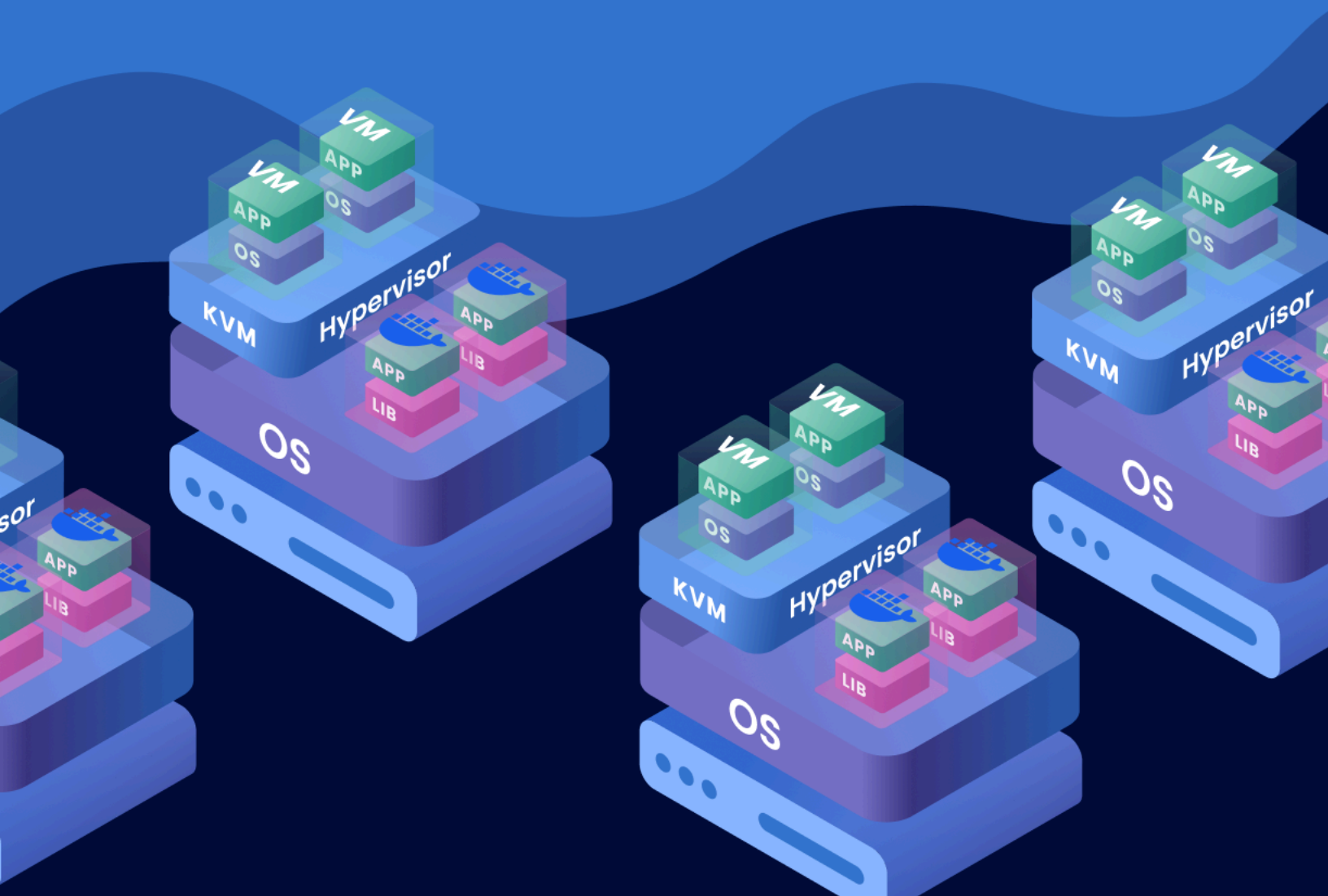
spectro cloud

Palette Virtual Machine Orchestrator (VMO)

Reference Architecture

Rev. 1.2.0 | May 2024



About this reference architecture

This document represents the first reference architecture for Palette Virtual Machine Orchestrator (VMO). It is a comprehensive explanation of the technology used, each component's purpose and configuration.

The selected vendors in this document are intended to showcase a known working solution. That does not imply that other vendors are incompatible, far from it. To aid in the selection of alternative vendors, this reference architecture provides the necessary requirements for running a Kubernetes cluster with VM workloads.

More partner validated designs are expected to be made available in the future.

Change History

Version	Release date	Change summary
1.0.0	June 2023	Initial version
1.1.0	November 2023	Upgrade to VMO 4.1
1.2.0	May 2024	Upgrade to VMO 4.3 Support boot from SAN

Introduction

This section provides the purpose, scope, and the intended audience of this document.

Purpose

This reference architecture provides a standard, repeatable and highly scalable design that can be adapted to specific environments and customer requirements. It aims at developing a scalable Kubernetes infrastructure environment capable of running virtual machines alongside containers.

Scope

This reference architecture:

- Demonstrates a platform for running virtual machines that provides the functionality that is typically required by Virtual Machine (VM) platform administrators.
- Validates the ability to provide traditional VLAN-based network connectivity for virtual machines, as well as overlay-based networking for hybrid workloads.
- Demonstrates storage performance, resilience and efficiency of Kubernetes deployments using Portworx and Pure FlashArray.

Audience

This reference architecture is intended for customers — IT architects, consultants and administrators — involved in the early phases of planning, design and deployment of Kubernetes-based VM management solutions using Spectro Cloud Palette. It is assumed that the reader is familiar with the concepts and operations of Kubernetes technologies and Spectro Cloud products.

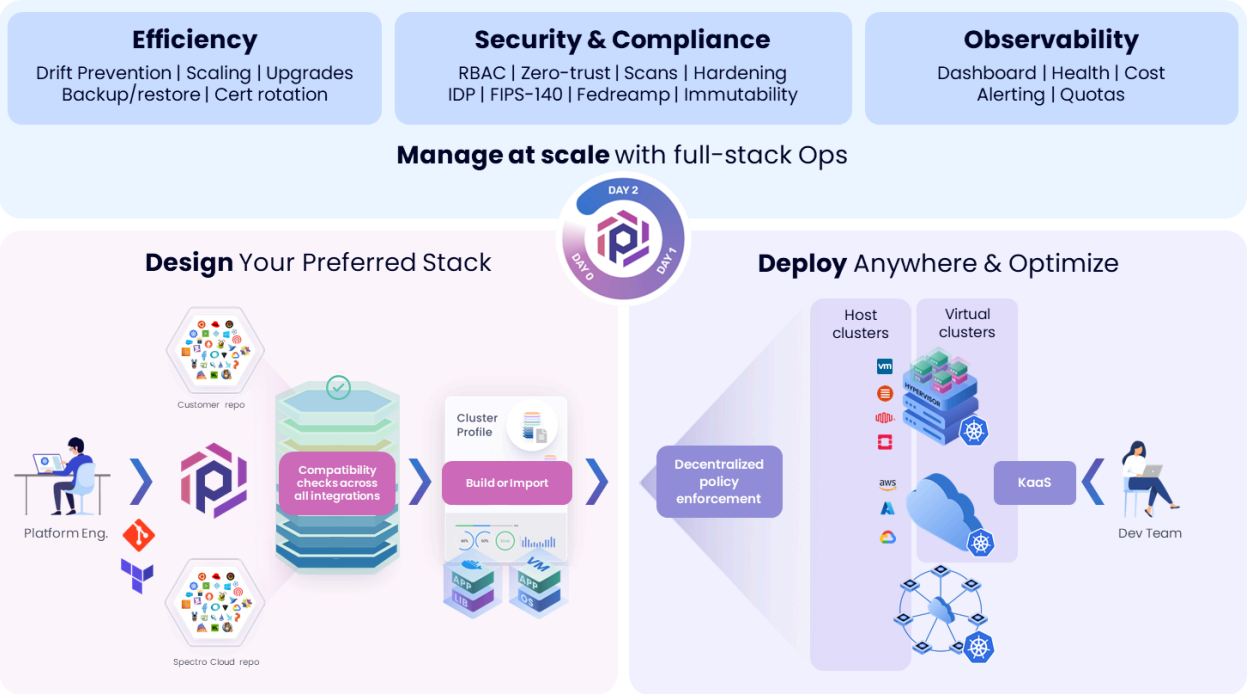
Technology Overview

Overview

This section provides an overview of the technologies that are used in this solution:

- Spectro Cloud Palette 4.3.2
- Canonical MAAS 3.4.1
- Ubuntu 22.04 LTS
- Kubernetes 1.28.3
- Cilium 1.15.3
- Portworx Enterprise 3.1.1
- Pure Storage FlashArray
- Descheduler 0.27.1
- Prometheus Grafana stack 55.8.3
- MetalLB 0.13.12
- Nginx 1.9.5
- Multus 4.0.2 with Dynamic Networks Controller
- KubeVirt 1.1.1
- KubeVirt CDI 1.58.0
- CSI Snapshot Controller 6.3.2

