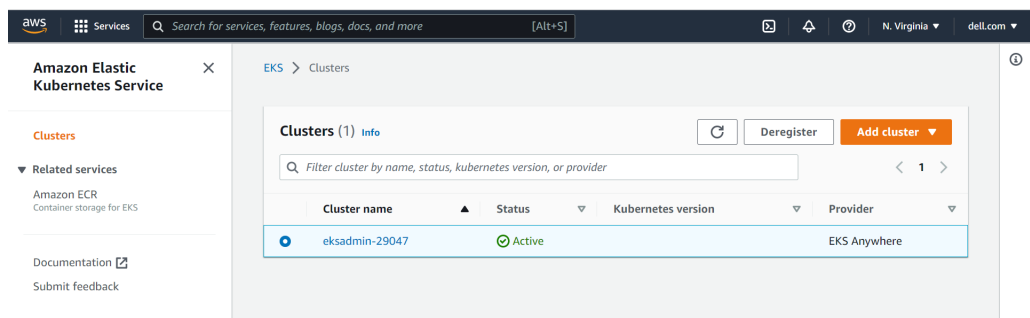
### Tags (0) [Info](#)

This cluster does not have any tags.

Remaining tags available to add: 50

**Figure 11. Registering the cluster**

- Once the cluster is added, the cluster name is displayed, and its status shows as **Active** as shown in the following figure:



**Figure 12. Clusters status**

- After registering the cluster, you will be redirected to the Cluster Overview page. Click **Download YAML file** to get the Kubernetes configuration file to deploy all the necessary infrastructure to connect your cluster to EKS.



**Figure 13. Cluster Registration**

### *Applying the connector YAML manifest file to the target*

Run the following command to apply the downloaded `eks-connector.yaml`.

```
$ kubectl apply -f eks-connector.yaml
```

The EKS Connector runs as a StatefulSet on your Kubernetes cluster. It establishes a connection and proxies the communication between the API server of your EKS Anywhere cluster and Amazon Web Services. It does this connection to display cluster data in the Amazon EKS console until you disconnect the cluster from AWS.

The manifest file that is generated from registering a cluster contains the following components:

**InitContainer:** This container registers the EKS Connector agent with the Systems Manager control plane service and persists the registration information in the Kubernetes backend data store. InitContainer mounts this data to the EKS Connector agent volume when it is recycled. This eliminates the need for registration whenever a pod is recycled.

**EKS Connector agent:** This is an agent based on the SSM agent, running in container mode. This agent creates an outbound connection from the Kubernetes cluster to the AWS network. All subsequent requests from AWS are performed using the connection channels that are established by the EKS Connector agent.

**Connector proxy:** This agent acts as a proxy between the EKS Connector agent and Kubernetes API Server. This proxy agent uses the Kubernetes service account to impersonate the IAM user that accesses the console and fetches information from the Kubernetes API Server.

The EKS connector agent interacts with the SSM service, which in turn interacts with EKS service using EventBridge. To facilitate these interactions, the EKS connector agent role is required with appropriate permissions to create, open, and control the SSM channels.

Upon successful registration, the changes can be seen in the AWS EventBridge services. A new event rule with the pattern of registration and deregistration is created under the “default” event bus.

### Step 3 - Permission to view Kubernetes resources of connected cluster from AWS Console.

#### *Granting access to a user to view the Kubernetes resources of a connected cluster from the AWS Console.*

This is a YAML consisting of cluster roles and bindings for the cluster to be registered. It gives access to all namespaces and resources that can be visualized in the console.

Download and apply the `eks-connector-console-dashboard-full-access.yaml` file:

```
$ curl -o eks-connector-console-dashboard-full-access-group.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-console-dashboard-full-access-group.yaml

$ kubectl apply -f eks-connector-console-dashboard-full-access.yaml
```

#### *Granting access to IAM user or Role to view Kubernetes resources in Amazon EKS console*

This is a YAML consisting of cluster roles and bindings for the cluster to be registered.

