PURESTORAGE®
Uncomplicate Data Storage, Forever

REFERENCE ARCHITECTURE

AlloyDB Omni with Pure Storage and OpenStack

A reference architecture for AlloyDB Omni on Kubernetes with Portworx® and OpenStack

Contents

Executive Summary 3

Introduction 3

- Solution Overview 3

Solution Benefits 4

- Modular Architecture 4
- Maximizing Cloud-native Potential 4
- Cloud-agnostic Deployment 4
- Scalability 5
- High Availability 5
- All-flash Enterprise Storage 5

Technology Overview 6

- Compute Resources 6
- Network Resources 6
- Pure Storage FlashArray 7
- Portworx Enterprise 7
- OpenStack 7
- Kubernetes 8
- AlloyDB Omni 8

Technical Solution Design 9

- Compute Layer 10
- Nova Instances 10
- Storage Layer 11
- AlloyDB Omni Design Considerations 11

Design Validation 13

- Configuration 13
- Results 15

Deployment 16

- Deployment Requirements 16
- OpenStack 16
- Kubernetes 27
- Portworx Enterprise 28
- AlloyDB Operator and AlloyDB Omni 30

Conclusion 31

Additional Resources 31

Executive Summary

In the landscape of modern enterprise computing, the demand for high-performance, scalable, and resilient databases capable of effectively managing hybrid transactional and analytical workloads is becoming essential. Traditional databases often falter under the weight of requirements set by container-based solutions, struggling to keep pace with the agility and elasticity these environments demand.

The evolution of container-based solutions has fundamentally transformed the way organizations construct, deploy, and manage their applications. At the heart of this revolution is Kubernetes, the industry standard for container orchestration, which provides developers with tools to achieve levels of scalability, flexibility, and agility previously unattainable. Despite its numerous benefits, Kubernetes introduces complex challenges, particularly in the areas of data persistence, high availability, and resilience. These challenges underscore the necessity for database solutions that are not only robust and scalable but also sufficiently adaptable to accommodate the dynamic needs of contemporary enterprises.

In this reference architecture, Pure Storage® FlashArray™, Portworx®, and AlloyDB Omni are components of a larger solution that solves these challenges that has been tailored for Kubernetes environments operating on OpenStack, an infrastructure-as-a-service (IaaS) platform that can be deployed in on-premise, public, and private clouds. Incorporating Pure Storage FlashArray as the backend storage further enhances the robust capabilities of AlloyDB Omni, a versatile database optimized for both transactional and analytical workloads. By leveraging the advanced stateful storage solutions provided by Portworx, AlloyDB Omni ensures seamless data management and high-performance storage, empowering businesses to efficiently handle diverse workloads while maintaining reliability and scalability. This integration facilitates the creation of a database landscape that not only meets but excels in the realms of high availability and data persistence.

Introduction

This reference architecture offers a comprehensive solution that leverages the full potential of Kubernetes deployments for AlloyDB Omni databases, paired with Portworx storage solutions on OpenStack, a cloud computing platform that provides a set of software tools for building and managing on-premise, public, and private cloud infrastructure. It outlines a detailed, structured framework that provides vital insights into the deployment processes, configuration considerations, and optimization strategies essential for achieving optimal outcomes in enterprise environments. Through this architecture, organizations are equipped to navigate the complexities of modern database deployment, ensuring robustness, scalability, and operational excellence.

Solution Overview

This reference architecture consists of multiple interconnected components (shown in Figure 1) starting with AlloyDB Omni deployed using the AlloyDB operator in Kubernetes running in Ubuntu-based OpenStack Nova instances. Stateful storage is provided to AlloyDB Omni using Portworx. In this specific solution, Portworx stateful storage is built using cinder volumes, which in turn are provisioned from FlashArray block storage volumes.

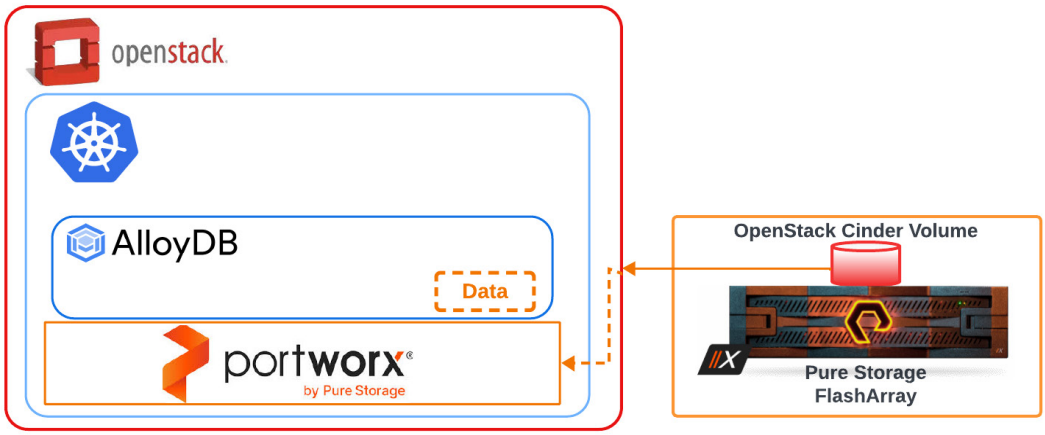


FIGURE 1 High-level overview of the interconnected component architecture

Solution Benefits

This solution provides a range of benefits, enhancing reliability and cost-efficiency for managing AlloyDB Omni workloads in container environments. These features collectively foster a robust, scalable, and flexible environment for deploying AlloyDB Omni, making it an ideal choice for enterprise container management. The primary benefits include the following aspects.

Modular Architecture

The system is designed with autonomous components, each featuring well-defined interfaces that streamline communication and interaction across the system. This modular design supports seamless upgrades, maintenance, and customization, maintaining system flexibility and adaptability to meet changing requirements.

Maximizing Cloud-native Potential

OpenStack is distinguished by its simplicity and robust security measures, including network isolation, access controls, and encryption. Organizations can effortlessly scale their cloud infrastructure to accommodate increasing needs while retaining the adaptability to address evolving demands. Its open-source character promotes collaboration and creativity, ensuring ongoing enhancements and alignment with industry standards. OpenStack delivers cost-effectiveness by serving as an economical option compared to proprietary cloud solutions, empowering organizations to efficiently manage resource allocation and minimize expenditure.

The combination of Kubernetes and OpenStack offers an effective platform for deploying and managing cloud-native applications along with numerous benefits like application portability across different infrastructure environments, seamless workload migration and enhanced security for organizations embracing cloud-native solutions to unlock the full potential of cloud-native computing.

Cloud-agnostic Deployment

Using this reference architecture, AlloyDB Omni with Portworx can be implemented on any infrastructure that supports Kubernetes and OpenStack, offering tremendous flexibility and preventing vendor lock-in. This approach reduces reliance on specific cloud services, enhances system resilience, facilitates seamless operation across multiple cloud platforms, and simplifies the migration and portability of workloads while reducing operational complexities.

Scalability

Portworx plays a crucial role in the scalability of the system, enabling organizations to swiftly expand their storage infrastructure in response to growing demands. It allows for the expansion of existing volumes without disrupting ongoing applications. The Autopilot feature in Portworx uses a rule-based engine to dynamically manage storage capacity, adapting to storage events to efficiently scale storage for individual container volumes or entire storage pools according to customized rules.