Spectro Cloud Palette



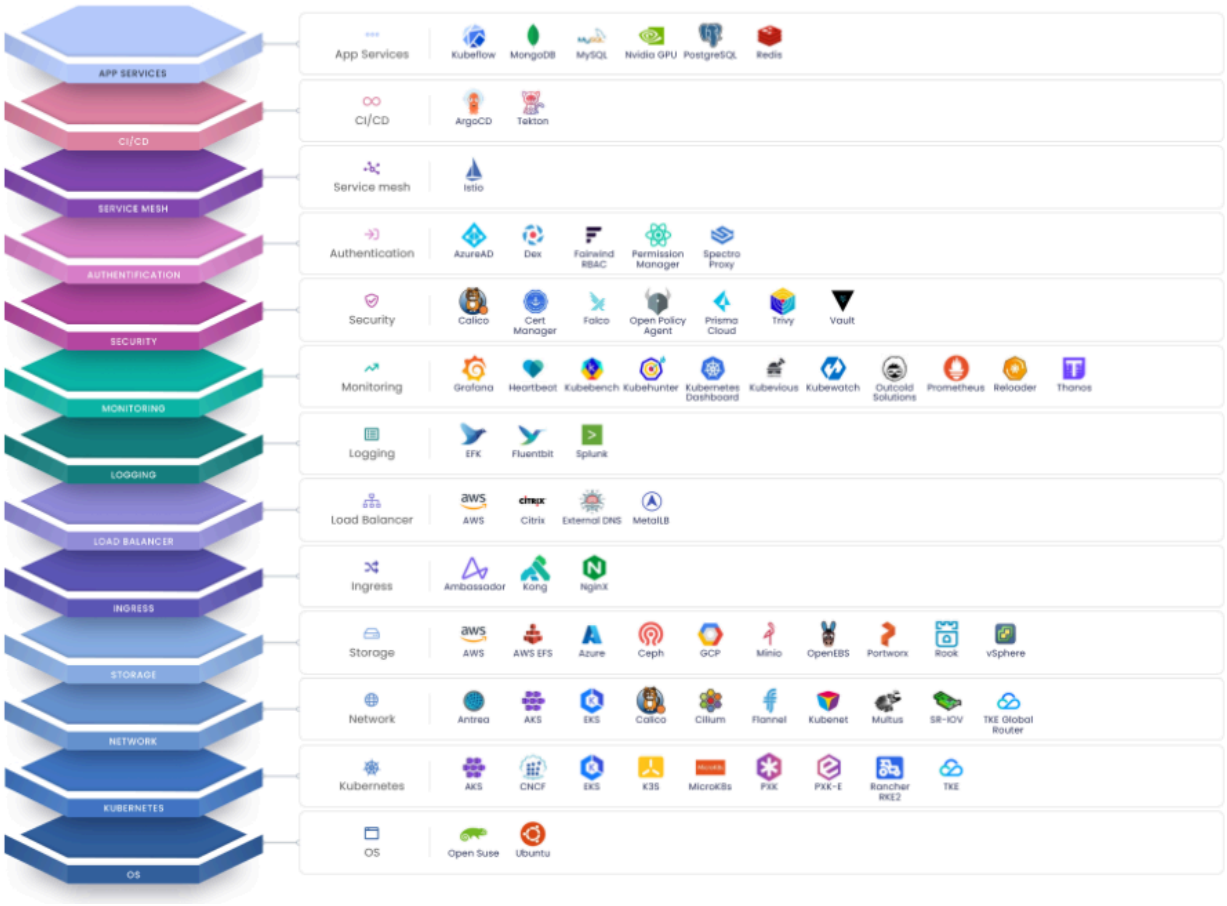
Spectro Cloud Palette automates deployment and lifecycle management of Kubernetes clusters in cloud, on-premises and edge environments.

Palette uses Cluster API to automate the deployment of Kubernetes clusters based on a desired state model.

Different providers for Cluster API enable the automation of Kubernetes cluster deployment in many different environments, from cloud providers to on-premises platforms like VMware, OpenStack and even bare metal.

While extremely helpful, Cluster API is limited to just deploying Kubernetes itself along with the network and storage plugins.

Spectro Cloud Palette extends the idea of desired state management of Kubernetes clusters to the entire stack of components that a Kubernetes cluster needs in practice. This is a technology we call **Cluster Profiles** (see our Glossary for this and other terms). Spectro Cloud provides a wide range of packs for these Cluster Profiles to aid in the configuration of a full-fledged Kubernetes cluster.



Spectro Cloud provides a pack for VM orchestration that can be added to the cluster profile. Our VM orchestration pack automates the installation and configuration of KubeVirt, KubeVirt CDI, CSI Snapshot Controller and Multus. It also installs a GUI for VM orchestration and management.

Canonical MAAS

Canonical MAAS is an open-source bare metal deployment solution that uses PXE network boot to deploy an operating system onto bare metal servers automatically. Spectro Cloud has developed a Cluster API provider for MAAS to enable fully automated deployment of Kubernetes clusters onto bare metal hardware and day-2 operations for cluster upgrades.

Ubuntu

Ubuntu 22.04 is the current OS of choice for our reference architecture. It is natively supported by Canonical MAAS, Spectro Cloud Palette and Portworx.

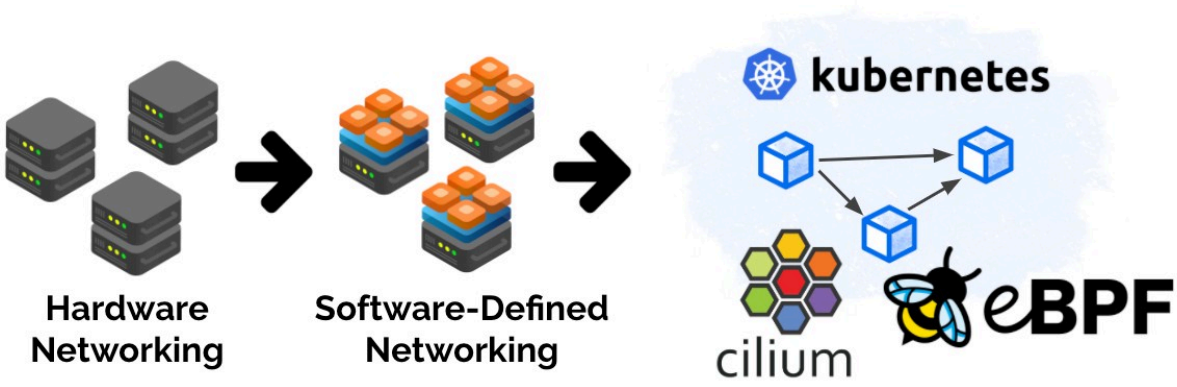
Kubernetes

Kubernetes 1.28 ensures support for all the components in the reference architecture, most importantly Spectro Cloud Palette, Portworx and KubeVirt. Every release of Kubernetes improves the capabilities for running virtual machines, hence we want to use the newest version of Kubernetes possible.

Spectro Cloud recommends keeping your Kubernetes versions current using Spectro Cloud Palette, where we validate and cross-reference all packs deployed in a Cluster Profile to provide guardrails and protection against incompatible and unsupported configurations. Palette handles the pre-flight validation so you don't have to learn the hard way.

Cilium

Cilium is a modern, open-source networking and security solution for containerized environments. It provides high-level visibility and control over network traffic and offers advanced security features, including encryption, network policy enforcement, and more. Cilium uses eBPF to provide high-performance networking and security, and it is designed to work with Kubernetes.



While Cilium is not required for this reference architecture — other CNI options like Calico are available — we recommend the use of Cilium as it provides a robust networking foundation for both containers and virtualization.

Portworx Enterprise

Portworx by Pure Storage is a modern, distributed, cloud-native storage platform that works with orchestrators such as Kubernetes.

Portworx provides persistent storage for stateful applications running on Kubernetes clusters through software-defined storage that can leverage storage resources from locally attached disks to traditional storage arrays and cloud-provider storage.

When combined with a Pure Storage FlashArray, Portworx can provide scalable, resilient and efficient storage for Kubernetes clusters that is suitable for VM workloads.

Pure Storage FlashArray

Pure Storage FlashArray provides an all-flash storage infrastructure containing flash memory drives. All-flash arrays offer speed, performance and agility for business applications. Affordable flash memory and deduplication technology have removed historical bottlenecks and constraints of spinning disks, enabling organizations to deliver apps at new levels of performance and scale.

FlashArray is natively supported as a backend array by Portworx, providing full automation of FlashArray configuration as Kubernetes nodes join and leave the cluster as part of the cluster's lifecycle.

KubeVirt

KubeVirt is an add-on for Kubernetes that enables Virtual Machine workloads on Kubernetes clusters. KubeVirt addresses the needs of organizations that have adopted Kubernetes but also have virtual machine workloads that cannot easily be containerized. Using KubeVirt, both types of workloads can run in the same Kubernetes cluster.

KubeVirt CDI

KubeVirt Containerized Data Importer (CDI) provides data import functionality for KubeVirt. It significantly simplifies the creation of VMs from OS templates, the import of existing data and the conversion of VMs from other platforms like VMware into KubeVirt.

Descheduler

The Descheduler project is an open source project aimed at assisting Kubernetes with automatic balancing of workloads across cluster nodes. Essentially it aims to achieve the same goal as VMware DRS does for virtual machines, but in this case for containers on a Kubernetes cluster.

Descheduler works by evicting pods from nodes when those nodes reach certain usage thresholds and better balance would be achieved if some pods were restarted on other nodes. This naturally assumes that the application uses multiple pods for availability and can handle a pod getting evicted from one node and restarted on another.

