Spectro Cloud

Palette Virtual Machine Orchestrator (VMO)

Reference architecture for bare metal environments

Rev. 1.7.0 | July 2025

About this reference architecture

This document represents the reference architecture for Palette Virtual Machine Orchestrator (VMO) for bare metal environments. 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 bare metal 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
1.3.0	August 2024	Upgrade to VMO 4.4 Adjust Portworx configuration
1.3.5	October 2024	Performance tuning Implement VM Instance Preferences
1.4.0	February 2025	Upgrade to VMO 4.6
1.4.1	March 2025	Improve config for Pure FlashArray
1.6.0	June 2025	Portworx 3.2.3 and VMO 4.6.4 Implement cluster profile variables Add solution sizing chapter Tuning for dedicated resources
1.7.0	July 2025	Portworx 3.3.1 with Block storage VMO 4.7.0 with Kubevirt 1.5.0

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.

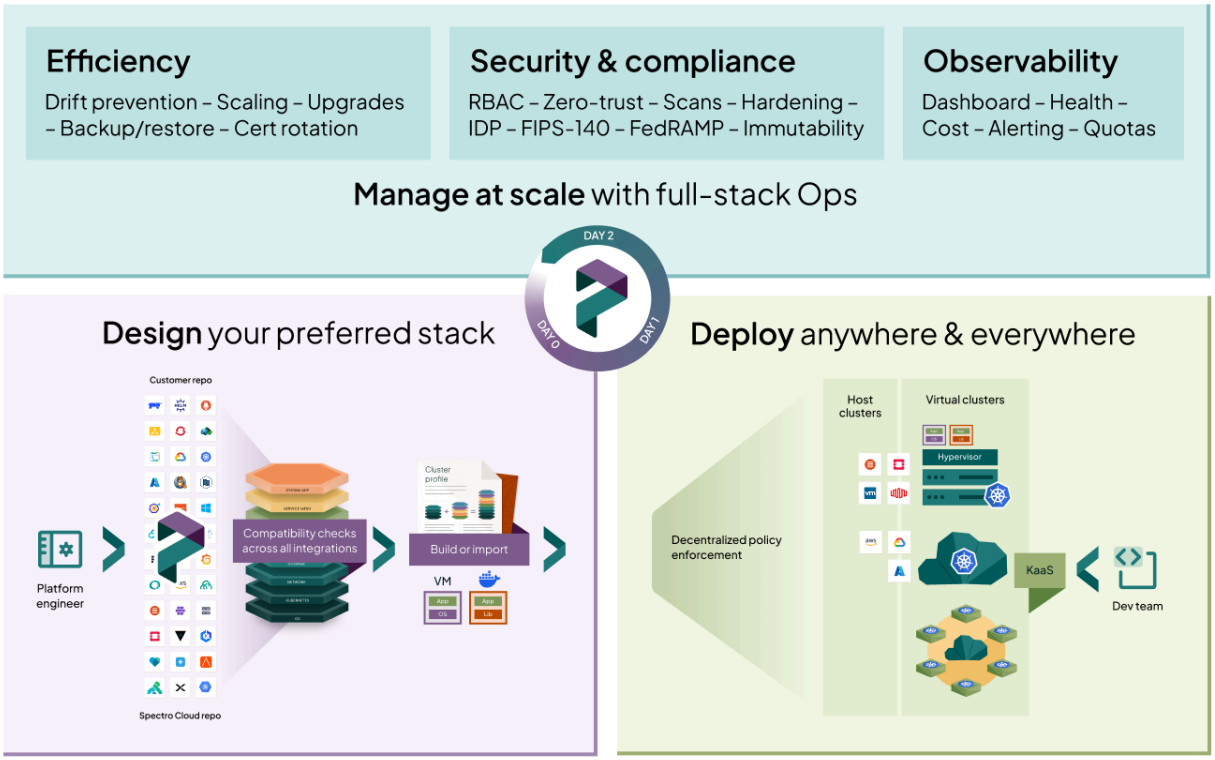
Technology Overview

Overview

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

- Spectro Cloud Palette 4.7.3 with the 4.7.0 VMO pack
- Canonical MAAS 3.6.0
- Ubuntu 22.04 LTS
- Kubernetes 1.32.4
- Cilium 1.17.4
- Portworx Enterprise 3.3.1
- Pure Storage FlashArray
- Descheduler 0.33.0
- Prometheus Grafana stack 70.2.1
- MetalLB 0.15.2
- Nginx 1.12.2
- Multus 4.1.4 with Dynamic Networks Controller
- KubeVirt 1.5.0
- KubeVirt CDI 1.62.0
- CSI Snapshot Controller 8.1.0