High Availability

Portworx offers a robust high availability (HA) storage layer for Kubernetes container environments, ensuring continuous data access and operational resilience even during hardware failures, node outages, or other disruptions. Key features include:

- **Data replication**: Portworx uses synchronous replication to maintain accurate and durable copies of AlloyDB Omni data across multiple nodes, enhancing data safety against node failures and eliminating single points of failure.
- **Auto-failover:** Portworx integrates with the STORK scheduler to detect node failures and quickly initiate an automated failover process. This transition minimizes downtime and maintains data availability, crucial for single-node deployments of AlloyDB Omni.
- **Data resynchronization**: After disruptions like node failures or network issues, Portworx undertakes data resynchronization to ensure all data copies across storage volumes remain consistent and up-to-date, crucial for maintaining uninterrupted operations and data integrity in dynamic containerized settings.

All-flash Enterprise Storage

The incorporation of FlashArray into this solution further enhances availability and increases overall performance. A FlashArray system's architecture is designed for optimal performance and reliability, supporting critical data operations without downtime. Its all-flash storage capabilities ensure fast data access and processing, critical for performance-sensitive applications. Moreover, its native support for non-disruptive operations allows for hardware and software upgrades without affecting the ongoing workload performance, ensuring that the AlloyDB Omni environment remains highly available and robust under all conditions.

Technology Overview

This reference architecture is a set of loosely coupled components that communicate with one another over redundant high-speed ethernet connections (Figure 2). OpenStack hosts can access block storage using Cinder volumes.

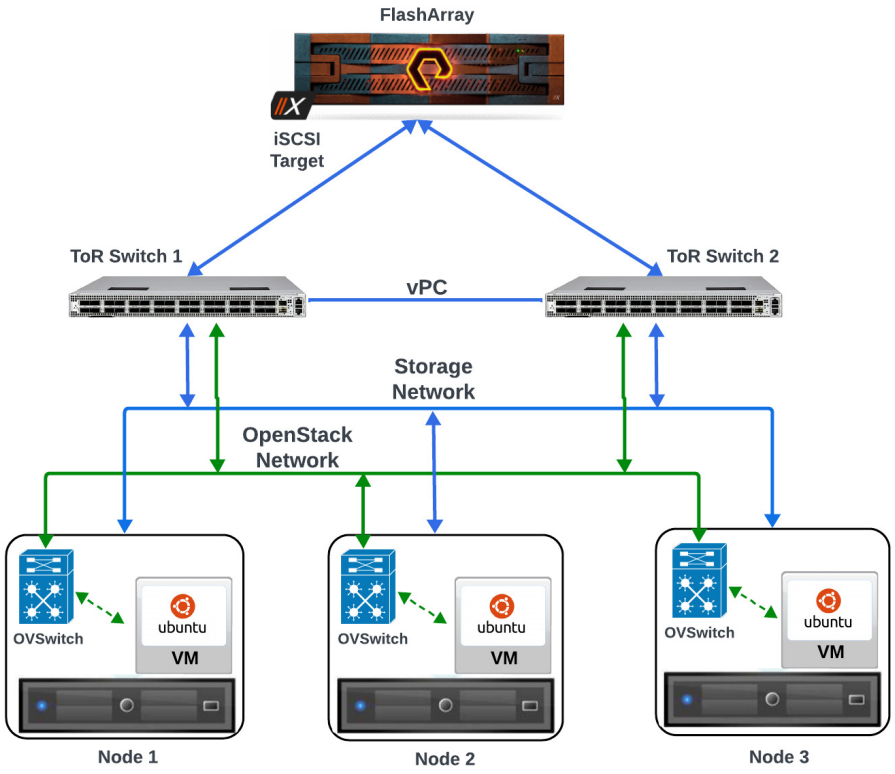


FIGURE 2 Technology component interconnects

Compute Resources

This reference architecture applies to physical multi-core servers with an operating system such as Ubuntu 22.04. At minimum three (3) servers are required, a single controller and two (2) compute nodes.

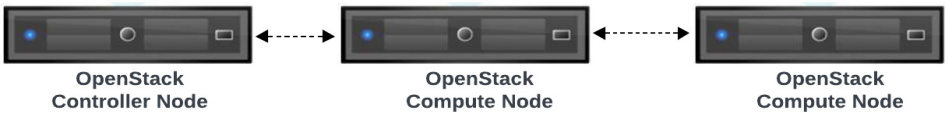


FIGURE 3 Compute resources

Network Resources

The network considerations for this reference architecture are the same as any enterprise IT infrastructure solution: availability, performance, and extensibility. The compute resources in the solution can attach to any compatible TCP/IPv4 or TCP/IPv6 network infrastructure with the general recommendation that minimum network speeds are capable of 10GbE. For this solution, the network should be configured using dual switches to eliminate a single point of failure.

Pure Storage FlashArray

Pure Storage FlashArray, a unified block and file-storage solution driven by software-defined technology, provides a seamless and reliable user experience. It incorporates data reduction capabilities without compromising performance. All Pure Storage products feature an Evergreen® subscription model, allowing for capacity and performance upgrades without requiring new storage purchases. Additionally, FlashArray empowers businesses and organizations to significantly reduce direct carbon emissions in their data storage systems, achieving up to an 80% decrease compared to competing all-flash systems and even more when compared to magnetic disks.

The FlashArray product line caters towards multiple business needs and use cases with these distinct offerings:

- **FlashArray//E™:** 80% less energy at 60% less cost for active data archives, file repositories and other use cases
- **FlashArray//C™:** An all-QLC FlashArray with consistent performance at 2-4ms latency for capacity-oriented workloads
- **FlashArray//X™:** Latency as low as 150µs to power critical applications and business operations
- **FlashArray//XL™:** Enterprise-grade performance and scalability for demanding workloads

Portworx Enterprise

Portworx is a sophisticated data management platform designed to enhance storage operations within cloud-native environments. Its container-native architecture allows for seamless integration with Kubernetes, enabling organizations to efficiently deploy and manage stateful applications. Portworx offers essential features such as high availability, data protection, and storage optimization, ensuring the resilience and performance of applications in dynamic containerized settings.

By simplifying the administration of stateful storage, Portworx offers a reliable solution for organizations facing critical data challenges. It operates as an operator within Kubernetes, automating storage cluster management and requiring specific permissions to function effectively. Portworx is versatile, supporting various Kubernetes platforms across both on-premises and major cloud environments, including AWS, Google Cloud, Azure, and Oracle.

Portworx utilizes block storage devices to create and manage distributed storage pools. These pools require one or more block storage devices connected to each node in a Kubernetes cluster, accommodating physical disks, NVMe drives, or cloud-based storage volumes. This approach ensures scalable and adaptable storage solutions tailored to meet the demanding needs of modern applications.

OpenStack

OpenStack is a free, open standard and robust open-source cloud computing platform. It is mostly deployed as infrastructure-as-a-service (IaaS) where virtual servers and other resources are made available to users. It provides an extensive array of services designed for building and managing both public and private clouds. Its modular architecture enhances flexibility and scalability, enabling organizations to tailor their cloud environments to specific needs. OpenStack supports a comprehensive suite of features, including computing, networking, storage, and identity management, facilitating the seamless deployment and orchestration of virtualized infrastructure. Additionally, its strong security and compliance features help safeguard data and ensure adherence to regulatory standards.