Download and apply `eks-connector-clusterrole`:

```
$ curl -o eks-connector-clusterrole.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-clusterrole.yaml

$ kubectl apply -f eks-connector-clusterrole.yaml
```

## Appearance of the EKS Console

**Dashboard:** After the Kubernetes cluster has been registered successfully with the EKS Connector, the EKS Cluster Overview section shows all the Cluster resources. All the objects are read-only and the user cannot edit or delete an object in the registered cluster. The following figure shows the dashboard of the EKS console:

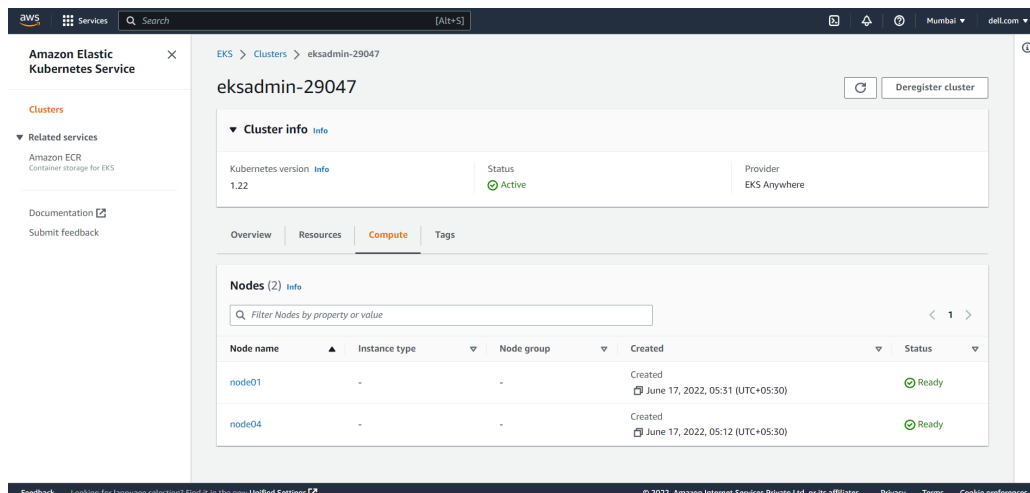


Figure 14. Overview Dashboard

**Compute:** The Compute section shows all the Node resources in the EKS Anywhere Cluster as shown in the following figure:

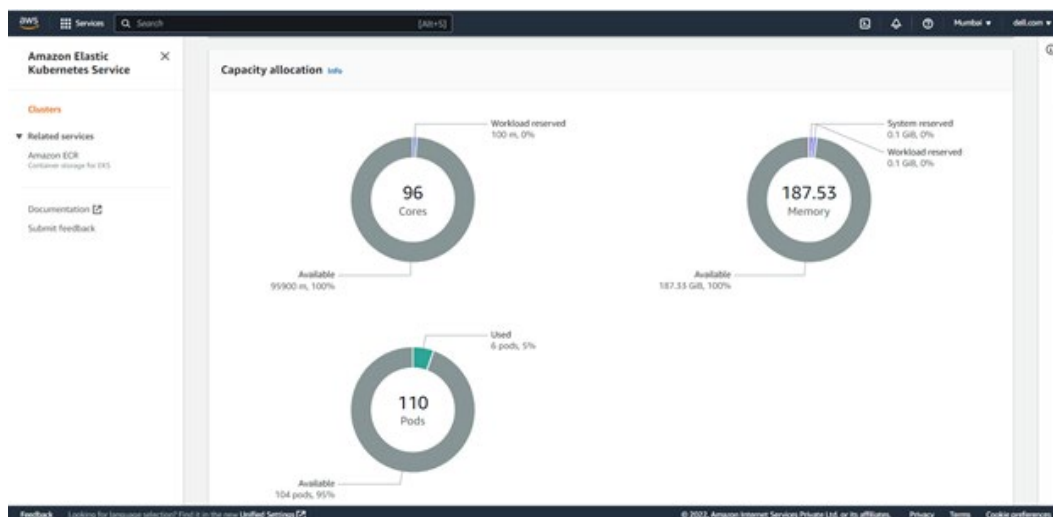
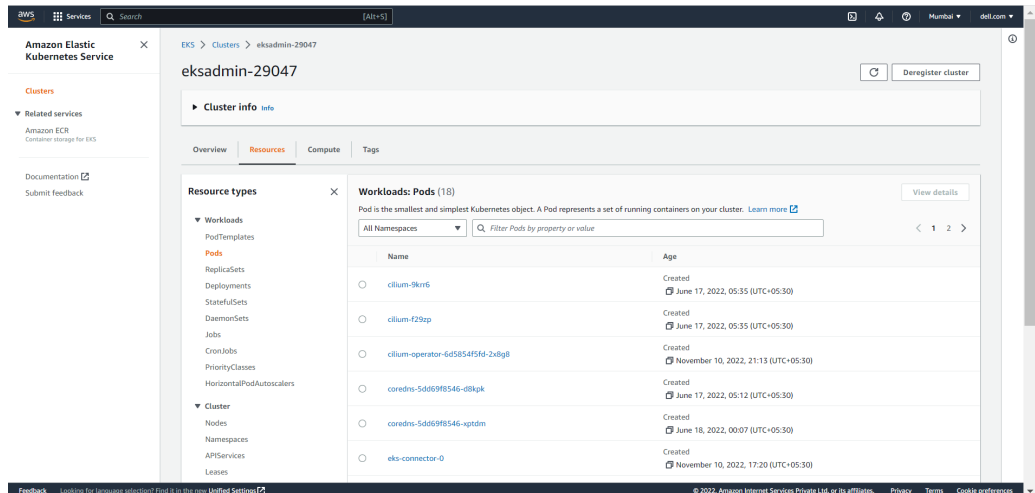