Spectro Cloud Palette



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

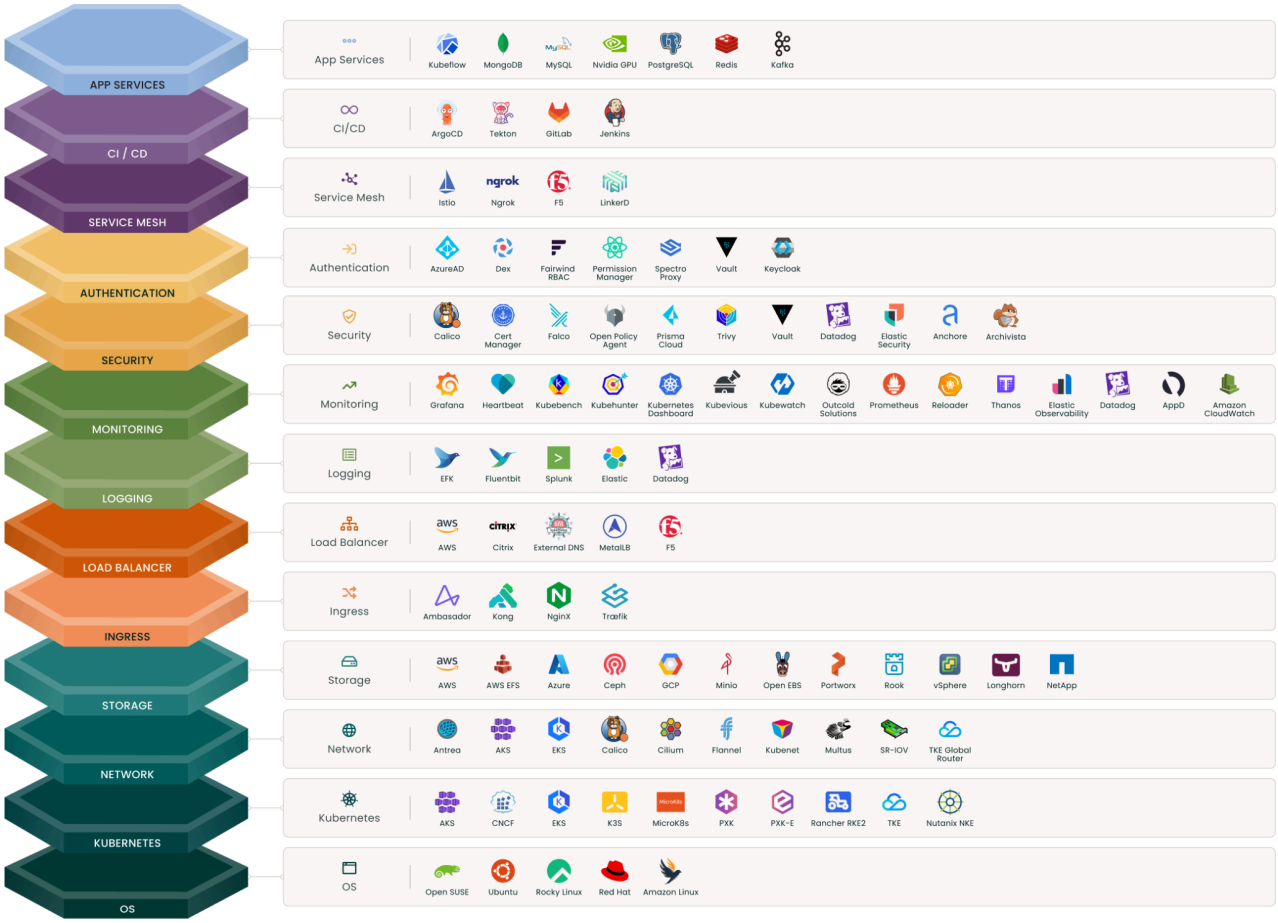
Palette uses Cluster API to automate the deployment of Kubernetes clusters based on a desired state model.

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.

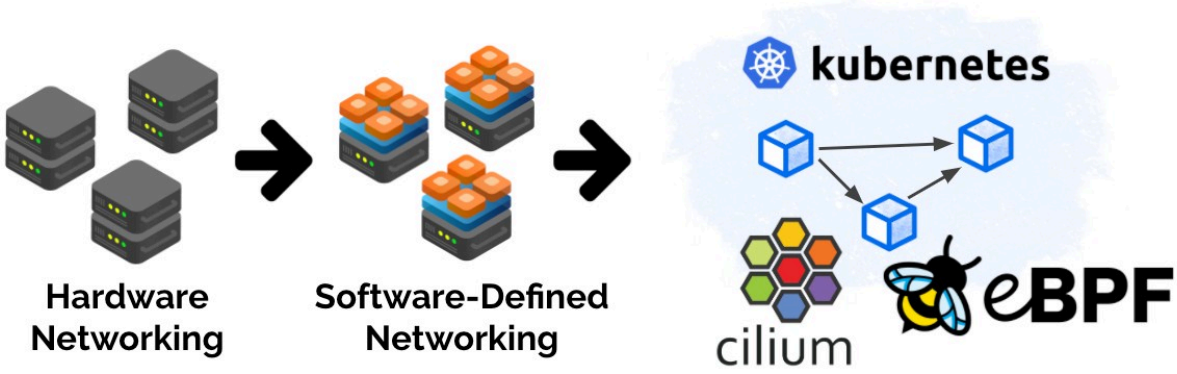
Kubernetes

Kubernetes 1.32 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

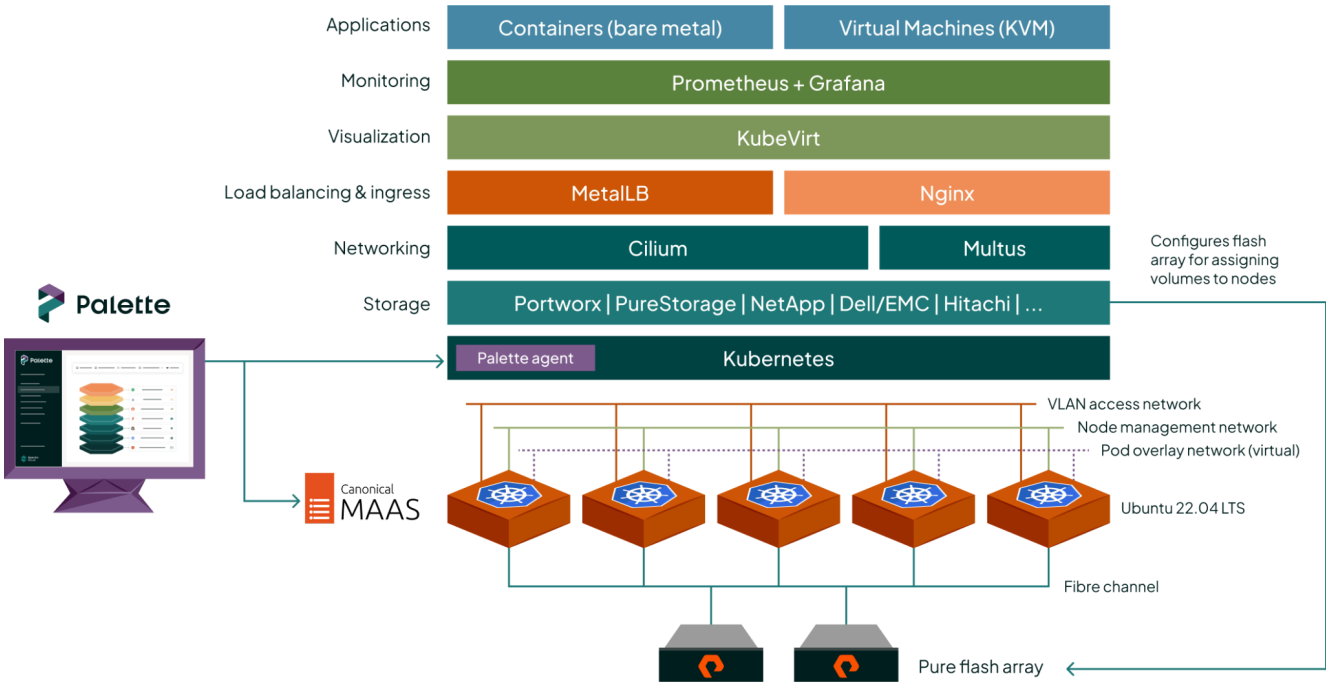
The CSI snapshot controller abstracts the need for storage volume snapshots into a generic concept, hiding the interaction with the actual CSI under the hood.