This platform serves as a critical infrastructure component, offering a resilient and scalable foundation for deploying and managing databases within Kubernetes environments

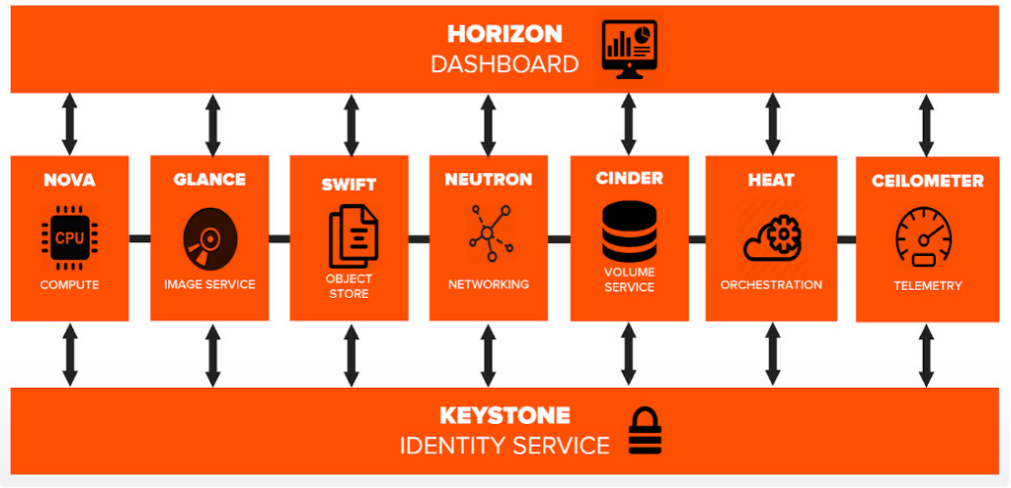


FIGURE 4 OpenStack component overview

Kubernetes

Kubernetes has become the industry standard for orchestrating containers, transforming the landscape of application deployment and management in cloud-native environments. Its advanced functionality automates the deployment, scaling, and administration of containerized workloads, empowering organizations to achieve unparalleled levels of agility and operational efficiency. Equipped with features such as service discovery, load balancing, and self-healing mechanisms, Kubernetes simplifies the complexities of deploying intricate microservices architectures.

It provides a resilient and adaptable platform for deploying and managing databases within OpenStack environments. This reference architecture leverages the capabilities of Kubernetes for container orchestration, OpenStack for underlying infrastructure management, and Portworx for storage orchestration

AlloyDB Omni

AlloyDB Omni, a downloadable edition of AlloyDB, is a versatile, PostgreSQL-compatible database designed to handle your most demanding workloads, from transactional to analytical processing. This service integrates a database engine developed by Google with a cloud-centric, multi-node architecture. This combination is engineered to deliver exceptional performance, reliability, and availability, meeting the expectations of enterprise-grade environments.

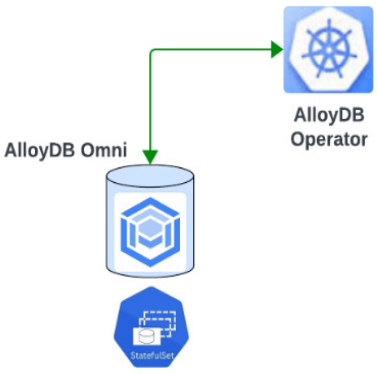


FIGURE 5 AlloyDB Omni component architecture

AlloyDB Operator

The AlloyDB operator is a Kubernetes API extension provided by Google that lets users run AlloyDB Omni in most Cloud Native Computing Foundation (CNCF)-compliant Kubernetes environments. The AlloyDB operator deploys and manages components such as persistent storage, services, secrets, etc. that are required to create a database cluster.

The operator constantly monitors the actual state of the AlloyDB cluster and ensures that it aligns with the desired state. It automates various tasks such as handling failover scenarios and applying updates or patches to the AlloyDB deployment.

AlloyDB Omni

AlloyDB for PostgreSQL is a versatile and fully managed database service that caters specifically to the requirements of an organization's most challenging workloads, encompassing a spectrum from hybrid transactional to analytical processing. Within this service, AlloyDB combines a database engine crafted by Google with a cloud-centric, multi-node architecture. This fusion is aimed at providing unparalleled levels of performance, reliability, and availability, meeting and exceeding the standards expected within enterprise-grade environments.

Technical Solution Design

AlloyDB Omni is a comprehensive database service tailored for handling both transactional and analytical workloads, significantly enhancing its utility across diverse data management scenarios. The service is seamlessly deployed within a Kubernetes-managed container environment, which leverages stateful storage capabilities provided by Portworx. This integration not only facilitates high data availability and durability but also supports dynamic scaling and maintenance of database clusters without disrupting operations, crucial for maintaining performance during varied load conditions.

The Kubernetes environment underpinning AlloyDB Omni is deployed on Nova compute instances within OpenStack nodes, all running on the Ubuntu operating system, a choice that offers robustness and security. This foundation enhances the reliability of the infrastructure and ensures a stable, consistent platform for database operations. Each OpenStack server is connected to a Pure Storage FlashArray, which hosts the Cinder volumes. This connection to high-performance storage arrays enables swift data access and storage efficiency, pivotal for both rapid transaction processing and complex query execution.

In the OpenStack configuration, four (4) virtual machines (Nova Instances) running the Ubuntu operating system are deployed. These VMs are carefully configured with optimal network and block storage settings within both the OpenStack Neutron and Cinder configuration files and the Instance operating system. This meticulous setup ensures that network traffic and storage access are optimized for performance and security, crucial for enterprise-grade database services.

Furthermore, the Kubernetes cluster and Portworx Enterprise are strategically deployed and configured within the OpenStack environment. This setup not only facilitates robust data replication and redundancy but also enhances data protection and recovery capabilities. The AlloyDB Operator and the AlloyDB Omni HA DBCluster are then deployed and configured with the necessary resources and appropriate StorageClass definitions. This orchestration allows AlloyDB Omni to leverage the full capabilities of Kubernetes and Portworx for efficient resource management and high availability, ensuring that the database service can withstand node failures and network disruptions while minimizing downtime.

Each component in this architecture—from the underlying Ubuntu operating system and OpenStack deployment to the integration of Kubernetes with Portworx—plays a pivotal role in enhancing the performance, scalability, and resilience of AlloyDB Omni. This combination ensures that the expectations for an enterprise-ready database service are met and capable of managing complex workloads effectively.