KubeVirt works well with Descheduler since it sets the eviction strategy for virtual machines to LiveMigrate, which means that when Descheduler selects a VM for eviction, the VM is not actually killed but instead live migrated to another node. As a result VMware DRS-like functionality is achieved.

Prometheus Grafana stack

Prometheus collects metrics from all components in the Kubernetes cluster, including KubeVirt and VM workloads. Grafana can then be used to graph these metrics and provide real-time monitoring of the environment.

MetalLB

MetalLB provides load-balancing services for bare metal clusters. It can advertise load-balanced IP addresses via ARP directly on cluster node interfaces or it can advertise BGP routes to existing BGP routers in the network. MetalLB is used for external access to Grafana dashboards and the Nginx ingress service, as well as being available for other applications in the cluster to use.

Nginx

Nginx ingress provides Kubernetes ingress services to the cluster. Ingress is used by the KubeVirt export service and the KubeVirt CDI import service, as well as being available for other applications in the cluster to use.

Multus with Dynamic Networks Controller

Multus is a Kubernetes CNI plugin that enables creating one or more pod network interfaces in Kubernetes on different networks. Multus enables three core use cases on Kubernetes:

1. Giving pods more than 1 network interface on the same network to increase throughput
2. Giving pods more than 1 network interface on different networks to provide dual-homing
3. Replacing the pod network interface for a pod with an interface on a different network

For VM workloads, we are mostly interested in the third use case, where we can use Multus to define a VLAN network and give the VM a network interface on that network. This is very similar to a port group for a specific VLAN on a vSwitch in VMware.

The Dynamic Network Controller allows the hot-adding of network interfaces to running virtual machines.

CSI Snapshot Controller

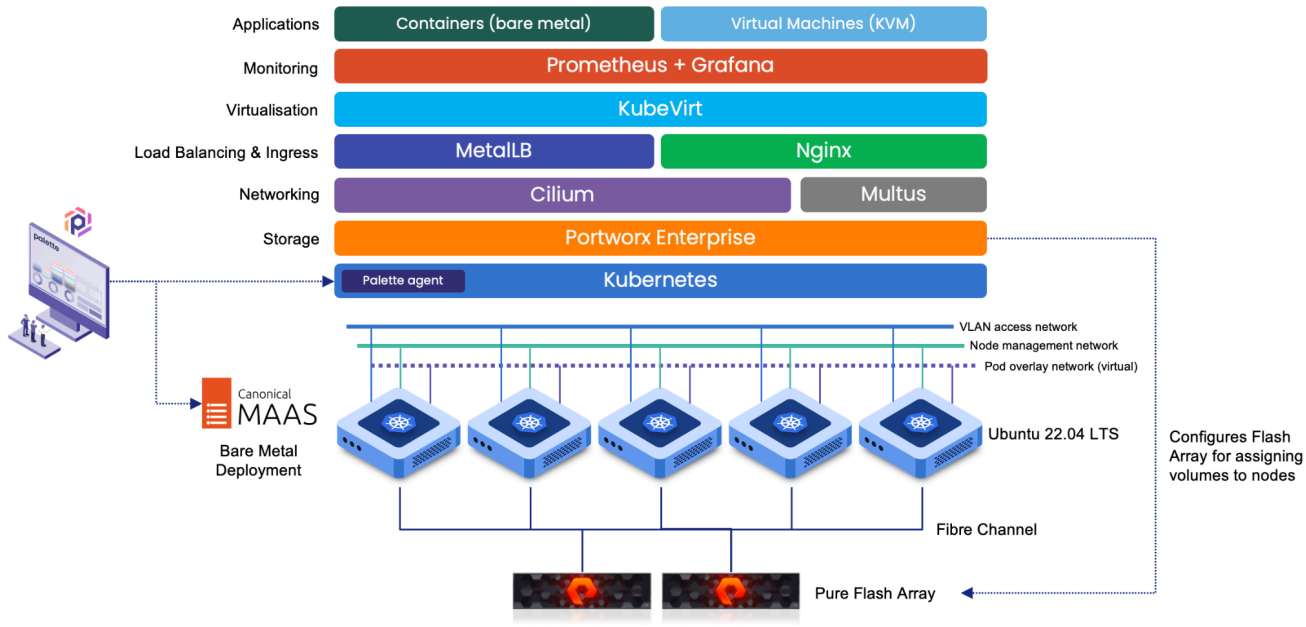
The CSI snapshot controller abstracts the need for storage volume snapshots into a generic concept, hiding the interaction with the actual CSI under the hood.

With the snapshot controller in place, a VolumeSnapshotClass is defined to inform the controller which CSI should be used to perform snapshots. When a snapshot is then triggered from the KubeVirt GUI, a VolumeSnapshot resource is automatically created which the VolumeSnapshotClass picks up to trigger the snapshot of a Persistent Volume.

Solution Configuration

This section describes the overall architecture of the solution, the resources required and the configuration of each component.

Architecture



The diagram shows the reference architecture for VMO with the following core technologies:

- **Canonical MAAS** to automate deployment of OS and Kubernetes on bare metal hardware
- **Pure Storage Flash Array** to provide storage services, orchestrated by **Portworx Enterprise** as the CSI. The bare metal servers do contain internal disks, this is not a hyperconverged configuration.
- **Cilium** to provide network services to containerized workloads. Cilium can also be used for virtual machines that do not need to be exposed on a VLAN and can on the pod overlay network instead (in the case of hybrid workloads).
- **Multus** to provide VLAN network access to virtual machines.
- **MetalLB** to provide IPs for Kubernetes service resources of type LoadBalancer.
- **Nginx** to provide Ingress services to KubeVirt, Prometheus and application workloads.
- **Prometheus** to provide metrics collection and **Grafana** to graph the metrics into monitoring dashboards.

Hardware resources

The hardware guidance detailed here provides minimum and recommended specifications for the components used in this reference architecture. However, this is not the only way to build a VMO cluster. This is just an example of a known-good working solution. The following table describes the hardware guidance for the worker nodes of the cluster:

Component	Minimum spec	Recommended spec	Remarks
Server	2U rackmount chassis	2U rackmount chassis	Needs to fit FC adapters and have sufficient NICs
CPU	Intel/AMD x64 CPU with 8 cores	Intel/AMD x64 CPU with 48+ cores	
RAM	24 GB	256 GB or more	Assume 20 VMs per node, multiple by median RAM per VM.
Network Adapters	2x 10 Gbps (data+mgmt)	2x 10Gbps (data) 2x 10Gbps (mgmt)	Pod overlay runs on mgmt network
Storage Adapters	2x 16 Gbps FC	2x 16 Gbps FC	
Disks	Local disk for OS boot (SAN boot supported)	Local disk for OS boot	Boot from SAN requires special consideration due to the multipath configuration.

The control plane nodes of the cluster typically do not run any of the VMO workloads. As a result, the control plane nodes can be much lighter in hardware specs. A 4-core, 8 GB RAM server is sufficient for a minimum spec control plane node.

Adjust the specs upwards depending on the total number of nodes (control plane + workers) that will be part of the cluster. The following table provides some guidance on sizing control plane nodes:

# of worker nodes	# of namespaces	CPU cores	Memory (GB)
10	100	4	8
25	500	4	16
100	1000	8	32
250	2000	16	64
500	4000	32	128

The table above assumes using at least 3 control plane nodes for a cluster.

A VMO cluster should consist of at least 10 worker nodes to ensure sufficient distribution of resources. This setting, which is necessary for seamless storage failover, requires that VMs cannot run on either of the two nodes that host the persistent storage for it. As a result, the VMO cluster needs to have sufficient nodes so there are always at least two nodes available to place the workload on.

For the Pure Storage FlashArray, the FlashArray//C or FlashArray//X models are compatible with Portworx and suitable for the VMO use case.

Software resources

The software guidance detailed here provides recommended versions for the used components in this reference architecture. However, this is not the only way to build a VMO cluster. This is just an example of a known-good working solution.

The following table describes the software guidance for the nodes of the cluster:

Component	Software product	Recommended version
Bare metal deployment	Canonical MAAS	3.4.1
Operating System	Canonical Ubuntu	22.04 LTS
Multipath software	multipath-tools	0.9.4
Kubernetes	Palette eXtended Kubernetes	1.28.3
Kubernetes networking	Cilium CNI	1.15.3
	Multus	4.0.2 (installed by VMO pack)
Kubernetes storage	Portworx Enterprise CSI	3.1.1
	Pure Storage FlashArray	Purity//FA 6.1.4 or higher
Virtualisation	KubeVirt	1.1.1 (installed by VMO pack)
Image import	KubeVirt CDI	1.58.0 (installed by VMO pack)
Storage Snapshotting	CSI Snapshot Controller	6.3.2 (installed by VMO pack)
Cluster balancing	Descheduler	0.27.1
Monitoring	Prometheus + Grafana	55.8.3
Ingress	Nginx	1.9.5
Load Balancing	MetalLB	0.13.12

Network configuration

Networking for the VMO use case requires a little extra care, compared to a regular Kubernetes cluster. That is because in most cases, some (or all) of the virtual machines will need to be made accessible on existing VLANs. This requires bypassing the typical Kubernetes pod networking stack altogether. This is why Multus is used, as it provides a way to achieve that. It also means there are some requirements to the host network configuration for the Kubernetes worker nodes, in order to have valid network targets to bridge the VMs onto.