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	10:1 overcommitment of CPU by default
RAM	24 GB	256 GB or more	Up to 235 VMs per node as long as CPU/RAM capacity allows
Network Adapters	2× 10 Gbps (data+mgmt)	2× 10Gbps (data) 2× 10Gbps (mgmt)	Pod overlay runs on mgmt network
Storage Adapters	2× 16 Gbps FC or 2× 10 Gbps iSCSI	2× 16 Gbps FC or 2× 25 Gbps iSCSI	
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 typically 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. The control plane nodes can be virtual machines, through the MAAS-supported platforms (VMware, Proxmox, OpenStack, LXD, Virsh).

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 4 worker nodes as Portworx requires a minimum of 3 nodes to build a quorum, and you want 1 node to be able to go offline without impact. We recommend a minimum worker pool of 5 nodes.

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	3.6.0
Operating System	Canonical Ubuntu	22.04 LTS
Multipath software	multipath-tools	0.8.8
Kubernetes	Palette eXtended Kubernetes	1.32.4
Kubernetes networking	Cilium CNI	1.17.4
	Multus	4.1.4 (installed by VMO pack)
Kubernetes storage	Portworx Enterprise CSI	3.3.1
	Pure Storage FlashArray	Purity//FA 6.1.4 or higher
Virtualization UI	VM Orchestrator (VMO)	4.7.0
Virtualization	KubeVirt	1.5.0 (installed by VMO pack)
Image import	KubeVirt CDI	1.62.0 (installed by VMO pack)
Storage Snapshotting	CSI Snapshot Controller	8.1.0 (installed by VMO pack)
Cluster balancing	Descheduler	0.33.0 (installed by VMO pack)
Monitoring	Prometheus + Grafana	70.2.1
Ingress	Nginx	1.12.2
Load Balancing	MetalLB	0.15.2

Solution Sizing

This reference architecture is suitable for a VMO cluster that fits a combination of Virtual Machines and container-native workloads that total ~15.000 pods. The recommended upper limit of this configuration is **12.500 virtual machines**, which allows for just under 200 VMs per bare metal node. The actual number of VMs per node will depend on available CPU/RAM.

The solution is sized in the following manner:

- Up to 64 bare metal nodes in the cluster
  - 63 nodes can be active during normal operation
  - 1 node must be kept in spare to facilitate control plane repaves
- Kubernetes pod CIDR configured to 100.64.0.0/18
  - This allows for 64 sub-CIDRs of /24 each, one for each node, up to 64 nodes.
- Maximum number of pods configured to 250
  - Allowing 250 * 63 = 15.750 pods to be active
  - Last /24 subnet reserved for the control pane repave
- Overhead pods:
  - 58 pods for controllers, agents and other non-daemonset overhead
  - 10 pods per node for Cilium, Multus, VMO, monitoring and Portworx daemons
  - Total overhead (63*10)+58=688, leaving 15.062 pods for VMs and/or workloads.
  - If additional software (e.g. backup/restore) is installed, this will add a bit more overhead. VMs running with hot-added disks will also add an additional hot-add pod alongside the VM pod. This also eats into the VM pod limit.

This sizing is deemed a good balance between VM density and controlling the blast radius. Larger solutions are possible, mainly by using more bare metal nodes, but this requires using a larger Kubernetes pod CIDR than 100.64.0.0/18 in order to account for more nodes (each using a /24 slice of the pod CIDR). To facilitate this, the Kubernetes service CIDR must be changed as well. For example, to facilitate a 128-node cluster, suitable CIDRs would be:

- Pod CIDR: 100.64.0.0/17
- Service CIDR: 100.64.128.0/17

If the cluster is expected to grow, select a service CIDR that is intentionally further away, such as 100.65.0.0/x, so that the pod CIDR can be increased to support more nodes later.