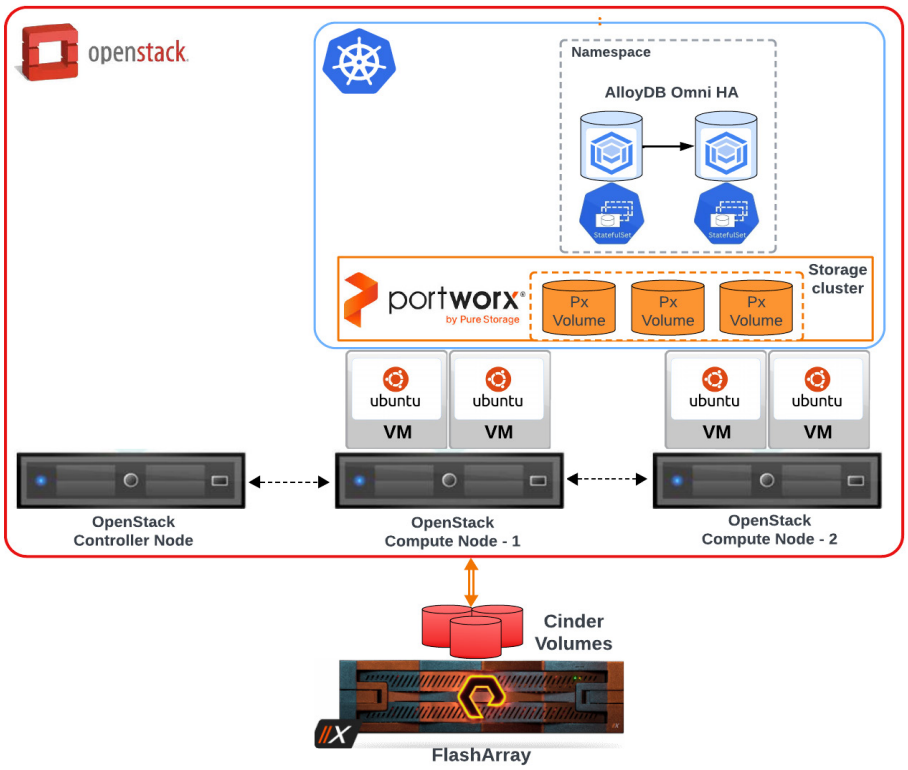


FIGURE 6 Reference architecture

Compute Layer

The OpenStack deployment must have a minimum of three (3) nodes, one (1) controller, and two (2) compute. The required servers are commodity x86-64 architectures with a recommended configuration for this reference architecture as follows:

- 24 cores
- 24GB or more memory
- 512GB local storage

Additional information about recommendations for configuration options can be found in the OpenStack documentation.

Nova Instances

OpenStack Nova, also known simply as "Nova," is the component of the OpenStack cloud computing platform that provides virtual servers upon demand. Nova is the primary computing engine behind OpenStack, responsible for creating and managing large numbers of virtual machines across a diverse set of hardware resources. This reference architecture made use of four (4) Nova instances with the following recommended configuration using a custom Nova flavor:

- 16 virtual CPUs
- 16GB or more memory
- Nova ephemeral serves as the boot volumes, each sized at 100GB
- 2 x cinder volumes
- 200GB for KVDB
- 1TB for Kubernetes worker node storage

Storage Layer

FlashArray block storage volumes were provisioned and connected to the OpenStack hosts using the iSCSI storage connection protocol with 25Gb/s connectivity. Two (2) Cinder volumes were provisioned per host.

Cinder volumes created on the FlashArray are presented to the Nova compute nodes and passed through to the Nova instances where Portworx uses these devices to provision its storage cluster. Cinder volumes are used as follows:

- 200GB volume dedicated to KVDB for Portworx
- 1TB volume for Kubernetes worker node storage

The guide from Portworx on tuning performance can be used to understand design time optimization parameters.

Storage is provisioned in this reference architecture with the following component inheritance structure.

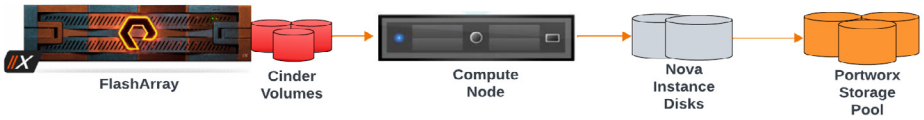


FIGURE 7 Storage component inheritance

AlloyDB Omni Design Considerations

Designing an architecture for AlloyDB Omni in a Kubernetes on OpenStack environment with a focus on using FlashArray block storage and Portworx stateful container storage involves several considerations. These considerations ensure high performance, scalability, resilience, and efficient data management to support the needs of the deployment (Figure 9).

Sizing

Storage sizing includes both performance and capacity considerations. For capacity, when deploying AlloyDB Omni it is recommended that the volume sizes (starting at the block storage layer) are provisioned at 150% of requirement and include data growth forecasting. Performance needs require a deeper understanding of the exact workload being performed, however, if the workload can be characterized in terms of IOPS and latency then the outcome would focus on tuning the various storage components.

Availability

Availability is a design time consideration for enterprises, where uninterrupted access to essential data resources may need to be ensured. With Kubernetes deployments, the infrastructure is abstracted to eliminate dependencies and enhance resilience against potential failures of individual components, such as servers, networks, or storage arrays.

There are two availability topologies relevant to this reference architecture: storage-based high availability and transparent database availability. The combination of these two methods guarantees that data remains highly available to applications and users continuously.

Storage-based High Availability

Portworx storage-based high availability offers significant advantages for AlloyDB Omni workloads in Kubernetes settings. By duplicating data across multiple nodes using replication, Portworx maintains the accessibility of storage volumes even when nodes fail or face disruptions. This built-in redundancy facilitates seamless failover by automatically redirecting data access to operational nodes, thereby reducing downtime, and preserving continuous operations for AlloyDB Omni workloads. This is shown in Figure 9.

Portworx eliminates a single point of failure for single-node AlloyDB Omni deployments in Kubernetes.

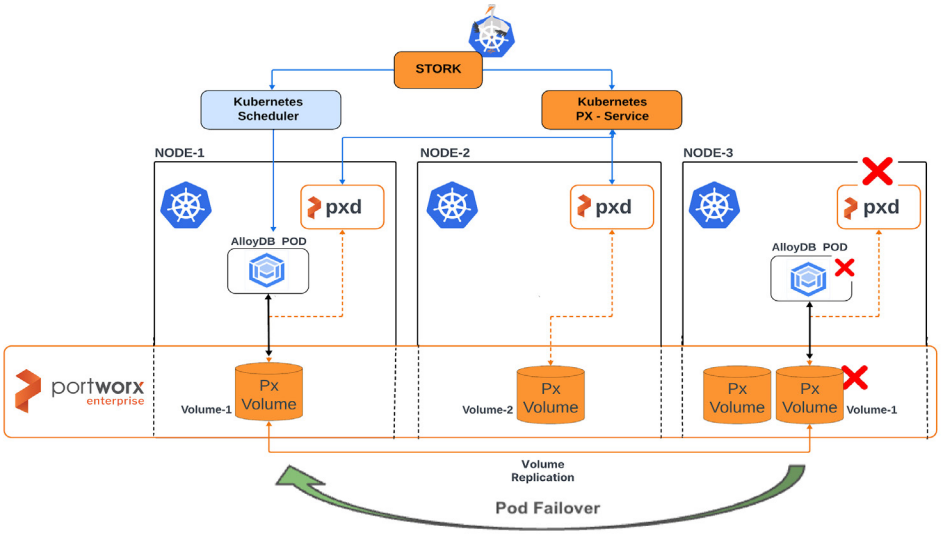


FIGURE 8 AlloyDB Omni failover using storage-based high availability.

Transparent Database High Availability

Transparent database high availability solutions deliver multi-layered availability that removes single points of failure by enhancing resilience at the database layer, independent of storage availability. This system proactively identifies failures at the database level and shifts operations to standby systems, thus preserving data integrity and preventing downtime. This method boosts system reliability and resilience, ensuring continuous access to vital data and services for users.

Database-level replication, which involves configuring one or more standby databases to synchronize with a primary instance through the shipment of transaction logs (Write-Ahead Logging), is one method to establish HA at the database level. For AlloyDB Omni workloads, the AlloyDB Omni operator handles the deployment and monitoring of the high availability setup.

The Google AlloyDB Omni documentation provides further details on how the AlloyDB operator facilitates high availability.

For additional information on how the AlloyDB operator manages high availability, refer to Google AlloyDB Omni documentation.