Let's first define the networks we need and assign some IP ranges to them for clarity:

Network	VLAN ID	Network CIDR	Gateway
Bare metal deployment	0 (native)	192.168.0.0/22	None
Kubernetes hosts (mgmt)	10	172.16.0.0/22	None
End user access (data)	20	10.20.30.0/16	10.20.30.1
Pod overlay network	N/A (virtual)	100.64.0.0/18	None
Cluster services network	N/A (virtual)	100.64.64.0/18	None
Existing VLANs for VMs	21 - 100		

The end user access network can be used for publishing non-VM apps in two different ways by MetalLB:

- As a network to advertise IP addresses directly onto, as Layer 2 advertisements.
- As a BGP network, where MetalLB can advertise BGP addresses to BGP routers.

Either of these options can be chosen, depending on the network equipment used.

It is recommended to use a dedicated VLAN for end user access, which is not shared with VLANs used by VMs. While it is possible to share the same VLAN for VMs and Kubernetes, special considerations must be taken into account if this VLAN also has the default gateway.

The following network configuration on the host, using a total of 4 NICs in 2 bonds, is suitable for the setup described above. We will use Canonical MAAS to automate that for us during server deployment.

Interface	Type	Consisting of	VLAN	CIDR	Gateway
bond_mgmt	Bond	enp1s0 enp2s0	native	192.168.0.0/22	None
bond_mgmt.10	VLAN	bond_mgmt	10	172.16.0.0/22	None
bond_data	Bond	enp1s1 enp2s1	native		
bond_data.20	VLAN	bond_data	20	10.20.30.0/16	10.20.30.1
br0	Bridge	bond_data	native		

The **br0** bridge interface is used as a master interface for Multus, on top of which Multus can automatically create VLAN interfaces as necessary to place virtual machines onto. The master interface for this scenario must be a bridge interface. It does not work with any other type.

For the setup above, it is assumed that the physical servers (the worker nodes) have 4 physical network interfaces, connected to the switch as follows:

Physical port	Name in OS	Purpose	Switchport config
NIC 1, Port 1	enp1s0	PXE Boot for OS deployment Management network	Trunk (allowing 0, 10)
NIC 1, Port 2	enp2s0	Management network	Trunk (allowing 0, 10)
NIC 2, Port 1	enp1s1	Data network	Trunk (allowing 20-100)
NIC 2, Port 2	enp2s1	Data network	Trunk (allowing 20-100)

The VLAN 0 (untagged/native) network for PXE boot can also be a tagged VLAN network, e.g. VLAN 5. However, to ensure you can successfully PXE boot on that network, it is recommended

to set the native VLAN on the switchport to that VLAN ID (5 in this case), so that the PXE boot can work with untagged traffic.

Alternatively, if the server supports UEFI PXE booting and allows setting the VLAN ID for PXE boot directly, that can also be used. In that case, the configuration for **bond_mgmt** above needs to be adjusted to run the 192.168.0.0/22 IP address on a **bond_mgmt.5** subinterface. PXE booting on a tagged VLAN is difficult to accomplish in practice though, we recommend using a native (untagged) VLAN for PXE.

The **bond_data.20** subinterface provides outbound connectivity as it has the default gateway. This is the primary way to publish services from container workloads to end users. Any specific datacenter networks we want to reach over the **bond_mgmt.10** subinterface instead, can be configured through static routes on the 172.16.0.0/22 subnet in Canonical MAAS. Those routes will be automatically applied upon server installation by MAAS.

For publishing workloads from virtual machines there are two options:

1. Run the virtual machine on the pod network (as if it was a container) and publish individual ports of the VM as Kubernetes services on the **bond_data.20** network, using MetalLB to assign IP addresses.
2. Place the entire VM on a VLAN, using Multus to assign the VM to a VLAN on top of the **br0** interface. It is then the responsibility of the VM itself (for static IPs) or the network (for DHCP) to assign IP addresses.

If your setup is limited to 2 physical network interfaces, some adjustments need to be made. Assuming the same networks and VLANs are used, the following network configuration (configured through Canonical MAAS) is suitable:

Interface	Type	Consisting of	VLAN	CIDR	Gateway
bond0	Bond	enp1s0 enp2s0	native		
bond0.10	VLAN	bond0	10	172.16.0.0/22	None
bond0.20	VLAN	bond0	20	10.20.30.0/16	10.20.30.1
br0	Bridge	bond0	native	192.168.0.0/22	None

For the setup above, it is assumed that the physical servers (the worker nodes) are connected to the switch as follows:

Physical port	Name in OS	Purpose	Switchport config
NIC 1, Port 1	enp1s0	PXE Boot for OS deployment Management network Data network	Trunk (allowing 0, 10, 20-100)
NIC 1, Port 2	enp2s0	Management network Data network	Trunk (allowing 0, 10, 20-100)

Note that in this configuration, VLANs 10 and 20 are NOT available for use by virtual machines on the **br0** interface. This is because VLAN subinterfaces on the master interface of a bridge are mutually exclusive with VLAN subinterfaces on the bridge.

If virtual machines need to be able to run on the same VLAN as either the mgmt (10) or the data (20) VLAN, this can be facilitated by changing the configuration as follows:

Interface	Type	Consisting of	VLAN	CIDR	Gateway
bond0	Bond	enp1s0 enp2s0	native		
bond0.10	VLAN	bond0	10	172.16.0.0/22	None
br0	Bridge	bond0	native	192.168.0.0/22	None
br0.20	VLAN	br0	20	10.20.30.0/16	10.20.30.1

In the example above, we have defined VLAN 20 as a subinterface of **br0** instead of on **bond0**. This configuration allows virtual machines to also run on VLAN 20 without conflict.

In order to allow traffic on **br0.20**, a special feature needs to be activated in the `charts.virtual-machine-orchestrator.vlanFiltering` section of the VMO layer in the Cluster Profile. Specifically:

- The `allowVlansOnSelf` parameter needs to be set to `"true"`
- The `allowedVlansOnSelf` parameter needs to be set to the combined range of VLAN IDs allowed for use by VMs, and the range of VLAN IDs to be used on the host itself.

If for some reason the Kubernetes nodes do not run on a VLAN subinterface, but on the **br0** interface directly, you must enable the “Run Cilium On Bridge (br0)” preset in the Cilium pack.

Storage configuration

Scalable, performant storage is an important component of a Kubernetes cluster, even more so when you want to run virtual machine workloads on it as well. This chapter details the minimum requirement to Kubernetes storage for KubeVirt, as well as the specific implementation based on Portworx and Pure Flash Array that was validated for this reference architecture.

While the Portworx + Flash Array solution is known to work, this does not mean it is the only viable solution. We have seen good results with other storage solutions as well, such as NetApp, DellEMC PowerFlex and Rook-Ceph. Future reference architectures from either Spectro Cloud or our implementation partners are expected to detail these options.

Enabling Live Migration

The core requirement of Kubernetes storage for KubeVirt is support for Persistent Volumes with a ReadWriteMany (RWX) access mode. This is because Live Migration of a VM is only possible with RWX volumes, and Live Migration is a common requirement for running virtual machines in production environments.

Most storage providers support RWX access for Persistent Volumes in one access mode, either Block or FileSystem mode. The following storage providers are known to have the proper support:

Storage Provider	**Version**	**RWX access mode**
Portworx Enterprise	2.x, 3.x	Filesystem
Rook-Ceph	1.11.x	Block
Longhorn	1.5.x	Filesystem
Netapp Trident	23.x	Filesystem
Dell CSM Operator - PowerFlex	2.10.0	Block
Dell CSM Operator - PowerMax	2.10.0	Block
Dell CSM Operator - PowerStore	2.10.0	Block

The preferred RWX access mode is Block when available, as this incurs less overhead than the Filesystem access mode.

When only ReadWriteOnce (RWO) is supported, this does not allow for Live Migration. It is essential that the chosen storage solution supports RWX volumes for either FileSystem or Block mode.

Considering the impact of Kubernetes upgrades

Distributed Kubernetes storage solutions such as Rook-Ceph and Portworx enable hyperconverged infrastructure solutions, where the storage is provided by aggregating local disks in servers (or LUNs from traditional storage arrays) and providing distributed storage logic on top of it (similar to technologies used in VMware vSAN). When using this approach, special consideration is needed with regards to Kubernetes upgrades for clusters managed through Cluster API (which includes Spectro Cloud Palette).

Clusters managed through Cluster API use a repave method to perform OS & Kubernetes version upgrades. Essentially a cluster node gets gracefully removed from the cluster, then replaced by a new node with a fresh installation using the newer OS and/or Kubernetes version. And then the next node, and so on until all nodes in the cluster have been replaced by new nodes running the desired version.

This presents a challenge for hyper-converged clusters, as every node repave causes a significant amount of storage I/O due to rebuilding the data from the lost node. This is of course undesirable for multiple reasons. To ensure bare metal CAPI clusters with hyper-converged storage can successfully perform repaves without storage issues, implement the approach outlined in our article on The New Stack: https://thenewstack.io/reliable-distributed-storage-for-bare-metal-capi-clusters/

The approach outlined in the article sets a fixed pool in MAAS for the worker nodes and prevents data on non-OS disks from being wiped during repaves.