Network configuration

Networking for the VMO use case requires some 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:

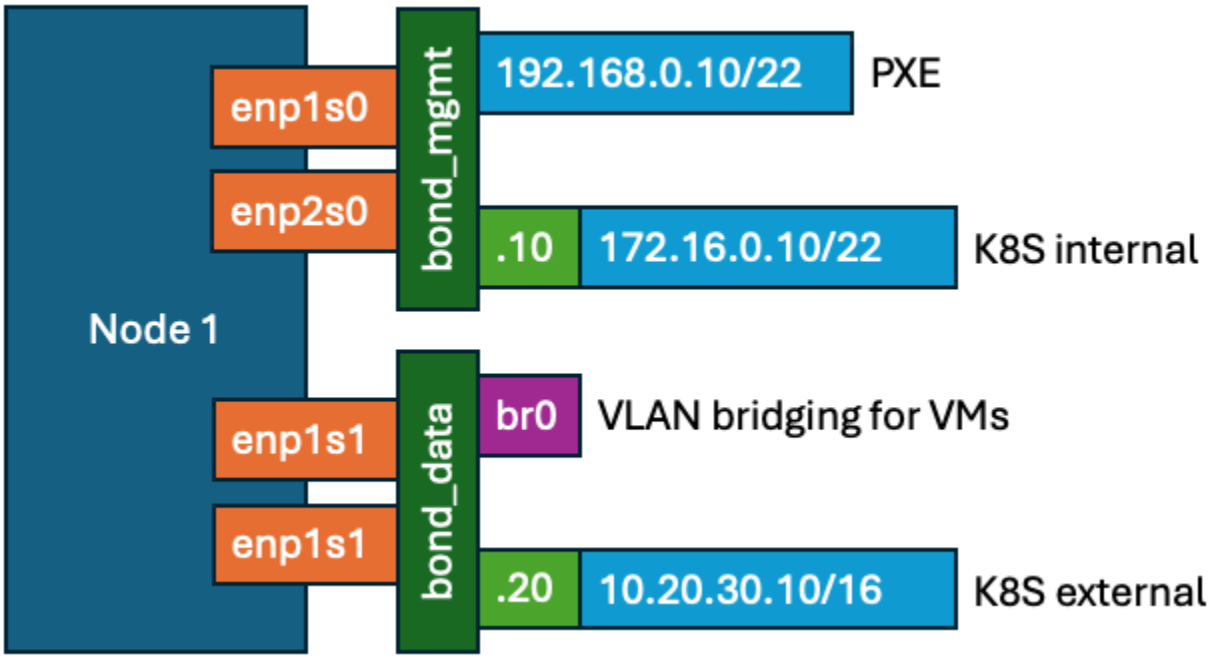
- As a network to advertise IP addresses directly onto, as Layer 2 advertisements.
- As a BGP network, where MetalLB can advertise BGP addresses to BGP routers.

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 of Kubernetes services, 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 (802.3ad)	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 (802.3ad)	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.

The **bond** interface that the **br0** interface is put on top of, can only support the following bonding modes:

- Mode 1: active-backup (does not require switch configuration)
- Mode 2: balance-xor (requires switch configuration)

- Mode 4: 802.3ad (requires switch configuration)

Our recommendation for best performance is the 802.3ad mode as this fully aggregates the bandwidth of the links. The other modes (0, 3, 5 and 6) are not supported for VLAN bridging due to broadcast storms, MAC address rewrites or poor TCP stream performance. See here for more information.

For the setup above, it is assumed that 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.

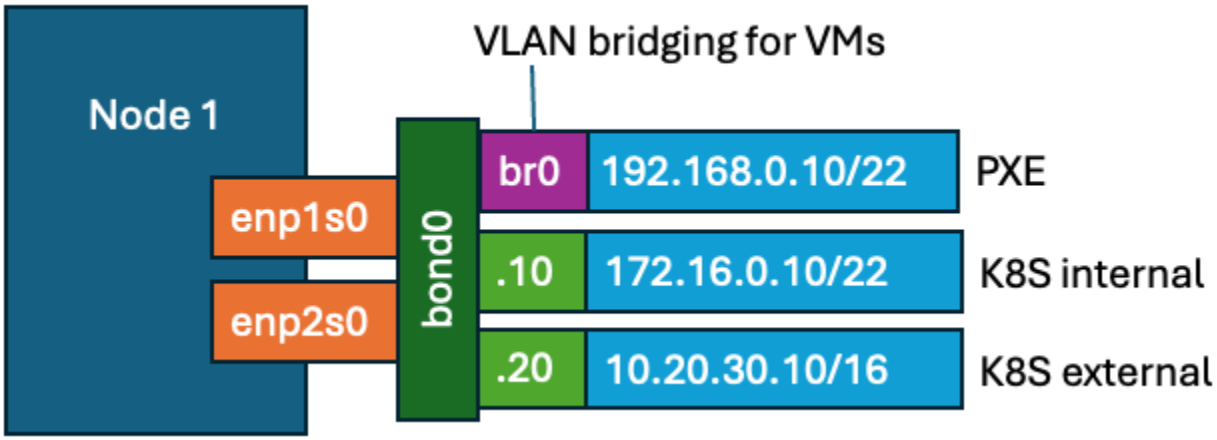
For publishing workloads from virtual machines there are two options:

1. 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. **This is the most-used and recommended option.**
2. Run the virtual machine on the pod network (as if it was a container) and publish individual ports of the VM as Kubernetes services either inside the cluster only or also externally on the **bond_data.20** network, using MetalLB to assign IP addresses. This approach is only suitable for workloads that can handle network disruptions well, as live migrations of VMs running on the pod network will cause existing network connections to that VM to be terminated.