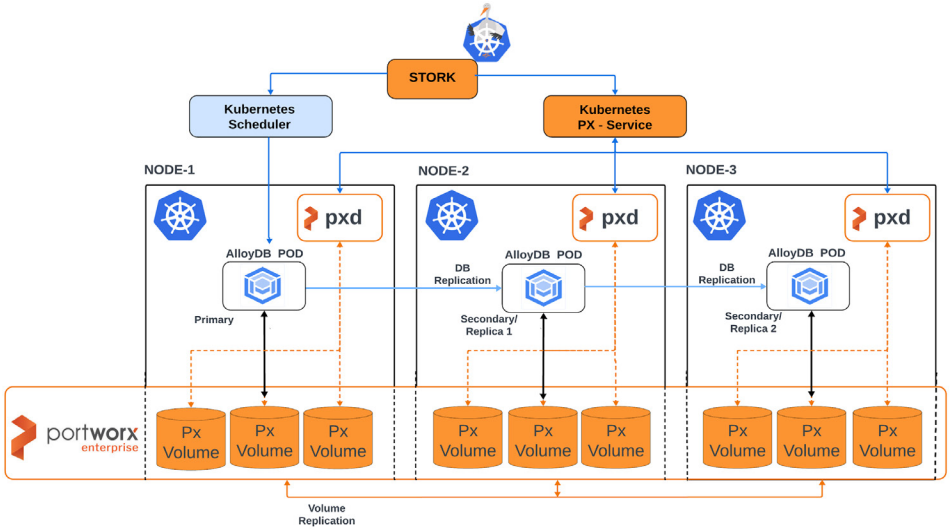


FIGURE 9 AlloyDB Omni transparent availability setup combined with storage availability.

Design Validation

The design validation provides evidence of the solution's integrity and demonstrates the benefits through a series of test scenarios using HammerDB.

NOTE: Caution: The results achieved in this reference architecture are meant to only be used as an indicator of what could be expected when deploying AlloyDB Omni in the same manner. These results are specific to the environment and circumstances used to create this reference architecture, different environments and circumstances will yield different results.
This test environment also used 25GB networking with a FlashArray//X90R3 and 128 Core AMD servers, each with 512GB memory.

The TPCC benchmark tests for PostgreSQL, conducted using HammerDB, provide valuable insights into the database system's performance under transactional workloads. These tests simulate a real-world online transaction processing (OLTP) environment to measure the database's ability to handle concurrent transactions, maintain data consistency, and scale effectively. HammerDB enables users to simulate various TPCC scenarios, including 30% cache and 100% cache tests, allowing for a comprehensive performance evaluation.

- The 30% cache test within the HammerDB OLTP benchmark for PostgreSQL assesses the performance of the database system when 30% of the accessed data is stored in cache memory. This test is crucial for evaluating how efficiently the database processes queries and transactions with a significant portion of data readily available in memory.
- The 100% cache test in the HammerDB OLTP benchmark for PostgreSQL measures the database system's performance when all accessed data is stored in cache memory. This scenario provides insights into the maximum potential performance of the database when all necessary data is immediately accessible in memory, facilitating an understanding of its optimal operational capabilities.

Configuration

The AlloyDB Omni single-node cluster was deployed with the following configuration:

- 12 CPU
- 100GB Memory
- 800GB Disk Size with the following Portworx optimizations
  - repl: "3"
  - io_priority: "high"
  - fs: "xfs"
  - io_profile: "db_remote"

Follow the below instructions to deploy the validation steps:

1. Deploy and configure HammerDB.
2. Configure AlloyDB Omni.
   - pg_hba.conf to enable/allow login for benchmark tool users
   - Modify/add the below parameters in postgresql.conf using the kubectl patch command

```
work_mem: 12MB
checkpoint_timeout: 300
max_connections: 1000
max_parallel_maintenance_workers: 12
max_parallel_workers: 12
max_wal_size: 5GB
max_parallel_workers_per_gather: 6
min_wal_size: 1GB
wal_buffers: 1GB
huge_pages: off
shared_buffers: 80GB
```

The following is an example of updating parameters in the postgresql.conf file using kubectl:

```
kubectl patch dbclusters.alloydbomni.dbadmin.goog alloydb -p '{"spec":{"primarySpec":{"pa-
rameters": {"work_mem": "12MB", "checkpoint_timeout": "300", "max_connections": "1000",
"max_parallel_maintenance_workers": "12", "max_parallel_workers": "12", "max_wal_
size":"5GB", "max_parallel_workers_per_gather": "6","min_wal_size":"1GB", "wal_buf-
fers":"1GB", "huge_pages":"off", "shared_buffers":"80GB"}}}}' --type=merge -n alloy
```

3. Reload the PostgreSQL configuration.

```
pg_ctl reload -D <data_dir_path>
```

4. Load Data in AlloyDB Omni using HammerDB.
5. Run TPCC Benchmark tests (30% cache & 100% cache tests).

The output of HammerDB TPCC Benchmark test results provides two sets of metrics:

1. **New orders per minute (NOPM):** Measures the rate at which new orders are created in a simulated online transaction processing (OLTP) environment.
2. **Transactions per minute (TPM):** Metric to measure the throughput or processing capacity of a database system in terms of the number of transactions completed within a minute.

In the validation process, performance metrics were achieved with transaction rates reaching 158,275 transactions per minute for the 30% cache test and an impressive 579,423 transactions per minute for the 100% cache test. These results underscore the efficiency and scalability of the tested system configuration, demonstrating its ability to handle substantial workloads with ease. Such transaction rates are indicative of the system's robustness and its capacity to support demanding enterprise applications effectively.

Results

Figure 11 shows the results of the TPCC benchmark tests of AlloyDB for PostgreSQL using the HammerDB application, aimed to validate the design using the following configuration.

In the validation process, the performance of the database system under different caching scenarios was quantified. For the 30% cache test, where only 30% of the accessed data was stored in cache memory, the system achieved transaction rates of 158,275 transactions per minute. This performance level indicates that the system can handle a considerable volume of transactions efficiently, even when a significant portion of the data requires retrieval from slower storage media.

The results were even more impressive in the 100% cache test, where all accessed data was stored in cache memory. Here, the system reached an astounding 579,423 transactions per minute. This high transaction rate demonstrates the system's maximum potential performance, showcasing its ability to deliver swift response times and handle peak workloads with ease. It reflects the system's robustness and its capacity to support demanding enterprise applications effectively.

Overall, these results underscore the efficiency and scalability of the design, demonstrating its ability to manage substantial workloads with relative ease. Such transaction rates are indicative of the system's robustness and its strong capability to support a range of enterprise applications under various operational scenarios.