Selected solution

In this reference architecture we selected Portworx as the CSI for the backend Pure Storage FlashArray//C, as this provides some unique benefits with regard to performance, efficiency and ease of management:

- Both products are owned by the same company and thus are made to work well together. Portworx is now the designated CSI for controlling Pure Storage arrays from Kubernetes.
- Portworx provides NFS-based shared volumes (RWX) based on storage pools backed by FlashArray storage.
- As Portworx initializes on each Kubernetes worker node, it automatically requests a LUN from the FlashArray to join the shared storage pool on the cluster. Persistent Volumes are then served from the shared storage pool.
- FlashArray performs deduplication on the backend to increase storage efficiency. Since Portworx will create two copies of the data for resiliency, the deduplication on the backend array wins back some of that capacity.
- Portworx can automatically reassign LUNs from FlashArray to other Kubernetes nodes, allowing cluster repaves to happen without the rebuilding of any data.

To ensure the Portworx configuration is suitable for our needs, we are setting the following specific options:

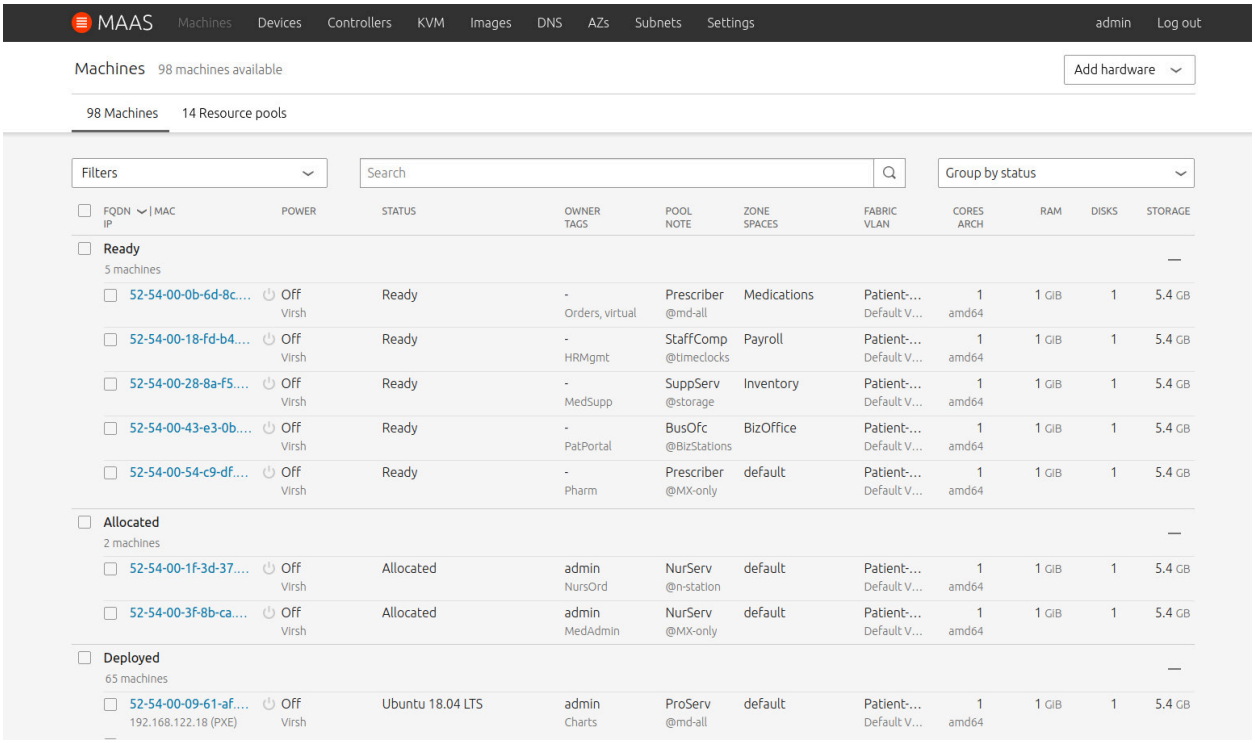
Setting	Value	Remarks
License type	Enterprise	Cannot use Direct Access volumes as those do not support RWX.
Storage cluster image	portworx/oci-monitor:3.1.1	Using latest version.
deleteStrategy	UninstallAndWipe	Ensure LUNs on the FlashArray are cleaned up as part of storagecluster deletion.
maxStorageNodesPerZone	[# of worker nodes] - 1	Ensures quick failover of LUNs during node failure or cluster upgrades.
cloudStorage.journalDeviceSpec	auto	Ensures creation of journal LUNs.
autopilot.providers.params	url = [prometheus url]	Autopilot depends on Prometheus metrics to function, we need to make sure it is pointing to the correct Prometheus instance.
stork.args	webhook-controller = true	Ensure the stork scheduler is involved when scheduling pods to account for storage topology.

Setting	Value	Remarks
env	PURE_FLASHARRAY_SAN_TYPE = FC	Informs Portworx to configure FlashArray for consumption via Fibre Channel..
storageClass.parameters	repl: “2” priority_io: "high" sharedv4: true sharedv4_svc_type: "ClusterIP" sharedv4_mount_options: “vers=3.0,nolock” stork.libopenstorage.org/preferRemoteNodeOnly: "true"	Configure shared v4 service volumes that can seamlessly failover. This is very important to ensure VMs stay up during node failures.

Canonical MAAS configuration

Spectro Cloud Palette natively supports Canonical MAAS for bare metal cluster deployment. The flexibility of MAAS to configure networking and storage for bare metal servers is particularly helpful for the VMO use case.

We will use MAAS to ensure consistent configuration across the nodes, allowing enough flexibility to fit any environment.



We will use the following features of MAAS to aid in the deployment of our VMO clusters:

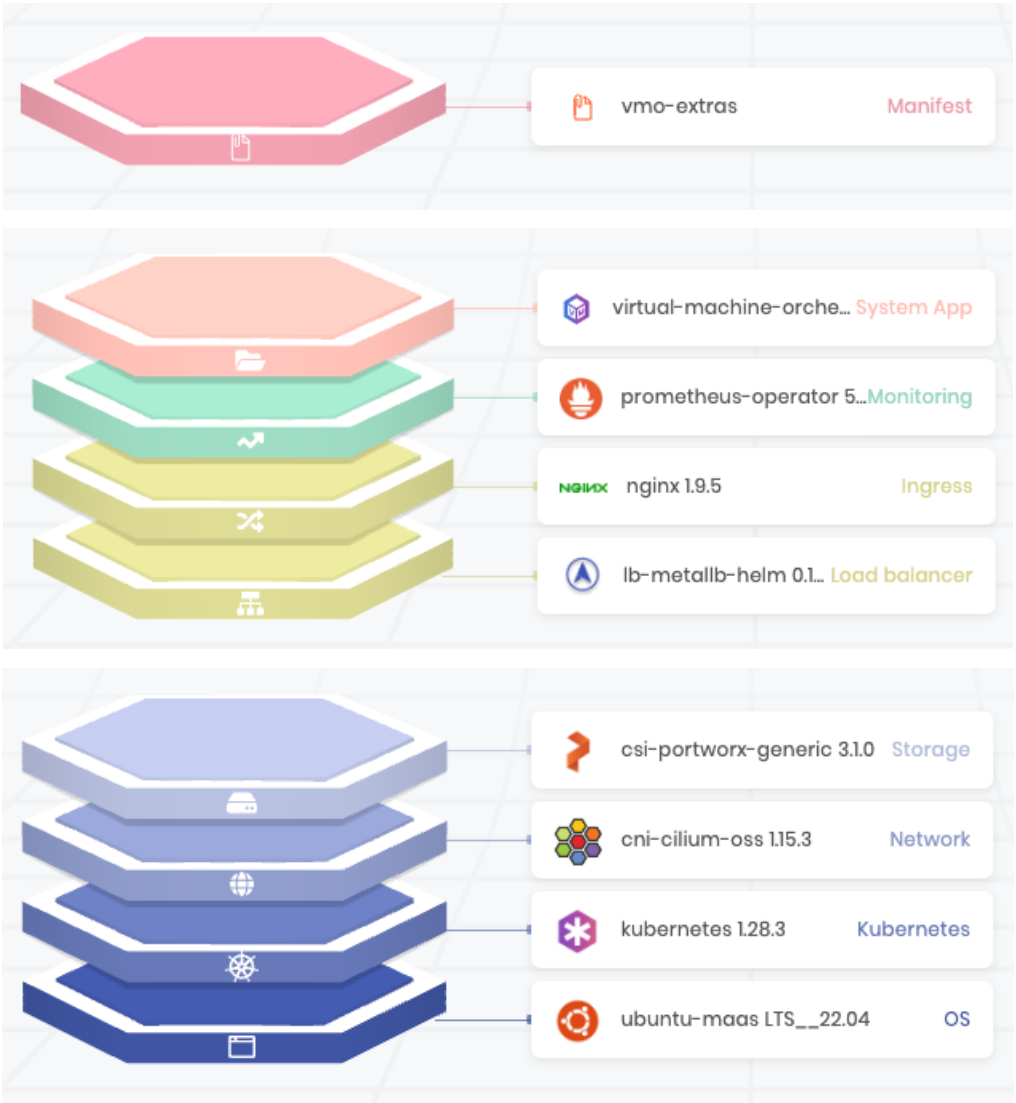
Feature	Configuration	Remarks
Resource Pools	`control-plane` and `worker-pool` resource pools. Add more as necessary.	Separate pools so hardware specs can be optimized for each function.
AZs	Align Availability Zones to datacenter layout	If your hardware is located in separate racks/aisles/rooms in the datacenter, you can create an AZ in MAAS for each. The cluster will be spread across the selected AZs.