When limited on network interfaces

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 (802.3ad)	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



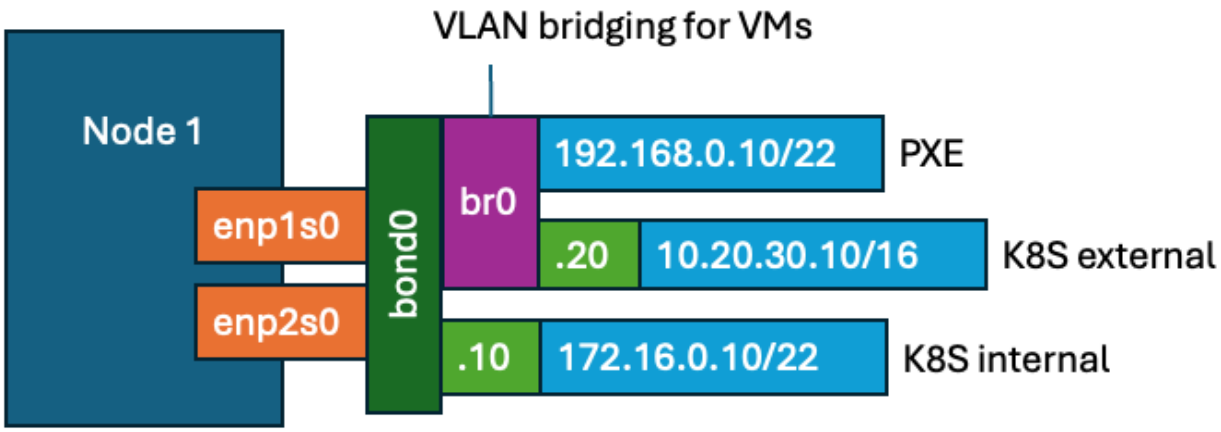
For the setup above, it is assumed that the physical servers (the worker nodes) are connected to the switch as follows:

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 VLANs 10 and 20 are already captured from the bond by the .10 and .20 subinterfaces on the bond, meaning that this traffic would never reach the bridge interface.

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 volume mode, either Block or FileSystem mode. The following storage providers are known to have the proper support:

Storage Provider	**Version**	**RWX for volume mode**
Portworx Enterprise	3.3.1	Block
Portworx CSI (for FlashArray DA)	25.4.0	Block
Piraeus (Appliance/Agent mode only)	2.9.x	Block
Rook-Ceph	1.16.x	Block
Longhorn	1.8.x	Block
Netapp Trident (iSCSI)	24.x	Block
Dell CSM Operator - PowerFlex	2.x	Block
Dell CSM Operator - PowerMax	2.x	Block
Dell CSM Operator - PowerStore	2.x	Block
Hitachi Storage Plugin for Containers	3.15.x	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 3.3 provides Block-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.

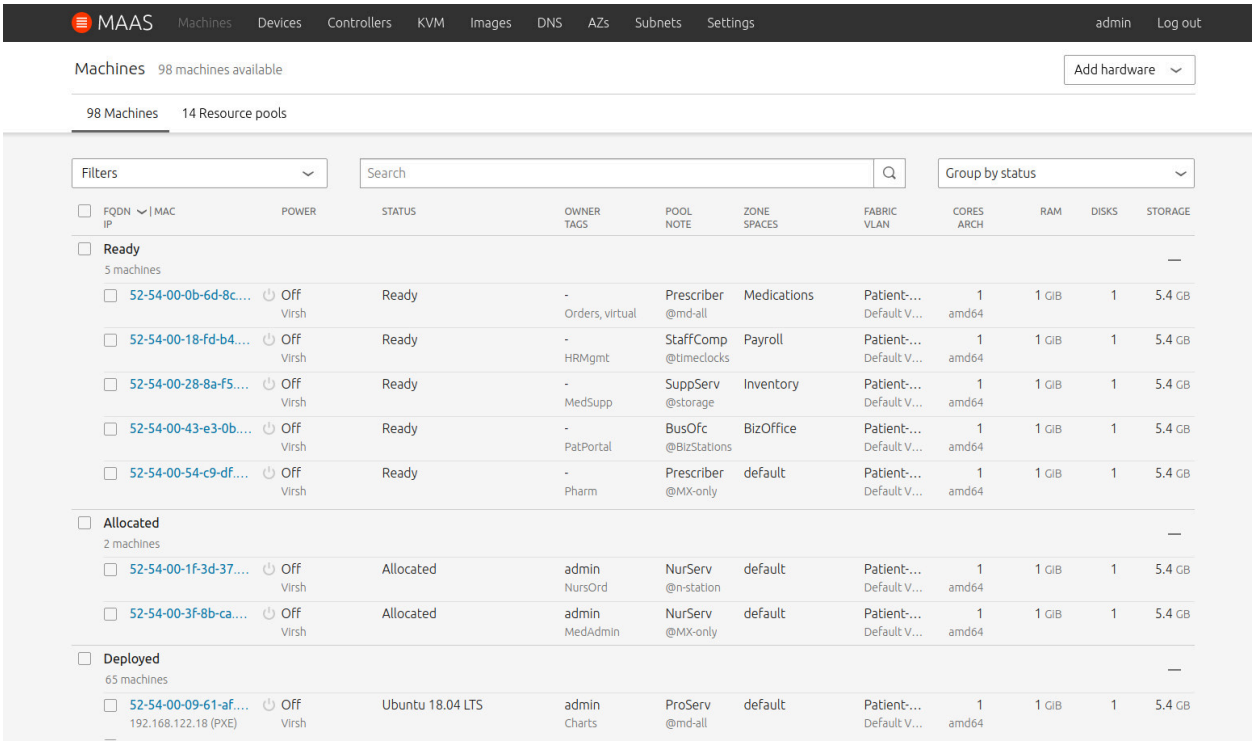
To ensure the Portworx configuration is suitable for our needs, we are setting the following specific options:

Setting	Value	Remarks
license.type	Enterprise	Cannot use Direct Access volumes as those do not support RWX.
annotations	portworx.io/misc-args: " -T px-storev2 "	Configures Portworx to use PxStorev2 mode for better performance.
env	PURE_FLASHARRAY_SAN_TYPE = FC	Informs Portworx to configure FlashArray for consumption via Fibre Channel..
cloudStorage.deviceSpecs	- size=1024	1 (or more) 1TB disks for data
cloudStorage.maxStorageNodes	[# of worker nodes]	Ensures quick failover of LUNs during node failure or cluster upgrades.
storageClass.parameters	io_profile: "auto_journal" priority_io: "high" repl: “2” nodiscard: true	Tune storageclass for VM block volumes and enable journal support.