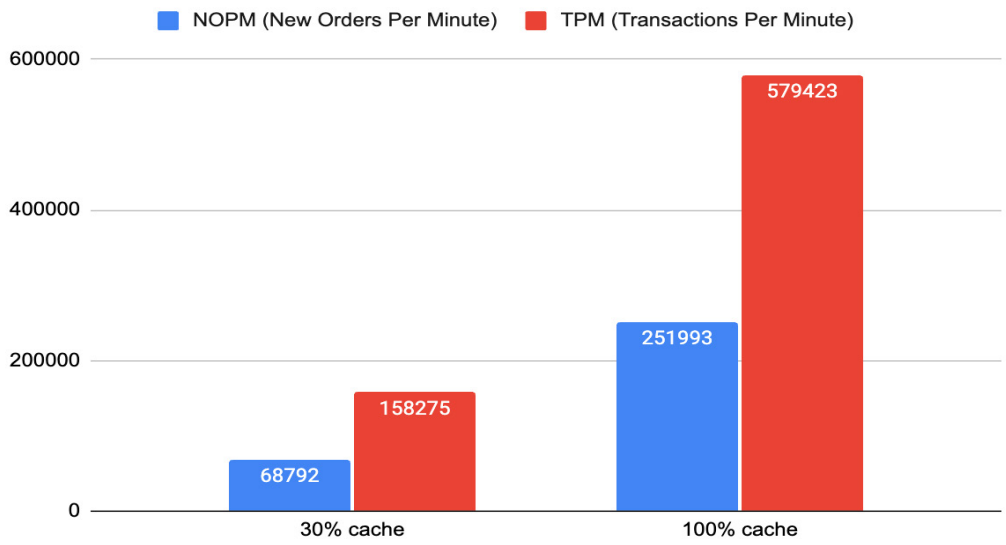


FIGURE 10 1 NOPM and TPM as expected metrics for this design

Deployment

This section provides detailed insights for implementing a deployment of AlloyDB Omni on Kubernetes in OpenStack with Portworx and FlashArray storage components. It provides guidance about deployment requirements, prerequisites, Installation and configurations for key components like OpenStack, Kubernetes, Portworx Enterprise, and AlloyDB Omni, ensuring the implementation of an efficient on-premises solution.

Deployment Requirements

The following table lists the component version requirements:

Component	Version
Storage	FlashArray with Purity//FA 6.0 or higher
OpenStack	2023.1 or higher
Operating System (OpenStack and Nova instances)	Ubuntu 22.04.4 LTS
Kubernetes	v1.23.0 or higher
Portworx	3.0.0 or higher
AlloyDB Operator	0.4.0 or higher
AlloyDB Omni	15.5.0 or higher

OpenStack

Setting up OpenStack involves executing several steps to build a dependable cloud infrastructure. Begin by installing the required OpenStack components, as outlined in the DevStack documentation, which provides prerequisites and detailed instructions for installing and configuring a multi-node OpenStack environment.

To keep things simple, we've opted for the DevStack deployment of OpenStack, although alternative deployment methods are also accessible.

This section outlines the procedures required to deploy the OpenStack environment (also providing the OpenStack Horizon dashboard), and storage setup. Storage multipathing and udev rules (which enhance the reliability, performance, and manageability of storage configurations in OpenStack environments) contribute to a robust and efficient storage infrastructure. Additionally, details are provided for the prerequisites needed to configure the Cinder service and highlight key configurations essential for Nova instances to ensure a successful deployment.

Storage

This section covers details of multipath and udev rules configuration for the storage layer and are prerequisite to configuring the Cinder service that provides block storage volumes to Nova instances, enabling persistent storage for data and applications.

1. Add the below configuration in /etc/multipath.conf file.

```
defaults {
 polling_interval 10
 user_friendly_names no
}
devices {
 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
 }
}
```

2. To refresh the multipath configuration, execute the following command:

```
systemctl restart multipathd
```

3. To create udev rules, Add the following configuration files in /etc/udev/rules.d/ path.

- 90-scsi-ua.rules

```
# Add SCSI Unit Attention rescan for resize
ACTION=="change", SUBSYSTEM=="scsi", ENV{SDEV_UA}=="CAPACITY_DATA_HAS_CHANGED", TEST=="rescan", ATTR{res-
can}="x"
```

- 99-pure-storage.rules

```
# Recommended settings for Pure Storage FlashArray.
# Use noop 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/schedul-
er}="none"
ACTION=="add|change", KERNEL=="dm-[0-9]*", SUBSYSTEM=="block", ENV{DM_NAME}=="3624a937*", ATTR{queue/sched-
uler}="none"
# Reduce CPU overhead due to entropy collection
ACTION=="add|change", KERNEL=="sd*[!0-9]", SUBSYSTEM=="block", ENV{ID_VENDOR}=="PURE", ATTR{queue/add_ran-
dom}="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_affini-
ty}="2"
ACTION=="add|change", KERNEL=="dm-[0-9]*", SUBSYSTEM=="block", ENV{DM_NAME}=="3624a937*", ATTR{queue/rq_af-
finity}="2"
# Set the HBA timeout to 60 seconds
ACTION=="add|change", KERNEL=="sd*[!0-9]", SUBSYSTEM=="block", ENV{ID_VENDOR}=="PURE", ATTR{device/time-
out}="60"
```

4. To refresh udev rules, execute the following commands.

```
udevadm control --reload-rules
udevadm trigger --type=devices --action=change
```

For more details about multipath and udev rules configuration, refer following Pure Storage best practice guides.

- Multipathing
- udev rule for SCSI Unit Attention
- udev rule for Queue Settings

DevStack Setup

This section covers the prerequisite configuration to configure the OpenStack environment on bare-metal servers.

1. Prepare and configure the "local.conf" files on the controller and compute nodes using the format provided below as a reference example.

On the controller node:

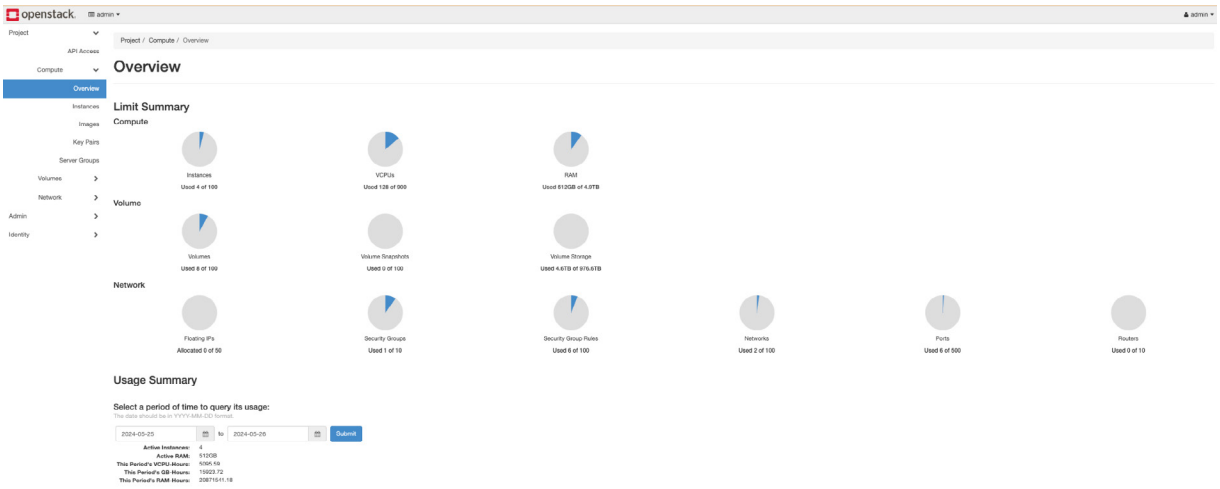
```
[[local|localrc]]
ADMIN_PASSWORD=<redacted>
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
CINDER_BRANCH=stable/2023.1
GLANCE_BRANCH=stable/2023.1
HORIZON_BRANCH=stable/2023.1
KEYSTONE_BRANCH=stable/2023.1
KEYSTONECLIENT_BRANCH=stable/2023.1
NOVA_BRANCH=stable/2023.1
NOVACLIENT_BRANCH=stable/2023.1
NEUTRON_BRANCH=stable/2023.1
SWIFT_BRANCH=stable/2023.1
HOST_IP=<Controller_Node_IP>
PUBLIC_INTERFACE=<Interface_Name>
LOGFILE=/opt/stack/logs/stack.sh.log
# Neutron options
Q_USE_SECGROUP=True
FLOATING_RANGE="10.1.0.0/24"
IPV4_ADDRS_SAFE_TO_USE="10.0.0.0/22"
Q_FLOATING_ALLOCATION_POOL=start=10.1.0.76,end=10.1.0.83
PUBLIC_NETWORK_GATEWAY="10.1.0.1"
PUBLIC_INTERFACE=eth0
# Open vSwitch provider networking configuration
Q_USE_PROVIDERNET_FOR_PUBLIC=True
OVS_PHYSICAL_BRIDGE=br-ex
PUBLIC_BRIDGE=br-ex
OVS_BRIDGE_MAPPINGS=public:br-ex
[[post-config|$CINDER_CONF]]
[DEFAULT]
verbose=true
debug=true
enabled_backends = puredriver-1
default_volume_type = puredriver-1
....
[puredriver-1]
volume_backend_name = puredriver-1
volume_driver = cinder.volume.drivers.pure.PureISCSIDriver
san_ip = <FLASHARRAY_DATA_IP>
pure_api_token = PURE_API_TOKEN
use_multipath_for_image_xfer = True
```