Feature	Configuration	Remarks
Subnets	192.168.0.0/22 (Auto assign) 172.16.0.0/22 (Auto assign) 10.20.30.0/16 (Auto assign[1])	Align subnets to network layout. The subnets for PXE deployment, mgmt and data should be configured in MAAS.
DNS	metal.[company DNS zone]	Recommended approach for DNS is to create a delegated DNS subzone in the corporate DNS infrastructure, delegating the subzone to MAAS.
Settings → Storage	"Erase nodes' disks prior to releasing" = disabled	This setting **must** be disabled so that LUNs on the FlashArray do not get erased when nodes are repaved.
Machines → Machine network configuration	Bond and Bridge interfaces configured for worker nodes	In accordance with the tables in the Network Configuration section.
Machines → Machine storage configuration	Only configure OS disk. Ensure all other disks are removed from the storage layout.	Recommend to use LVM or Flat layout.

[1] While giving every node an IP address in the 10.20.30.0/16 range isn't strictly necessary for MetalLB to function, assigning a node IP and a default gateway simplifies network routing as the node already knows how to route traffic back to clients via the default gateway. This prevents us from having to put this routing information in place on each worker node separately via other means.

Spectro Cloud Palette configuration

This section shows the detailed configuration of each layer in the Cluster Profile for this reference architecture.



This reference architecture can be imported by retrieving the exported cluster profiles from this Github Gist and saving the JSON files locally. They can then be imported one by one into Palette by selecting **Import Cluster Profile**.

Ubuntu layer (VMO-RA-Infra-PureFA profile)

Pack Type	OS
Registry	Public Repo
Pack Name	Ubuntu
Pack Version	LTS__22.04
Values	Modified from default

By default this layer does not contain any files or commands, so all of the configuration in this layer is custom for this reference architecture.

```
kubeadmconfig:
  preKubeadmCommands:
    - 'echo "====> Applying pre Kubeadm commands"'
    # Force specific IP address as the Node InternalIP for kubelet
    - |
      apt update && apt install -y grepcidr
      NETWORKS="172.16.0.0/22"
      IPS=$(hostname -I)
      for IP in $IPS
      do
        echo "$IP" | grepcidr "$NETWORKS" >/dev/null && echo " --node-ip=$IP" >> /etc/default/kubelet
        if [ $? == 0 ]; then break; fi
      done
    # Update CA certs
    - update-ca-certificates
    # Build 0.9.4 of multipath-tools as the 0.8.x versions that come with Ubuntu have issues
    - |
      apt install -y multipath-tools multipath-tools-boot
      apt install -y make pkg-config gcc libmount-dev libdevmapper-dev libaio-dev libudev-dev libjson-c-dev liburcu-dev
libsystemd-dev
      mkdir /tmp/mptools
      cd /tmp/mptools
      wget "https://github.com/opensvc/multipath-tools/archive/0.9.4.tar.gz" -O multipath-tools-0.9.4.tar.gz
      tar xvf multipath-tools-0.9.4.tar.gz
      cd multipath-tools-0.9.4
      make plugindir="/usr/lib/multipath"
      make plugindir="/usr/lib/multipath" install
      echo "/usr/lib64" > /etc/ld.so.conf.d/multipath.conf
      ldconfig
      systemctl daemon-reload
      systemctl restart multipathd
      sleep 15
      /etc/kubernetes/check_and_fix_mpath_alias.sh
      systemctl daemon-reload
      systemctl restart multipathd
      sleep 5
      update-initramfs -u -k all
    # Start containerd with new configuration
    - systemctl daemon-reload
    - systemctl restart containerd
  postKubeadmCommands:
    - 'echo "====> Applying post Kubeadm commands"'
    # Reload udev rules, this could crash the node so do it after K8S has been configured
    - udevadm control --reload-rules
    - udevadm trigger
  files:
    - targetPath: /etc/containerd/config.toml
      targetOwner: "root:root"
      targetPermissions: "0644"
      content: |
        ## template: jinja

        # Use config version 2 to enable new configuration fields.
        # Config file is parsed as version 1 by default.
        version = 2
```

```
      imports = ["/etc/containerd/conf.d/*.toml"]

      [plugins]
        [plugins."io.containerd.grpc.v1.cri"]
          sandbox_image = "registry.k8s.io/pause:3.9"
          device_ownership_from_security_context = true
        [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
          runtime_type = "io.containerd.runc.v2"
        [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
          SystemdCgroup = true

  - targetPath: /usr/local/share/ca-certificates/org_ca.crt
    targetOwner: "root:root"
    targetPermissions: "0644"
    content: |
      -----BEGIN CERTIFICATE-----
      MIIDHzCCAgegAwIBAgIUC2VKm0kzJFb8sDxcYIAHpRsrkZwwDQYJKoZIhvcNAQEL
      BQAwEjEQMA4GA1UEAwwHcm9vdF9jYTAeFw0yMzA1MTUwMjI3MDdaFw0yOTAxMjYx
      MDI3MzdaMA0xCzAJBgNVBAMTAmNhMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
      CgKCAQEA1xaF23C53r7hOjAY6lTeNX5aBCaF9scdl/Dt4G16eKLRbGhOxdglk6N1
      A6nrErJMxk8QIFY33ZBKCDGIvefsDEoHX7J9MTjlSpvXJgyjLsmRUhWvTOnNf+oe
      xdIC2dRWuU+8y/6JbnaWfx4hYyLbYEguJyrFrFUjwQ1bdJIChnvzRFUV43uKLENC
      vR/CiWy/v9m7itCl4WmVRw6+4GLv/IYzSc+EWpC4Rr/z8beQT0l+fmAgdakyfuKP
      Icuo6QhKSszZxGUnVnh5LVKkHGdk9TgGBEEp1Ja6YI9k2ODiugnNMgJbIdf8TmXd
      jXWCo1Db+6IsrWPqjwCPrrDQolDD6QIDAQABo3IwcDAOBgNVHQ8BAf8EBAMCAQYw
      DwYDVR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQUKq7+i3gHH5s3WZGkL5C09SG7fN4w
      HwYDVR0jBBgwFoAUsuhA2cGXGoYZZzAeJrIVHkkkwFgwDQYDVR0RBAYwBIICY2Ew
      DQYJKoZIhvcNAQELBQADggEBABsD4LDofyRC2cVas0QlECYfSrJhtIVxD5CsGurO
      pWpYIvTnC5AMSX5PpWlc3nAnv5OMGs+G+jD4paEQRPofZvjA+fUUMLVNtVOv0p9E
      Dm5FpgZBlwRlcrieejAXDl8wOlO54BBCK/3vkIgtcaUJ3WrOCoI8QU0NEvYhBZW3
      +vt2oini0M5J5keB3kVKrfuQHQs41yMmWLWpuz5GlX5354IpDoDyeEzx6/Q0Uqhg
      espKDCkDqjNYIhrhHnhXOW8fGOspOISfDKj5H6bgfKAF3TFfqd5P+AdzZNand8cC
      euKlWnTpf9FTumw6znzybwsmfUUopmY17mN65V8X91ks15Y=
      -----END CERTIFICATE-----
  - targetPath: /etc/multipath.conf
    targetOwner: "root:root"
    targetPermissions: "0644"
    content: |
      defaults {
        user_friendly_names no
        find_multipaths     no
        polling_interval    10
      }
      blacklist {
        devnode "^pxd[0-9]*"
        devnode "^pxd*"
        device {
          vendor "VMware"
          product "Virtual disk"
        }
      }
      devices {
        device {
          vendor                      "NVME"
          product                     "Pure Storage FlashArray"
          path_selector               "queue-length 0"
          path_grouping_policy        group_by_prio
          prio                        ana
          failback                    immediate
          fast_io_fail_tmo            10
          user_friendly_names         no
          no_path_retry               0
          features                    0
          dev_loss_tmo                60
        }
        device {
          vendor                    "PURE"
          product                   "FlashArray"
          path_selector             "service-time 0"
          hardware_handler          "1 alua"
          path_grouping_policy      group_by_prio
          prio                      alua
          failback                  immediate
          path_checker              tur
          fast_io_fail_tmo          10
          user_friendly_names       no
          no_path_retry             0
          features                  0
          dev_loss_tmo              600
        }
      }
  - targetPath: /etc/udev/rules.d/99-pure-flasharray.rules
    targetOwner: "root:root"
    targetPermissions: "0644"
    content: |
      # Recommended settings for Pure Storage FlashArray.
      # Use none scheduler for high-performance solid-state storage for SCSI devices
      ACTION=="add|change", KERNEL=="sd*[!0-9]", SUBSYSTEM=="block", ENV{ID_VENDOR}=="PURE", ATTR{queue/scheduler}="none"
      ACTION=="add|change", KERNEL=="dm-[0-9]*", SUBSYSTEM=="block", ENV{DM_NAME}=="3624a937*", ATTR{queue/scheduler}="none"

      # Reduce CPU overhead due to entropy collection
```

```
        ACTION=="add|change", KERNEL=="sd*[!0-9]", SUBSYSTEM=="block", ENV{ID_VENDOR}=="PURE", ATTR{queue/add_random}="0"
        ACTION=="add|change", KERNEL=="dm-[0-9]*", SUBSYSTEM=="block", ENV{DM_NAME}=="3624a937*", ATTR{queue/add_random}="0"

        # Spread CPU load by redirecting completions to originating CPU
        ACTION=="add|change", KERNEL=="sd*[!0-9]", SUBSYSTEM=="block", ENV{ID_VENDOR}=="PURE", ATTR{queue/rq_affinity}="2"
        ACTION=="add|change", KERNEL=="dm-[0-9]*", SUBSYSTEM=="block", ENV{DM_NAME}=="3624a937*", ATTR{queue/rq_affinity}="2"

        # Set the HBA timeout to 60 seconds
        ACTION=="add|change", KERNEL=="sd*[!0-9]", SUBSYSTEM=="block", ENV{ID_VENDOR}=="PURE", ATTR{device/timeout}="60"

        # Set maximum IO size to 4MB
        ACTION=="add|change", KERNEL=="sd*[!0-9]", SUBSYSTEM=="block", ENV{ID_VENDOR}=="PURE",
ATTR{queue/max_sectors_kb}="4096"
    - targetPath: /etc/kubernetes/check_and_fix_mpath_alias.sh
      targetOwner: "root:root"
      targetPermissions: "0744"
      content: |
        #!/bin/bash
        if cat /boot/grub/grub.cfg | grep "/dev/mapper/mpath" > /dev/null
        then
          echo "Multipath is using friendly names, we need to set an alias to ensure we can reboot after disabling friendly
names..."
          ALIAS=$(mount |grep -E 'mapper.*on / type'|awk -F' ' '{print $1}'|awk -F/ '{print $4}'|awk -F- '{print $1}')
          echo "Multipath friendly name: $ALIAS"
          WWID=$(multipath -ll|grep -B1 '128G' | grep FlashArray | awk '{print $1}')
          echo "Multipath WWID: $WWID"
          echo "Adding the following alias to /etc/multipath.conf"
          printf '\nmultipaths {\n    multipath {\n        wwid "%s"\n        alias "%s"\n    }\n}\n' $WWID $ALIAS
          printf '\nmultipaths {\n    multipath {\n        wwid "%s"\n        alias "%s"\n    }\n}\n' $WWID $ALIAS >>
/etc/multipath.conf
        fi
    - targetPath: /etc/nfs.conf
      targetOwner: "root:root"
      targetPermissions: "0644"
      content: |
        [general]
        pipefs-directory=/run/rpc_pipefs
        #
        [exports]
        #
        [exportfs]
        #
        [gssd]
        #
        [lockd]
        #
        [mountd]
        manage-gids=y
        port=9024
        #
        [nfsdcld]
        #
        [nfsdcltrack]
        #
        [nfsd]
        #
        [statd]
        #
        [sm-notify]
        #
        [svcgssd]
        #
```