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.



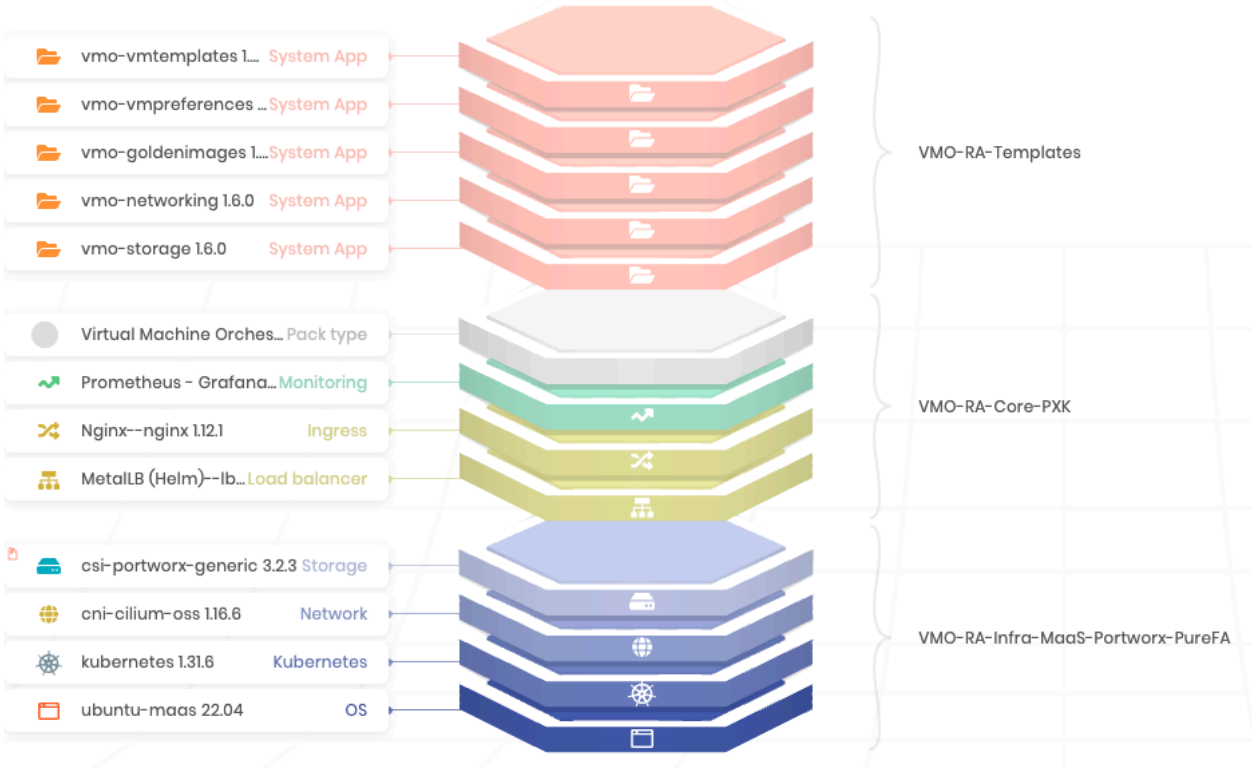
We will use the following features of MAAS to aid in the deployment of our VMO clusters:

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.
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**.

The following sections describe changes made to each layer in the profiles.

Ubuntu layer (VMO-RA-Infra-MaaS-PureFlashArray 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.

- In the `kubeadmconfig.preKubeadmCommands` section:
  - Add the node's IP address on the management network to /etc/default/kubelet, to ensure this IP becomes the InternalIP for this cluster node
  - Run `update-ca-certificates` so that custom CA certificates are trusted. 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.
  - Reload `multipathd` with the new configuration. 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.
  - Apply VMO tuning to worker nodes. This is skipped for control plane nodes so that those do not require additional CPUs and RAM to operate. It can be adjusted if control plane nodes will also run VMs.
  - Restart containerd and kubelet services to let new configurations take effect
- In the `kubeadmconfig.postKubeadmCommands` section:
  - Reload udev rules to let new udev rules for the Pure FlashArray go into effect.
- In the `kubeadmconfig.files` section:
  - VMO tuning files:
    - Move the containerd and kubelet services into a dedicated systemd kube.slice with CPU affinity for specific CPU cores (cores 0–7 by default).
    - Tuning parameters for sysctl that improve performance on large hosts running many workloads.

        - Tuned kubelet configuration that enables CPU Manager, Memory Manager and Topology Manager in Kubernetes, raises the maximum number of pods to 250 and reserves CPU cores and memory for the OS, Portworx & kubelet.
    - Configure containerd to include the device_ownership_from_security_context = true setting, which is required to properly handle block devices in Kubevirt.
    - Import a custom CA certificate. The provided certificate in the profile is an example and should be replaced by your custom CA certificate if you have one.
    - Configure multipath in accordance with Portworx best practices.
    - Add udev rules for Pure FlashArray in accordance with best practices.
    - Place a script on the system that can handle SAN boot scenarios when the user_friendly_names option is changed for multipath.

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-MaaS-PureFlashArray profile)

Pack Type	Kubernetes
Registry	Public Repo
Pack Name	Palette eXtended Kubernetes
Pack Version	1.32.4
Values	Modified from default

The modified aspects of the configuration of this layer are as follows:

- The OIDC Identity Provider is set to Palette. You should use an OIDC provider with VMO, otherwise anyone with access to the VM dashboard has full permissions. You can change this setting to use your own OIDC provider if so desired.
- The Pod CIDR is set to a profile variable for easy configuration. This variable defaults to `100.64.0.0/18` and is read-only. Refer to the Solution Sizing section on changing this.
- The Service CIDR is set to a profile variable for easy configuration. This variable defaults to `100.64.64.0/18` and is read-only. Refer to the Solution Sizing section on changing this.
- The `--config-dir` extra argument for kubelet is added, enabling the use of the `/etc/kubernetes/kubelet.conf.d` directory for drop-in configuration files (which is where the OS layer writes the kubelet tuning file)

Cilium layer (VMO-RA-Infra-MaaS-PureFlashArray profile)

Pack Type	Network
Registry	Public Repo
Pack Name	Cilium
Pack Version	1.17.4
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
Pod networking	Use VXLAN Overlay
VMO Compatibility	Enable
VMO - Bridge Interface	Autodetect Cilium Interface
Loadbalancer mode	No XDP Acceleration

Cilium is configured to leverage eBPF to replace kube-proxy for better performance in clusters with large amounts of service resources. The pod network still runs over a VXLAN overlay for maximum compatibility.

