PURESTORAGE®

Red Hat

WHITEPAPER

Portworx Enterprise with Red Hat OpenShift Container Platform Two-Node w/Arbiter

Contents

Executive Summary

Edge workloads are increasingly evaluating and adopting Kubernetes based solutions. According to Gartner, 50% of enterprises will initiate PoCs by 2026 to phase out their existing VMware based deployments (up from 10% in 2024) to adopt distributed hybrid infrastructure (DHI), with Edge playing an important role.

Kubernetes plays a fundamental role in realizing this vision of DHI as it enables consolidation of containerized and virtualized workloads on the same infrastructure. This is especially true for Edge given the emergence of new use cases (5G, AI, etc.) not just traditional use cases of Retail, Industrial, Healthcare, Defense, etc.

Red Hat announced two new deployment models with OpenShift, namely - Two-Node with Arbiter (TNA) and Two-Node with Fencing (TNF). These allow Red Hat to service these specialized use cases deployed at Edge.

Portworx and Red Hat have partnered together to co-engineer a solution using Portworx Enterprise storage in combination with Red Hat OCP in a TNA deployment model to enable our mutual customers to run containerized and virtualized workloads on a single technology stack that can be deployed in Multi-cloud, On-prem and/or Edge environments.

This whitepaper is intended to educate our mutual customers how to leverage Portworx Enterprise with Red Hat OpenShift to provide storage for these workloads running on the TNA architecture in Edge and Far Edge environments irrespective of underlying Server HW platform.

The scope of this white paper is to document the various aspects of this co-engineered solution, including SW configuration, a high level system setup which is agnostic of any underlying HW, and last but not the least, potential cost savings. This white paper does not aim to cover the Two-Node with Fencing (TNF) architecture.

Business Value and Outcomes

This co-engineered solution helps customers run *any* containerized and/or virtualized workload on *any* commodity-off-the-shelf HW that can be deployed in Edge environments that require a cost-effective, limited compute and/or storage solution that provides high availability and resilience to tolerate single failures while ensuring business continuity. This joint solution eliminates the need to maintain separate infrastructure for containerized and virtualized workloads, or staffing separate technical teams to operate and maintain these two separate environments.

This joint solution also enables customers to run these demanding workloads with the same level of programmability, elasticity, availability and resiliency as *any* legacy hypervisor, but at a small fraction of the licensing cost of running it on one.

Features critical for Edge deployments

Typically, these deployments don't require a large capacity or extreme performance, nor do they require niche features such as role based access control, application IO control, multitenancy, etc.

However, they do require cost-efficient compute and storage that delivers *consistent performance*, ability to quickly *recover from failures*, and high-availability, etc. when deployed on commodity, off-the-shelf hardware.

Protection against [*single*] failures (*unplanned failures*)

HW failures are a fact of life (CPU, Motherboard, Disk, NIC, DIMM, etc.). Applications deployed in these edge environments ***must*** be able to run and service requests even in the face of such failures (unplanned events). Portworx supports replication of data such that in the event of an unplanned failure, the applications can be successfully brought back up on the surviving host *without* any loss of data.