The `kubeadmconfig.preKubeadmCommands` section is used to run a number of commands to setup our nodes for virtualization:

- Add the node's IP address on the **bond_mgmt.10** network to /etc/default/kubelet, to ensure this IP becomes the InternalIP for this cluster node
- Build and install version 0.9.4 of the multipath-tools from source. Ubuntu 22.04 LTS ships with v0.8.8 version, which has a high cpu usage issue under some circumstances. If the system is SAN-booted, a script will adjust the Grub boot configuration to account for the change in multipath.conf with regards to user_friendly_names needing to be disabled for Portworx.

- Reconfigure containerd to include the `device_ownership_from_security_context = true` setting, which allows us to export VMs from the cluster when needed.
- Fix the TCP port used by mountd in order to be compatible with NFSv3.
- Best practices
    - Run update-ca-certificates to make the system trust any custom certificates you may have (`/usr/local/share/ca-certificates/org_ca.crt` is an example certificate)
    - Setting Linux best practices for retrieving data from flash/NVMe based storage with the highest performance (by setting udev rules and reloading them).

We use the `kubeadmconfig.files` section to enforce certain file content on the system, such as the containerd and multipathd configuration, as well as any certificate authorities that the system needs to trust. If Palette is used as the OIDC Identity Provider in the Kubernetes layer, the issuing CA for the certificate that Palette is using, needs to be installed on all nodes. If you are using your own OIDC Identity Provider and use private certificates, you need to install the issuing CA's certificate on all nodes.

If you require the use of a proxy server to access the internet, the `wget` command for retrieving the v0.9.4 multipath-tools tarball might not work. In that case, either host the file somewhere internally so that it can be retrieved, or precede the `wget` command with `https_proxy=<proxy-server-ip>:<port>` to inform it of the proxy config.

The files from the `kubeadmconfig.files` section are put in place before the `preKubeadmCommands` section is executed, so we can always assume the files are already there.

Kubernetes layer (VMO-RA-Infra-PureFA profile)

Pack Type	Kubernetes
Registry	Public Repo
Pack Name	Palette eXtended Kubernetes
Pack Version	1.28.3
Values	Modified from default

The modified aspects of the configuration for the Kubernetes layer are as follows:

```
pack:
  podCIDR: "100.64.0.0/18"
  serviceClusterIpRange: "100.64.64.0/18"
  palette:
    config:
      dashboard:
        identityProvider: palette
```

Most of this layer is in its default configuration, the changes made are:

- Line 5: setting the Pod CIDR in accordance to the network design `podCIDR: "100.64.0.0/18"`
- Line 8: setting the Service CIDR in accordance to the network design `serviceClusterIpRange: "100.64.64.0/18"`
- Line 9-12: setting the OIDC Identity Provider option (in the GUI) as desired. It is recommended to either use “Palette”, or set your own OIDC provider for single sign on.

Cilium layer (VMO-RA-Infra-PureFA profile)

Pack Type	Network
Registry	Public Repo
Pack Name	Cilium
Pack Version	1.15.3
Values	Set using Pack Presets

The following pack presets were selected:

IPAM mode	Kubernetes
Cilium Operator	For Multi-Node Cluster
Kube-proxy replacement	Replace Kube-Proxy with EBPF
VMO Compatibility	Enable
VMO - Bridge Interface	Autodetect Cilium Interface
Loadbalancer mode	Best Effort XDP LB Acceleration

These presets result in the following adjustments to the pack configuration:

```
charts:
  cilium:
    k8sServiceHost: auto
    k8sServicePort: ""
    kubeProxyReplacement: "true"
    kubeProxyReplacementHealthzBindAddr: 0.0.0.0:10256
    autoDirectNodeRoutes: true
    ipv4NativeRoutingCIDR: 100.64.0.0/17
    routingMode: native
    bpf:
      masquerade: true
    cni:
      exclusive: false
    socketLB:
      hostNamespaceOnly: true
```

Portworx layer (VMO-RA-Infra-PureFA profile)

Pack Type	Storage
Registry	Public Repo
Pack Name	Portworx /w Operator
Pack Version	3.1.0
Values	Modified from default

The following pack presets were selected:

License Models	PX Enterprise
Storagecluster Specs	Pure Storage Flash Array
Kvdb and Etcd	Use Internal Kvdb
HTTP(S) Proxy	*<none selected>*

After enabling these presets, further changes are made as follows:

```
charts:
  portworx-generic:
    license:
      enterprise:
        activateLicense: true
        activationId: your_license_key
      type: enterprise

    storageCluster:
      autoGenerateName: false
      name: "px-{{.spectro.system.cluster.name}}"
      spec:
        image: portworx/oci-monitor:3.1.1
        imagePullPolicy: Always
        deleteStrategy:
          type: UninstallAndWipe
        kvdb:
          internal: true
        cloudStorage:
          maxStorageNodesPerZone: 24
          deviceSpecs:
            - size=1024
          kvdbDeviceSpec: size=32
          journalDeviceSpec: auto
        network:
          dataInterface: bond_mgmt.10
          mgmtInterface: bond_mgmt.10
        secretsProvider: k8s
        stork:
          enabled: true
          args:
            webhook-controller: "true"
        autopilot:
          enabled: true
          providers:
          - name: default
            params:
```

```
            url: http://prometheus-operator-prometheus.monitoring.svc.cluster.local:9090
          type: prometheus
      runtimeOptions:
        default-io-profile: "6"
      csi:
        enabled: true
      monitoring:
        telemetry:
          enabled: true
        prometheus:
          enabled: false
          exportMetrics: true
      env:
        - name: PURE_FLASHARRAY_SAN_TYPE
          value: "FC"

  storageClass:
    name: spectro-storage-class
    isDefaultStorageClass: true
    annotations: {}
    #   If you need additional annotations, specify them here
    allowVolumeExpansion: true
    reclaimPolicy: Delete # Delete or Retain
    volumeBindingMode: WaitForFirstConsumer # WaitForFirstConsumer or Immediate
    parameters:
      repl: "2"
      priority_io: "high"
      sharedv4: true
      sharedv4_svc_type: "ClusterIP"
      sharedv4_mount_options: vers=3.0,nolock
      stork.libopenstorage.org/preferRemoteNodeOnly: "false"
```

The changes made are:

- Line 53: a valid license key for Portworx Enterprise needs to be set.
- Line 69: the image version for Portworx was increased from 3.1.0 to 3.1.1
- Line 79-83: adjusting the `cloudStorage` spec to our specific needs:
    - Setting `maxStorageNodesPerZone` to 1 less than the number of worker nodes in the cluster. This ensures that clouddisks are always reattached to other nodes, preventing unnecessary replication during cluster repaves.
    - Increasing the `deviceSpecs` from the minimum 150 GB to 1024 GB. This will make each worker node contribute 1TB of storage capacity to the shared pool.
    - Adding `journalDeviceSpec: auto` to ensure a journal device automatically gets created.
- Line 84-86: adding a `network` section to instruct Portworx to use specific interfaces for management and data traffic. By default, Portworx will select the first routable interface, which would be **bond_data.20** in our design. We want to ensure this interface only handles application traffic, so we force the use of the **bond_mgmt.10** interface instead. If the physical nodes had 6 NICs, we could also create a 3rd network bond for storage and set the `dataInterface` property to the storage bond instead, giving it dedicated bandwidth. However since we are using Fibre Channel for the majority of storage traffic, this is not strictly necessary and replication can happen over the management network.
- Line 111: changing the FlashArray SAN type from `iSCSI` to `FC`. The default for the Pure Storage Flash Array preset is `iSCSI`, so we need to change that for our FC-based design.