Additional changes made to the Cilium layer are:

- Four pod annotations are set in `podAnnotations` in order to ensure smooth upgrades from earlier Cilium versions.
- The `forceDeviceDetection` option is enabled, which allows scenarios where worker nodes need the "Run Cilium on Bridge (br0)" preset to be active, while also using control plane nodes that do not have a br0 network adapter. With `forceDeviceDetection` enabled, Cilium will detect the regular NIC on the control plane node and use that instead.
- The `envoy.enabled` option is set to false, since we are using MetalLB for loadbalancing. This saves an unnecessary pod from running.

Portworx layer (VMO-RA-Infra-MaaS-PureFlashArray profile)

Pack Type	Storage
Registry	Public Repo
Pack Name	Portworx /w Operator
Pack Version	3.3.1
Values	Modified from default

The following pack presets were selected:

License Models	PX Enterprise
Node selection	Run on worker nodes only
Palette Mode	This is a CAPI or Appliance mode cluster
Storagecluster Specs	Pure Storage Flash Array
Kvdb and Etcd	Use Internal Kvdb, No TLS
HTTP(S) Proxy	*<not enabled>*
Custom CA certs	*<not enabled>*

The modified aspects of the configuration of this layer (beyond the selected presets) are as follows:

- A dummy license key is set for the `license.enterprise.activationId` parameter. This layer defaults to the Portworx Enterprise license model. If you use a Portworx SaaS Pay-As-You-Go key, change the License Models preset to SaaS and enter the key for the `license.saas.key` parameter instead.
- The StorageCluster configuration is adjusted:
  - Portworx PxStorev2 is enabled for higher performance. This requires a dedicated 64GB drive to be used as the Metadata Disk, which will be automatically requested from the FlashArray through the `storageCluster.spec.cloudStorage.systemMetadataDeviceSpec` parameter.
  - The environment variables for the StorageCluster are modified:
    - The `PURE_FLASHARRAY_SAN_TYPE` is changed from `ISCSI` to `FC`

        - The `KUBERNETES_OPS_QPS_RATE` and `KUBERNETES_OPS_BURST_RATE` parameters are added to ensure stability in busy environments
    - A `storageCluster.spec.network` section is added 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 `storageCluster.spec.network.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.
    - The `storageCluster.spec.cloudStorage` spec is set to our specific needs:
        - Setting `maxStorageNodes` to a profile variable for easy configuration. At cluster deployment, set this variable to 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. Adjust this as necessary.
        - Setting the `systemMetadataDeviceSpec` to 64GB
        - Setting the `journalDeviceSpec` to 3GB
        - Setting the `provider` to `pure` as recent Portworx versions now require the provider to be configured instead of auto-detected.
- Adjusting the configuration of the `storageClass` resource:
    - Setting an annotation to inform Kubevirt that this storageclass should be the default storageclass for VMs
    - Setting `parameters.priority_io: "high"` to match the priority of the FlashArray pool that Portworx will set up.
    - Setting `parameters.io_profile: "auto_journal"` to enable journaling for better performance.
    - Setting `parameters.nodiscard: true` to align with recommend settings for VM block volumes
    - Reducing `parameters.repl` from 3 to 2 for better performance.

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`

MetalLB layer (VMO-RA-Core-PXKE profile)

Pack Type	Load balancer
Registry	Public Repo
Pack Name	MetalLB (Helm)
Pack Version	0.15.2
Install order	10
Values	Modified from default

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

- `configuration.ipaddresspools.first-pool.spec.addresses` is set to a cluster profile variable for easy configuration. The default value for the value is a block of addresses in the **bond_data.20** network.
- `configuration.l2advertisements.default.spec.interfaces` is set to a cluster profile variable for easy configuration. The default value for the value is the **bond_data.20** interface, which instructs MetalLB to advertise the IPs on only that interface (by default, MetalLB advertises on all interfaces, which is often undesirable).
- `metallb.speaker.ignoreExcludeLB` is set to `true` so that control plane nodes are able to advertise MetalLB L2 addresses.

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-PXKE profile)

Pack Type	Ingress
Registry	Public Repo
Pack Name	Nginx
Pack Version	1.12.2
Install order	10
Values	Modified from default

The modified aspect of the configuration for the Nginx layer is as follows:

- Added two options for compatibility with Palette virtual clusters (vCluster):
  - `controller.extraArgs.enable-ssl-passthrough: true`
  - `tcp.6443: "nginx/nginx-ingress-nginx-controller:443"`

This enables coexistence with Palette Virtual Clusters using the Ingress method for external access. It ensures that Nginx also listens on port 6443, so that kubeconfig files for virtual clusters (generated by Palette) can be used without modification.

Prometheus layer (VMO-RA-Core-PXKE profile)

Pack Type	Monitoring
Registry	Public Repo
Pack Name	Prometheus - Grafana
Pack Version	70.2.1
Install order	20
Values	Modified from default

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

- The mandatory value for `grafana.adminPassword` has been set to “welcome”
- `grafana.ingress` has been enabled and configured, in order to not consume an additional MetalLB IP address and to serve the Grafana site over HTTPS only.
- A manifest is added to the Prometheus layer to create a self-signed Issuer (cert-manager) in the `monitoring` namespace, which will generate the SSL certificate for the Ingress.