Figure 15. Compute

**Workloads:** The Workloads section displays all objects of Type: Deployment, DaemonSet, and StatefulSet. Users can select these objects to select a pod-level overview as shown in the following figure:



**Figure 16. Workloads**

## Amazon EKS Connector troubleshooting

Steps to troubleshoot the Amazon EKS Connector:

1. Error loading GenericResourceCollection/Namespaces.

Amazon EKS is unable to communicate with your Kubernetes cluster API server. The cluster must be in an ACTIVE state for successful connection. Try again in a few minutes.

- a. Run the following command to check for the connector pod status:

```
$ kubectl get pods -n eks-connector
```

- b. Run the following command to verify the Amazon EKS Connector logs:

```
$ kubectl logs eks-connector-0 --container connector-init
-n eks-connector
$ kubectl logs eks-connector-1 --container connector-init
-n eks-connector
```

- c. If the pod eks-connector-0 is not able to communicate with EKS-connector-1. Then delete the eks-connector pod and registered secret and reapply the Connector.yaml as follows:

```
$ kubectl apply -f connector.yaml
```

2. If you receive this error: An error occurred (SignatureDoesNotMatch) when calling the CreateRole operation: Signature not yet current: 20220826T144825Z is still later than 20220826T093628Z (20220826T092128Z + 15 min.)

Run the following command to check the date and time on all the nodes and Amazon EKS connector pods.

```
$ kubectl exec eks-connector-0 --container connector-agent
-n eks-connector -- cat /var/log/amazon/ssm/amazon-ssm-agent.log
```

Stop and start the container service.

3. If you get the following error: Error loading namespaces..users "arn:aws:iam::xxxxxxxxxxxxxxxx" is forbidden: User "system:serviceaccount:eks-connector:eks-connector" cannot impersonate resource "users" in API group "" at the cluster scope.



Download the YAML file and update with %IAM\_ARN% user field as follows:

```
$ curl -o eks-connector-clusterrole.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-clusterrole.yaml

$ kubectl apply -f eks-connector-clusterrole.yaml
```

4. If you get the following error: Error loading namespaces: "Namespace is forbidden :user "users "arn:aws:iam::xxxxxxxxxxxxxxxxxxxxx" cannot list resources "namespaces" in API group at the cluster.

Download the YAML file and update the YAML with %IAM\_ARN% user field as follows:

```
$ curl -o eks-connector-console-dashboard-full-access-group.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-console-dashboard-full-access-group.yaml

$ kubectl apply -f eks-connector-console-dashboard-full-access-group.yaml
```

Create the following IAM Role and Policy:

**EKS external cluster management role:** This role is used by the EKS control plane to manage the AWS resources on behalf of the customer.