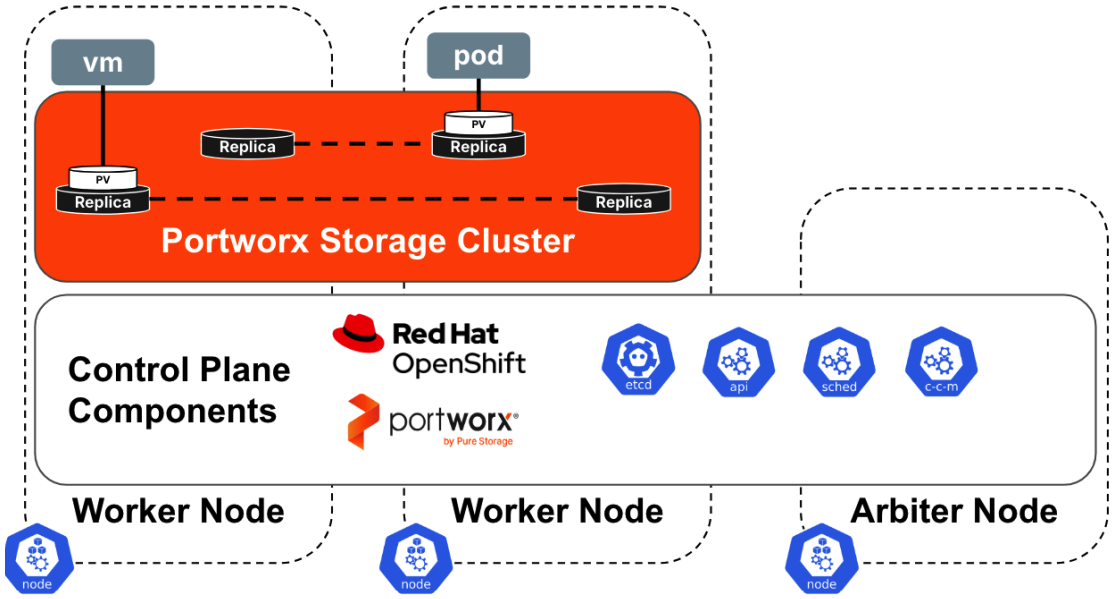

Figure 1. Portworx & Red Hat Two-Node Arbiter Architecture Diagram

Host upgrades and planned maintenance

Live Migration is a critical feature to ensure that if and when a host has to be taken down for a planned maintenance event (service HW failure, perform SW/FW upgrades, apply application patches, etc.), these mission critical applications can continue to run uninterrupted by live migrating them to the surviving worker node during this window, and migrate them back upon successful completion of aforementioned maintenance.

Cost-effective, commodity HW

Moreover, in order to keep the costs down, these applications must be able to run on commodity hardware and be able to deliver consistent performance even during failures or maintenance to ensure that these businesses are able to meet and exceed all requirements and objectives without any hiccups on the most cost-effective hardware.

Observability and Manageability

Typically there are hundreds of such small clusters in a customer's edge environment that requires administrators to monitor, administer and troubleshoot these vast numbers of clusters remotely, as well as perform lifecycle management operations from a single pane of glass. Red Hat ACM enables one to monitor and administer hundreds of such clusters remotely from a single pane of glass.

Supportability

Last but not the least, in the event of a HW failure, any [3rd party] field technician with minimal training should be able to perform simple break-fix operations to service and maintain this HW in remote environments (e.g., replace a broken part / **F**ield **R**eplaceable **U**nit). Moreover, by sending telemetry to Pure 1, one could troubleshoot these clusters remotely.

Proposed Solution

Portworx Enterprise provides container-native data management platform for virtualized or containerized applications running on Kubernetes, including Red Hat OpenShift Container Platform.

Red Hat TNA is a deployment option which leverages Red Hat OpenShift Container Platform. These deployments, in conjunction with Portworx, can be visualized as follows:

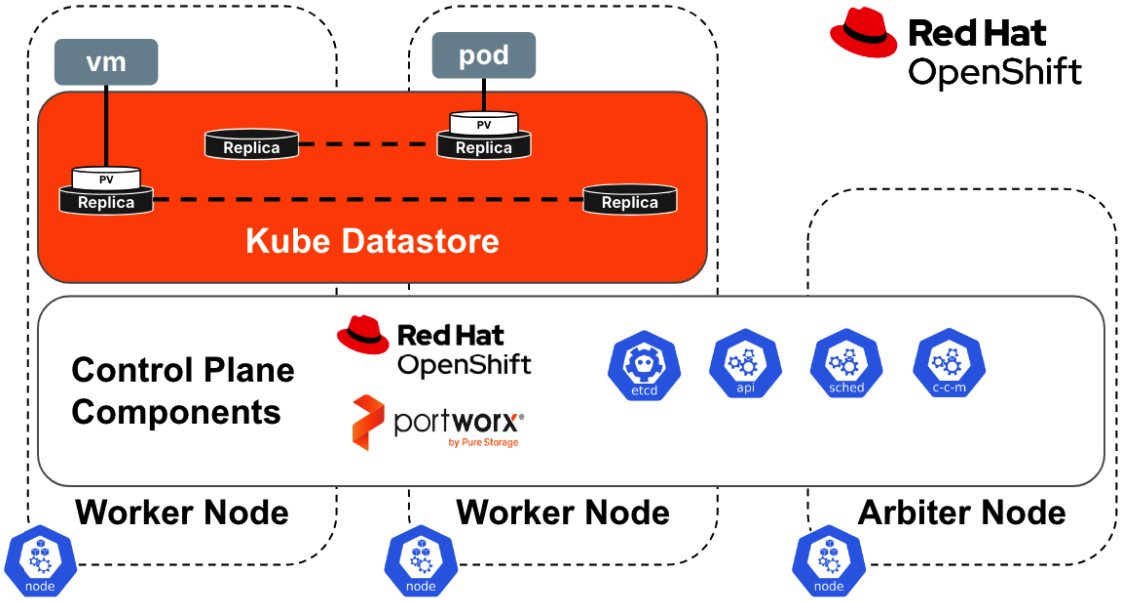

Figure 2. Portworx & Red Hat Two-Node Arbiter Architecture Diagram

When you step back and look at this configuration, this is essentially a 3-node Portworx cluster, with the restriction that no volume replica be placed on the Arbiter Node (i.e., it serves as a Storageless node and is used for storing Portworx metadata). Red Hat OCP ensures no VM or Container workload gets placed on the Arbiter node as well.

We have validated this solution in a lab environment leveraging commodity off-the-shelf servers, with an SOC based Arbiter node that had limited system resources.

Cluster Configuration

Total Nodes in cluster	Storage Nodes	Arbiter Nodes
3	2	1

Table 1. Cluster Configuration

Software Versions

Software	Version
Portworx Enterprise	PXE 3.5.0
Red Hat Enterprise Linux CoreOS	RHCOS 9.6
OpenShift Container Platform	OCP 4.20 (GA)

Table 2. Software Versions

Hardware Configuration

Description	Worker Node	Arbiter
CPU Cores (Physical, Minimum)	8	4
CPU Cores (Physical, Recommended)	16	4
Networking (Minimum)	1 Gbe	
Networking (Recommended)	10 Gbe	
Network latency (Maximum)	10ms	200ms
Memory (Minimum)	8 GiB	8 GiB
Memory (Recommended)	16 GiB	8 GiB
Storage (Minimum)	312 GiB	184 GiB
Storage (Recommended)	>312+ GiB	184 GiB
# Root disks	1	1
# Metadata disks	1	1
# Data disks (minimum)	1	0

Table 3. Hardware Configuration

Storage

The storage on each of the worker nodes and the Arbiter node can be visualized as follows.

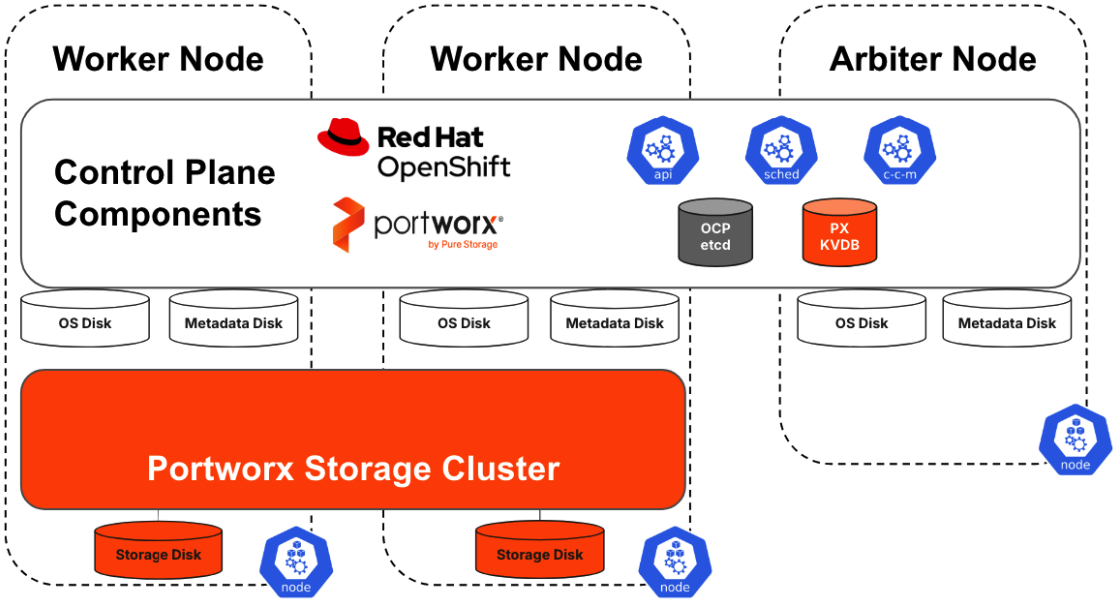

Figure 3. Portworx & Red Hat Architecture Diagram

Description	Storage Nodes	Arbiter Nodes
OS Disk (Root Disk)	120 GB	120 GB
Metadata Disk	64 GB	64 GB
Storage disk (data disk)	128 GB	-Not Applicable-

Table 4. Storage Node Requirements

System Limits

With Portworx Enterprise Release 3.5.0, in addition to enabling support for Red Hat's TNA deployment model, we have also increased the number of concurrent volume attachments per node from 256 to 1024 enabling higher workload consolidation, provided the underlying HW platform can support it.