- Line 124-138: adjusting the configuration of the `storageClass` resource:
  - Adding `priority_io: "high"` to specify the highest performance level
  - Enable Shared v4 Service Volumes for higher fault tolerance and seamless live migration by setting:
    `sharedv4: "true"`
    `sharedv4_svc_type: "ClusterIP"`
    `sharedv4_mount_options: vers=3.0,nolock`
  - Adding `stork.libopenstorage.org/preferRemoteNodeOnly: "true"` to force the cluster to schedule VM workloads on nodes that are not also serving the PersistentVolume(s) for that VM. This enables seamless failover in the event of a node failure or cluster repave. Note that this setting requires enough nodes in the cluster to facilitate this anti-affinity rule. Do not use this setting in clusters with 6 or fewer nodes in the worker pool.

Finally, we also add an extra manifest to the layer, named `px-pure-secret` and containing:

```
apiVersion: v1
data:
  pure.json: <base64-encoded content of pure.json file>
kind: Secret
metadata:
  name: px-pure-secret
  namespace: kube-system
type: Opaque
```

The content of `pure.json` looks like this and should be adjusted for the IP address of the FlashArray management endpoint, as well as the API token that provides administrative access:

```
{
  "FlashArrays": [
    {
      "MgmtEndPoint": "172.16.200.50",
      "APIToken": "347eba54-4ab6-96cd-e123-f694ab394a3b"
    }
  ]
}
```

The base64-encoded content can then be created by running:
`cat pure.json | base64`

Prometheus layer (VMO-RA-Core profile)

Pack Type	Monitoring
Registry	Public Repo
Pack Name	Prometheus - Grafana
Pack Version	55.8.3
Install order	10
Values	Modified from default

The modified aspects of the configuration for the Monitoring layer are as follows:

```
charts:
  kube-prometheus-stack:
    grafana:
      adminPassword: "welcome"
```

This sets a default password of “welcome” for the Grafana portal.

MetalLB layer (VMO-RA-Core profile)

Pack Type	Load balancer
Registry	Public Repo
Pack Name	MetalLB (Helm)
Pack Version	0.13.12
Install order	10
Values	Modified from default

The modified aspects of the configuration for the Descheduler layer are as follows:

```
charts:
  metallb-full:
    configuration:
      ipaddresspools:
        first-pool:
          spec:
            addresses:
              - 10.20.30.100-10.20.30.200
            avoidBuggyIPs: true
            autoAssign: true
      l2advertisements:
        default:
          spec:
            ipAddressPools:
              - first-pool
            interfaces:
              - bond_data.20
```

Most of this layer is in its default configuration, the changes made are:

- Line 25: setting a block of addresses in the **bond_data.20** network that can be used for services of type LoadBalancer.
- Line 34-35: adding an interfaces section to instruct MetalLB to only use the **bond_data.20** interface for advertising the IP addresses (by default, MetalLB advertises on all interfaces).

If the target infrastructure has BGP routing capabilities, MetalLB could alternatively be configured to advertise IP addresses to the BGP routers instead.

Nginx layer (VMO-RA-Core profile)

Pack Type	Ingress
Registry	Public Repo
Pack Name	Nginx
Pack Version	1.9.5
Install order	10
Values	All left at default

The configuration for the Ingress layer is all in its default configuration, no changes were made.

VMO layer (VMO-RA-Core profile)

Pack Type	System App
Registry	Public Repo
Pack Name	Virtual Machine Orchestrator
Pack Version	4.3.3
Install order	20
Access	Proxied
Values	Modified from default

This layer has 2 access options: Proxied and Direct.

- The default option is **Proxied**, which is the recommended option for Palette SaaS tenants. It will leverage the Spectro Proxy to ensure access to the Virtual Machines tab, as well as VM remote consoles, from anywhere. The Service resource for the VM orchestration GUI will be configured as ClusterIP and only be accessible through the proxy.
- The **Direct** option is only intended for self-hosted Palette installations, where a proxy is typically not implemented or needed .The Service resource for the VM orchestration GUI will be configured as LoadBalancer and gets directly accessed by the end user. This requires that the user is on a network that can reach the IP address given (by MetalLB) to the LoadBalancer service.

The modified aspects of the configuration for the VM Orchestrator layer are as follows:

```
charts:
  virtual-machine-orchestrator:
    sampleTemplates:
      fedora37: false
      ubuntu2204: false
      ubuntu2204WithVol: false
      ubuntu2204staticIP: false
      fedora37staticIP: false

    vmEnabledNamespaces:
      - "default"
      - "virtual-machines"
    vlanFiltering:
      enabled: true
      namespace: kube-system
      env:
        # Which bridge interface to control
        bridgeIF: "br0"
        # Beginning of VLAN range to enable (comma seperated, use "-" for ranges)
        allowedVlans: "21-100"
```

```
      # Set to "true" to enable VLANs on the br0 interface for the host to use itself
      allowVlansOnSelf: "true"
      # Beginning of VLAN range to enable for use by the node itself (if enabled, must include VLAN range for VMs)
      allowedVlansOnSelf: "1,10,20,21-100"

  snapshot-controller:
    volumeSnapshotClass:
      create: true
      name: portworx-snapshot-class
      driver: pxd.portworx.com
      deletionPolicy: Delete
      params: {}
      extraLabels:
        velero.io/csi-volumesnapshot-class: "true"
```

Most of this layer is in its default configuration, the changes made are:

- Line 31-35: disabling all built in templates.
- Line 50: enabling the vlanFiltering function, which is a daemonset that configures the **br0** interface on each worker node to become VLAN-aware, making it possible to place VMs directly on a VLAN.
    - Set `allowedVlans: "21-100"` to allow VMs to be placed on VLANs within this range. Adjust as necessary.
    - Set `allowedVlansOnSelf: "10,20,21-100"` to allow the worker node itself to use subinterfaces of the **br0** interface on VLANs in the defined range. This feature is enabled by setting `allowVlansOnSelf: "true"` and is only needed if the worker node has subinterfaces on **br0** that it depends on. If the `allowVlansOnSelf` feature is enabled, the `allowedVlansOnSelf` range must always include the `allowedVlans` range, otherwise those VLANs will not be accessible.
- Line 171-180: configuring the `volumeSnapshotClass` resource:
    - Set `create: true` to enable the creation of the volumeSnapshotClass
    - Set the name of the class to `portworx-snapshot-class`
    - Set the driver to `pxd.portworx.com` to reference the Portworx driver
    - Set `deletionPolicy: Delete` to ensure cleanup of snapshot content for which the associated snapshot resource has been deleted
    - Set an extra label to inform Velero that this snapshot class should be used for backups. This is not required if Velero is not used for backup.

Note that in order for the descheduler to be able to live migrate VMs, the following annotation must be set on each VM resource:

```
spec:
  template:
    metadata:
      annotations:
        descheduler.alpha.kubernetes.io/evict: "true"
```

It is therefore recommended to include this annotation on all VM Templates.

VMO extras layer (VMO-RA-AddOn profile)

Pack Type	Manifest
Manifest Name	vmo-extras
Install order	30

This layer contains 9 additional manifests to round out certain aspects of the VMO solution. Some of these manifests are expected to migrate into the VM Orchestrator pack in a future release to ease the implementation process.

The following manifests are implemented in this layer:

Name(s)	Type	Purpose
portworx-autopilot-rules	AutopilotRule	Portworx AutoPilot rules for automatically increasing PVC sizes and balancing storage usage across the cluster
portworx-cluster-dashboard portworx-etcd-dashboard portworx-node-dashboard portworx-perf-dashboard portworx-volume-dashboard	ConfigMap	Grafana dashboards for Portworx
storageprofile-cdi-pwx	storageProfile	A storage profile for CDI that informs it to use RWX volumes for uploaded images and use fast CSI cloning for cloning volumes.
nad-vlan-30	NetworkAttachmentDefinition	A sample network attachment configuration for VLAN access
vmtemplate-ubuntu-2204	VmTemplate	A sample VmTemplate for a persistent VM using a DataVolume
datavolume-templates	DataVolume	Importing external template disks for cloning VMs from.

Some of these layers require slight configuration for the environment in which they are deployed:

- **nad-vlan-30**:
    - Adjust line 4, 10 and 14 to reflect the VLAN ID you want to make available.
    - Adjust line 5 to define in which namespace the VLAN will be available.
    - To make the VLAN available in multiple namespaces, duplicate the resource (separated by a divider line) and set the namespace for each copy.
    - Create another nad-vlan-xxx manifest for every other VLAN that should be available this way.