VMO layer (VMO-RA-Core-PXKE profile)

Pack Type	System App
Registry	Public Repo
Pack Name	Virtual Machine Orchestrator
Pack Version	4.7.0
Install order	30
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:

- All built-in templates in `sampleTemplates` are disabled. The reference architecture provides its own templates in the VMO-RA-Templates add-on profile.
- The `vlanFiltering` section is enabled, which configures the **br0** interface on each worker node to become VLAN-aware, making it possible to place VMs directly on a VLAN.
    - The env.allowedVlans parameter is set to a cluster profile variable for easy configuration. It defaults to "21–100" to allow VMs to be placed on those VLANs, and can be adjusted as necessary.
    - The env.allowVlansOnSelf toggle is set to a cluster profile variable for easy configuration. Enable this option when the **br0** interface on the host has an IP address or has VLAN-subinterfaces defined.
    - The env.allowedVlansOnSelf parameter is set to a cluster profile variable for easy configuration. It only takes effect if the allowVlansOnSelf toggle is enabled. When

that is the case, the associated cluster profile parameter ("VLANs on top of br0") should be set to the range of VLANs that will be used directly by the host on **br0**. To determine which VLANs should be defined:
    - If the **br0** interface has an IP address itself, enable VLAN 1.
    - If the **br0** interface has VLAN subinterfaces defined, also enable those VLANs
  - The VLANs for VMs will be automatically added to the `env.allowedVlansOnSelf` parameter and thus do not need to be defined twice. We add these because otherwise the VLAN-aware bridge will refuse to pass the VLANs for VMs when `allowVlansOnSelf` is enabled.
- The `snapshot-controller.volumeSnapshotClass` section is enabled and configured for Portworx, including some parameters that enable compatibility with Velero for backup/restore.
- Add additional feature gates for persistent TPM, CPU Manager and strict HyperV checking.
- Configure the storageclass to use to store persistent TPM data

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"
```

We therefore set this annotation in the VM preferences in the VMO-RA-Templates profile.

VMO-Storage layer (VMO-RA-Templates profile)

Pack Type	Manifest
Manifest Name	storageprofile-cdi
Install order	40

The VMO-RA-Templates profile contains 5 manifest layers to round out certain aspects of the VMO solution. The following manifest is included in the vmo-storage layer:

Name(s)	Type	Purpose
storageprofile-cdi	storageProfile	A storage profile for CDI that informs it to use RWX volumes for uploaded images and use fast CSI cloning for cloning volumes.

VMO-Networking layer (VMO-RA-Templates profile)

Pack Type	Manifest
Manifest Name	nad-vlan-30 nad-vlan-untagged
Install order	40

The VMO-RA-Templates profile contains 5 manifest layers to round out certain aspects of the VMO solution. The following manifests are included in the vmo-networking layer:

Name(s)	Type	Purpose
nad-vlan-30	NetworkAttachmentDefinition	A sample NAD for VLAN 30 access
nad-vlan-untagged	NetworkAttachmentDefinition	A sample NAD for untagged VLAN access

The manifests in this layer will likely require adjustment to the environment in which they are deployed:

- **nad-vlan-30**:
  - This is an example manifest to make VLAN 30 available for end users to place their VMs on. Copy the YAML from this example to create new manifests for your own VLANs.
- Create another nad-vlan-xxx manifest for every other VLAN that should be made available:
  - Ensure that line 12 matches the name of the VLAN-aware bridge interface on your cluster nodes (should be “br0”)
  - Adjust line 13 to reflect the VLAN ID you want to make available
  - Adjust lines 4 and 10 to reflect the name of this network that end users should see.
- Delete nad-vlan-30 if no longer needed

VMO-GoldenImages layer (VMO-RA-Templates profile)

Pack Type	Manifest
Manifest Name	dv-ubuntu-2204 dv-windows-2022
Install order	40

The VMO-RA-Templates profile contains 5 manifest layers to round out certain aspects of the VMO solution. The following manifests are included in the vmo-goldenimages layer:

Name(s)	Type	Purpose
dv-ubuntu-2204	DataVolume	A sample DataVolume for Ubuntu 22.04 to be used as a golden image for quickly provisioning VMs
dv-windows-2022	DataVolume	A sample DataVolume for Windows Server 2022 (Evaluation Edition) to be used as a golden image for quickly provisioning VMs

VMO-VMPreferences layer (VMO-RA-Templates profile)

Pack Type	Manifest
Manifest Name	vmcp-generic-linux vmcp-generic-windows
Install order	40

The VMO-RA-Templates profile contains 5 manifest layers to round out certain aspects of the VMO solution. The following manifests are included in the vmo-vmpreferences layer:

Name(s)	Type	Purpose
vmcp-generix-linux	VirtualMachineClusterPreference	VM preferences for Linux that optimize performance and reduce overhead
vmcp-generic-windows	VirtualMachineClusterPreference	VM preferences for Windows that optimize performance and reduce overhead

VMO-VMTemplates layer (VMO-RA-Templates profile)

Pack Type	Manifest
Manifest Name	cm-unattend-xml vmtemplate-ubuntu-2204 vmtemplate-windows-2022
Install order	40

The VMO-RA-Templates profile contains 5 manifest layers to round out certain aspects of the VMO solution. The following manifests are included in the vmo-vmtemplates layer:

Name(s)	Type	Purpose
cm-unattend-xml	ConfigMap	A sample sysprep file for Windows 2022 in both the default and the virtual-machines namespace
vmtemplate-ubuntu-2204	VmTemplate	A sample VmTemplate for a Ubuntu 22.04 VM using a pre-imported golden image
vmtemplate-windows-2022	VmTemplate	A sample VmTemplate for a Windows 2022 VM using a pre-imported golden image