Description	Node	Arbiter
Persistent Volumes/Node	1024	0

Table 5. System Limits

Kubernetes Worker and Arbiter Node Configuration

Node Type	Label
Storage Node	portworx.io/node-type=storage
Arbiter Node	portworx.io/node-type=storageless
Master Node	portworx.io/run-on-master : "true"

Table 6. Kubernetes Worker and Arbiter Node Configuration

To ensure that no replica is placed on the Arbiter node, one needs to include the following selector in the StorageCluster spec (replace the below-mentioned device names with the device names in your node):

```
nodes:
  - selector:
      labelSelector:
        matchLabels:
          kubernetes.io/hostname: “os-tna3-arbiter”
    storage:
        devices: []
      useAll: false
      kvdbDevice: /dev/sda
```

One needs to configure the Kubernetes cluster to assign roles to the Worker and Arbiter nodes. This will ensure that PVs (replicas) are only created on Worker nodes, and not on Arbiter nodes. Furthermore, VMs and/or containers will only be scheduled on Worker nodes, and will not be scheduled on Arbiter node due to the taint applied to it.

Portworx Configuration

Apply belowmentioned tolerations on the Operator pods at time of deployment (before creation of STC) under Spec. Template.Spec:

```
tolerations:
      - key: “node-role.kubernetes.io/arbiter”
        effect: “NoSchedule”
      - key: “node-role.kubernetes.io/master”
        effect: “NoSchedule”
      - key: “node-role.kubernetes.io/control-plane”
        effect: “NoSchedule”
```

Conclusion

Red Hat TNA combined with Portworx provides customers an optimized, cost-effective alternative to deploy their virtualized workloads in Kubernetes environments at a fraction of the cost of deployment on a legacy hypervisor while maintaining the same levels of high availability, resilience, and performance.

Appendix 1

Sample Storage Class Spec

The Portworx **StorageCluster** is a Kubernetes Custom Resource Definition (CRD) that declaratively defines every aspect of a Portworx deployment—from storage topology and networking to security, telemetry, and upgrade strategy. By encapsulating the entire Portworx configuration in a single YAML object, the StorageCluster spec allows platform teams to manage the lifecycle of the storage layer just like any other Kubernetes workload.

The StorageCluster spec below is an example specification showing Portworx Enterprise deployed on a two-node arbiter architecture. There are two major differences to notice between a traditional StorageCluster specification and the example below for Two-Node Arbiter:

- The arbiter node doesn't define storage for anything other than the metadata device used to house Portworx Enterprise's KVDB.
- Placement rules are in place to ensure the Portworx control plane can run on arbiter and control plane nodes.

```
kind: StorageCluster
apiVersion: core.libopenstorage.org/v1
metadata:
 name: px-cluster
 namespace: portworx
 annotations:
   portworx.io/is-openshift: “true”
   portworx.io/misc-args: “ -T px-storev2 “
   portworx.io/run-on-master: “true”
   portworx.io/preflight-check: “skip”
   portworx.io/health-check: “skip”
spec:
 image: portworx/oci-monitor:3.5.0
 imagePullPolicy: Always
 kvdb:
   internal: true
 nodes:
 - selector:
     labelSelector:
```

```
      matchLabels:
        kubernetes.io/hostname: “os-tna1”
  storage:
    devices:
      - /dev/sda
    systemMetadataDevice: /dev/sdb
- selector:
    labelSelector:
      matchLabels:
        kubernetes.io/hostname: “os-tna2”
  storage:
    devices:
      - /dev/sda
    systemMetadataDevice: /dev/sdb
- selector:
    labelSelector:
      matchLabels:
        kubernetes.io/hostname: “os-tna3”
  storage:
      devices: []
    useAll: false
    kvdbDevice: /dev/sda
placement:
  tolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
```

```
  - effect: NoSchedule
    key: node-role.kubernetes.io/control-plane
  - effect: NoSchedule
    key: node-role.kubernetes.io/arbiter
secretsProvider: k8s
startPort: 17001
stork:
  enabled: true
  args:
    webhook-controller: “true”
autopilot:
  enabled: true
csi:
  enabled: true
monitoring:
  telemetry:
    enabled: true
  prometheus:
    enabled: true
    exportMetrics: true
```