On the compute node:

```
[[local|localrc]]
HOST_IP=<Compute_Node_IP> # change this per compute node
FLOATING_RANGE=<0.0.0.0/24>
LOGFILE=/opt/stack/logs/stack.sh.log
ADMIN_PASSWORD=<redacted>
DATABASE_PASSWORD=<redacted>
RABBIT_PASSWORD=<redacted>
SERVICE_PASSWORD=<redacted>
DATABASE_TYPE=mysql
SERVICE_HOST=<Controller_Node_IP>
MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
ENABLED_SERVICES=n-cpu,placement-client,ovn-controller,ovs-vswitchd,
 ovsdb-server,q-ovn-metadata-agent
NOVA_VNC_ENABLED=True
NOVNCPROXY_URL="http://$SERVICE_HOST:6080/vnc_lite.html"
VNCSERVER_LISTEN=$HOST_IP
VNCSERVER_PROXYCLIENT_ADDRESS=$VNCSERVER_LISTEN
PUBLIC_INTERFACE=<Interface_Name>
CINDER_BRANCH=stable/2023.1
GLANCE_BRANCH=stable/2023.1
HORIZON_BRANCH=stable/2023.1
KEYSTONE_BRANCH=stable/2023.1
KEYSTONECLIENT_BRANCH=stable/2023.1
NOVA_BRANCH=stable/2023.1
NOVACLIENT_BRANCH=stable/2023.1
NEUTRON_BRANCH=stable/2023.1
SWIFT_BRANCH=stable/2023.1
[[post-config|$NOVA_CONF]]
[libvirt]
volume_use_multipath=True
```

2. Run "stack.sh" as the "stack" user on the controller and compute nodes. Run the controller node deployment before running on the compute nodes.
3. Upon successful completion of the OpenStack environment setup process, the script output from both the controller and compute nodes will include the URL for accessing the Horizon dashboard.

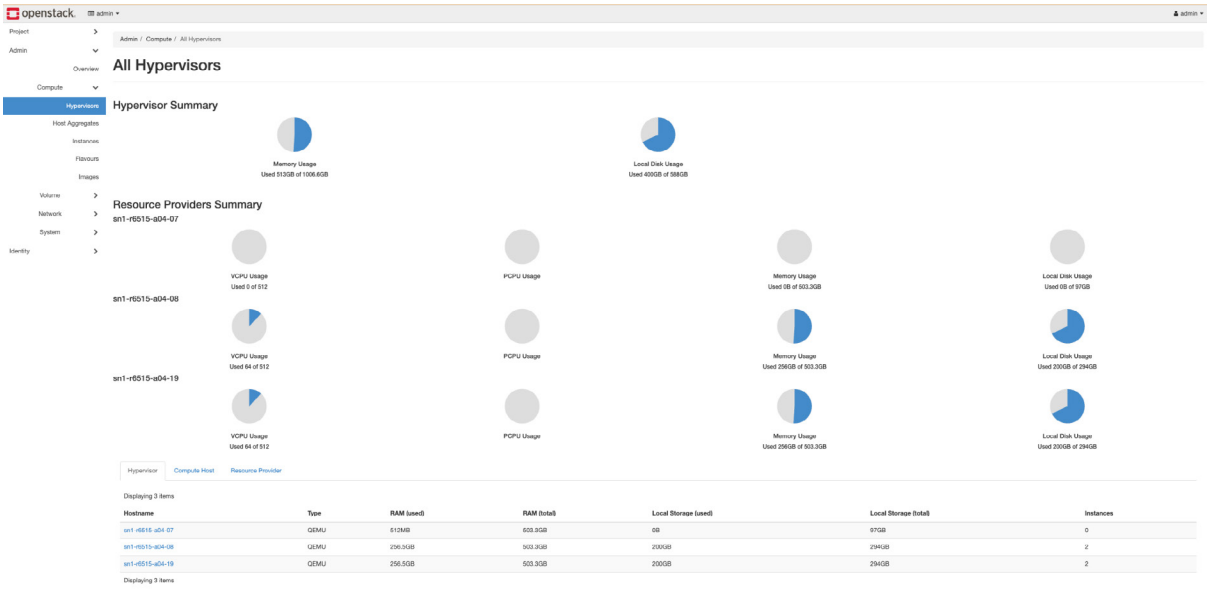
Here is a screenshot depicting the appearance of the Horizon Dashboard.



4. OpenStack deployments do not automatically include compute nodes in the infrastructure. Instead, the following nova-manage command will discover compute hosts.

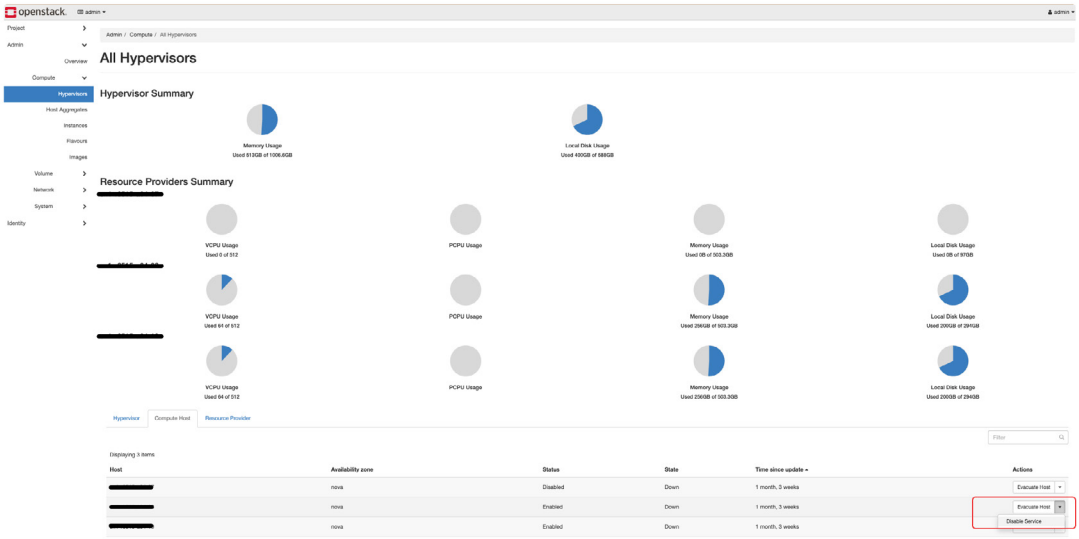
```
nova-manage cell_v2 discover_hosts
```

5. To validate setup, compute and controller nodes should be listed in the “Hypervisor” tab under the “Admin/Compute/All Hypervisors” page.