**Define the trust policy:** Steps to define the trust policy:

1. Define the policy document.

```
$ cat <<EOF > integration-trust-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EKSCheckAccess",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "eks-connector.amazonaws.com",
          "eks.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

```
$ cat <<EOF > integration-policy-document.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AccessSSMService",
      "Effect": "Allow",
      "Action": [
        "ssm:CreateActivation",
        "ssm:DescribeInstanceInformation",
        "ssm>DeleteActivation"
      ],
      "Resource": "*"
    },
    {
      "Sid": "ConnectorAgentStartSession",
      "Effect": "Allow",
      "Action": [
        "ssm:StartSession"
      ],
      "Resource": [
        "arn:aws:eks:*:*:cluster/*"
        "arn:aws:ssm:*:*:document/AmazonECS-ExecuteInteractiveCommand",
        "arn:aws:ssm:*:*:document/AmazonEKS-ExecuteNonInteractiveCommand"
      ]
    },
    {
      "Sid": "ConnectorAgentDeregister",
      "Effect": "Allow",
      "Action": [
        "ssm:DeregisterManagedInstance"
      ],
      "Resource": [
        "arn:aws:eks:*:*:cluster/*"
      ]
    },
    {
      "Sid": "PassAnyRoleToSsm",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": [
            "ssm.amazonaws.com"
          ]
        }
      }
    }
  ]
}
EOF
```

## 2. Create the IAM role:

```
$ aws iam create-role \
  --role-name eks-external-cluster-integration \
  --assume-role-policy-document file://integration-trust-policy.json
```

3. Attach the policy to the role:

```
$ aws iam put-role-policy \  
--role-name eks-external-cluster-integration \  
--policy-name eks-connector-policy \  
--policy-document file://integration-policy-document.json
```

4. Define the Amazon EKS Connector agent role:

This role is used by the EKS connector agent to communicate back to AWS from the Kubernetes cluster.

Define the trust policy:

```
$ cat <<EOF > agent-trust-policy.json  
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "SSMAccess",  
      "Effect": "Allow",  
      "Principal": {  
        "Service": [  
          "ssm.amazonaws.com"  
        ]  
      },  
      "Action": "sts:AssumeRole"  
    }  
  ]  
}  
EOF
```

5. Define the policy document:

```
$ cat <<EOF > agent-policy-document.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SsmControlChannel",
      "Effect": "Allow",
      "Action": [
        "ssmmessages:CreateControlChannel"
      ],
      "Resource": "arn:aws:eks:*:*:cluster/*"
    },
    {
      "Sid": "ssmDataplaneOperations",
      "Effect": "Allow",
      "Action": [
        "ssmmessages:CreateDataChannel",
        "ssmmessages:OpenDataChannel",
        "ssmmessages:OpenControlChannel"
      ],
      "Resource": "*"
    }
  ]
}
EOF
```

#### 6. Create the Amazon EKS Connector agent role:

```
$ aws iam create-role \
  --role-name eks-connector-agent \
  --assume-role-policy-document file://agent-trust-
policy.json
```

#### 7. Attach the policy to the Amazon EKS Connector agent role:

```
$ aws iam put-role-policy \
  --role-name eks-connector-agent \
  --policy-name eks-connector-agent-policy \
  --policy-document file://agent-policy-document.json
```

## Conclusion

This solution shows how Amazon EKS Anywhere Bare Metal can be used with the PowerFlex storage family to create and manage on-premises Kubernetes clusters using Amazon tools on customer PowerFlex infrastructure. Organizations can use the Kubernetes containers environment together with the performance, scale, management, and storage APIs that are provided by PowerFlex storage. This combination of PowerFlex and EKS Anywhere brings together key elements of a true hybrid cloud with ease of deployment, flexible design, and high performance at any scale.

## Hardware qualification details

### Storage-only nodes

The following table provides the configuration details of the Dell PowerEdge R640 server storage-only nodes:

**Table 3. Storage Only nodes**

Hardware	Configuration
CPU Cores	2 x Intel Xeon Gold 6248 R CPU @ 3.00 GHz
Memory	192 GB (12 x 16 GB, 2933 MHz)
NIC	2 x Mellanox ConnectX-4 LX 25 GbE SFP Adapter
	1 x Intel Ethernet 10G 4P X710/I350 rNDC
Storage	BOSS S1 Controller 2 x 240 GB SATA SSD
	Dell HBA330 controller 8 x 1.9 TB SAS SSD
Operating system	Embedded operating system
Dell PowerFlex	3.6.0.2

### Compute-only nodes

The following table provides the configuration details of the Dell PowerEdge R740xd server compute-only nodes:

**Table 4. Compute Only nodes**

Hardware	Configuration
CPU Cores	2 x Intel Xeon Gold 6248 R CPU @ 3.00 GHz
Memory	192 GB (12 x 16 GB, 2933 MHz)
NIC	2 x Mellanox ConnectX-4 LX 25 GbE SFP Adapter
	1 x Intel Ethernet 10G 4P X710/I350 rNDC
Storage	BOSS S1 Controller 2 x 240 GB SATA SSD
	Dell HBA330 controller 8 x 1.9 TB SAS SSD
Operating system	Embedded operating system
Dell PowerFlex	3.6.0.2

## Amazon EKS Anywhere configuration details

The following tables provide the configuration details of an EKS Anywhere management:

**Table 5. EKS Anywhere Admin machine details**

Components	Items	Details
Admin Machine	CPU	2 x Intel(R) Xeon(R) Silver 4112 CPU @ 2.60 GHz
	Memory	192 GB
	Hard disk	2 x 223.57 SATA SSD 2 x 447.13 GB (SATA SSD)
	NIC	Network rNDC - Intel(R) 10 GbE 4P X710 rNDC
Software	Operating system	Ubuntu 20.04.3 LTS
	Container Runtime	Docker 20.10.16
	Docker composer	V2.5.0
	EKS Anywhere	v0.1
	Kubernetes	v1.22.9
	EKS-A CLI	0.74.0
	HELM	V3.9
	Dell CSI	v2.0
Network Access	<ul style="list-style-type: none"> <li>public.ecr.aws</li> <li>api.github.com</li> <li>eks-anywhere-beta.s3.amazonaws.com</li> <li>distro.eks.amazonaws.com</li> </ul>	Network access To download manifests, OVAs, and EKS distro.

**Table 6. Tinkerbelle machine details**

Components	Items	Details
Admin Machine	CPU	2 x Intel(R) Xeon(R) Silver 4112 CPU @ 2.60 GHz
	Memory	192 GB
	Hard disk	2 x 223.57 SATA SSD 2 x 447.13 GB (SATA SSD)
	NIC	Network rNDC - Intel(R) 10 GbE 4P X710 rNDC
Software	Operating system	Ubuntu 20.04.3 LTS
	Docker	Docker 20.10.16
	Docker composer	V2.5.0

Components	Items	Details
Network Access	<ul style="list-style-type: none"> <li>• <a href="#">public.ecr.aws</a></li> <li>• <a href="#">api.github.com</a></li> <li>• <a href="#">eks-anywhere-beta.s3.amazonaws.com</a></li> <li>• <a href="#">distro.eks.amazonaws.com</a></li> </ul>	<p>Network access</p> <p>To download manifests, OVAs, and EKS distro.</p>

## References

### Dell Technologies documentation

The following Dell Technologies documentation provides additional information. Access to these documents depends on your login credentials. If you do not have access to a document, contact your Dell Technologies representative.

- [PowerFlex Overview – Video](#)
- [PowerFlex Specification Sheet](#)
- [PowerFlex Solutions Documents – InfoHub](#)
- [PowerFlex Networking Best Practices and Design Considerations](#)
- [PowerFlex - Storage definitions](#)
- [Dell CSI Drivers - Dell CSI Git Repo](#)

### Amazon EKS Anywhere documentation

The following Amazon EKS documentation provides additional information:

- [EKS Anywhere Overview](#)
- [EKS Anywhere Installation](#)
- [Compare EKS Anywhere and EKS](#)
- [EKS Anywhere Cluster creation workflow](#)
- [EKS Anywhere Cluster management](#)
- [EKS Anywhere Cluster Troubleshooting](#)