6. Disable the “Compute Host” service on the controller node to ensure controller resources are not assigned/allocated to instances deployed in the OpenStack environment.

To disable the “Compute” service on controller node, open the Horizon dashboard and navigate to “Admin→Compute→All Hypervisors” page, choose the "Compute Host" tab, and within the "Actions" dropdown menu, select "Disable Service" corresponding to the controller node.



Nova

Nova is a core component of OpenStack, and acts as the cloud compute orchestration service. It's responsible for the lifecycle management of virtual machines (VMs) within your OpenStack cloud. Nova enables the creation of virtual machines and bare metal servers (via ironic) while also offering partial support for system containers. Nova runs as a set of daemons on top of Ubuntu servers to provide that service.

This section covers the key configuration of Nova instances along with steps to create Nova instances.

Nova's configuration file, typically named nova.conf, resides on each OpenStack compute node. It defines various settings for Nova, including:

- API service configuration (port, rate limiting)
- Database connection details
- Resource allocation limits for virtual machines (CPU, memory)
- Image and instance naming conventions

Note: OpenStack inherits CPU properties. However, if it fails to inherit CPU properties like Advanced Encryption Standard (AES) compatibility issues may arise during AlloyDB Omni installation. This occurs because AlloyDB Omni binary is compiled with AES enabled, potentially leading to installation failures

Edit and add the following properties in the “libvirt” section of nova.conf and nova-cpu.conf configuration files in compute nodes.

- Add the “cpu_model_extra_flags” property and set it to “aes”.

The following is an example of the “libvirt” section for reference.

```
[libvirt]
cpu_model = <cpu_model>
cpu_mode = custom
cpu_model_extra_flags = aes
```

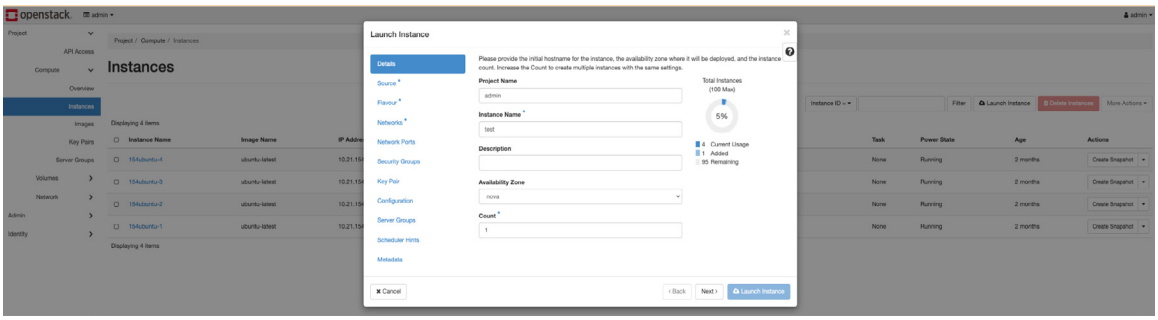
Nova offers the flexibility to configure features of the virtual CPU exposed to instances. This combined set of CPU features is termed the CPU model. These features are implemented using the libvirt driver. Refer to the Nova documentation for more details on CPU models.

Nova Instance Creation

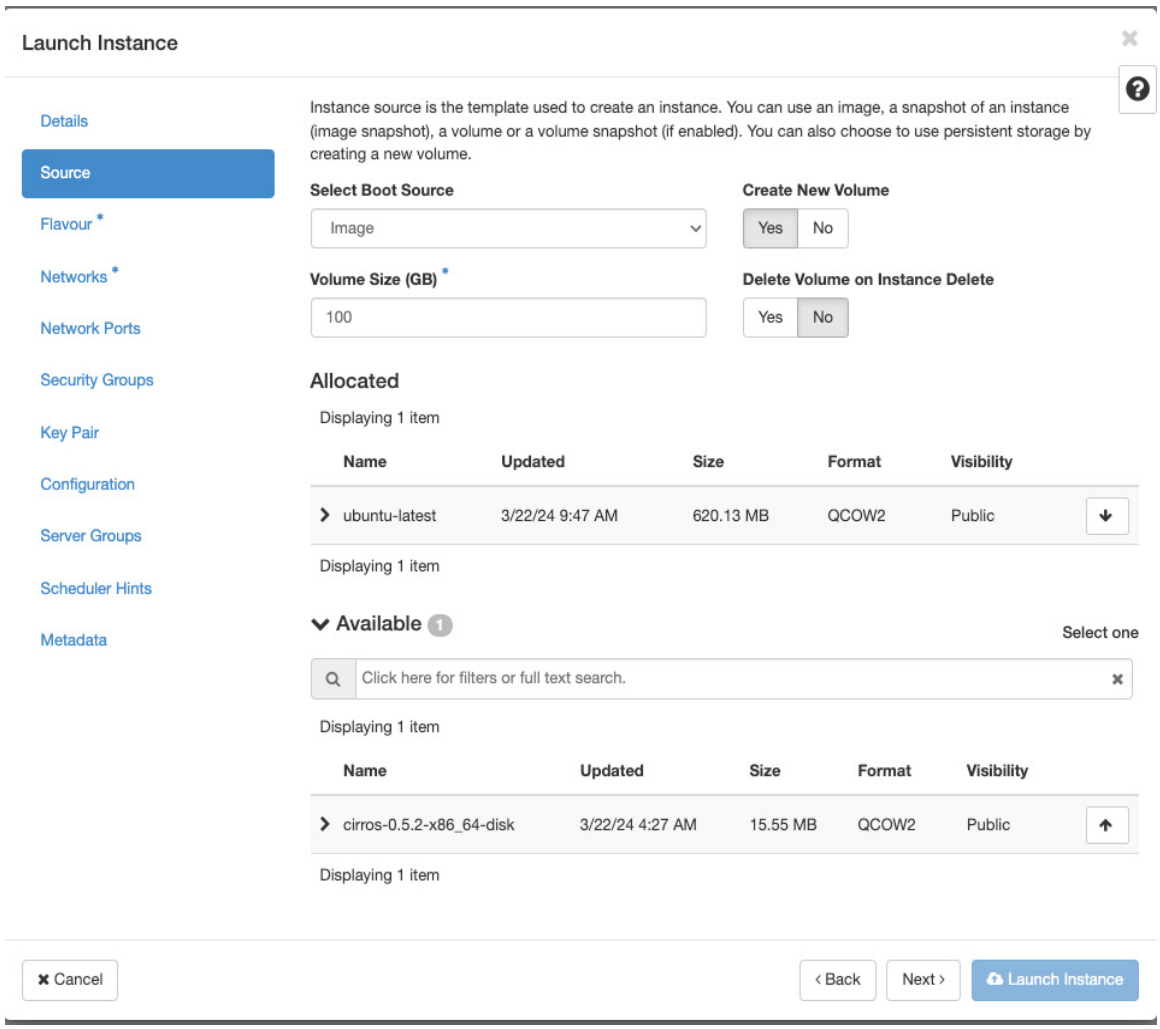
Nova instances can be created using the OpenStack Horizon dashboard.

1. Navigate to the “Project→Compute→Instances” page, click on the “Launch Instance” button. This opens “Launch Instance” page.

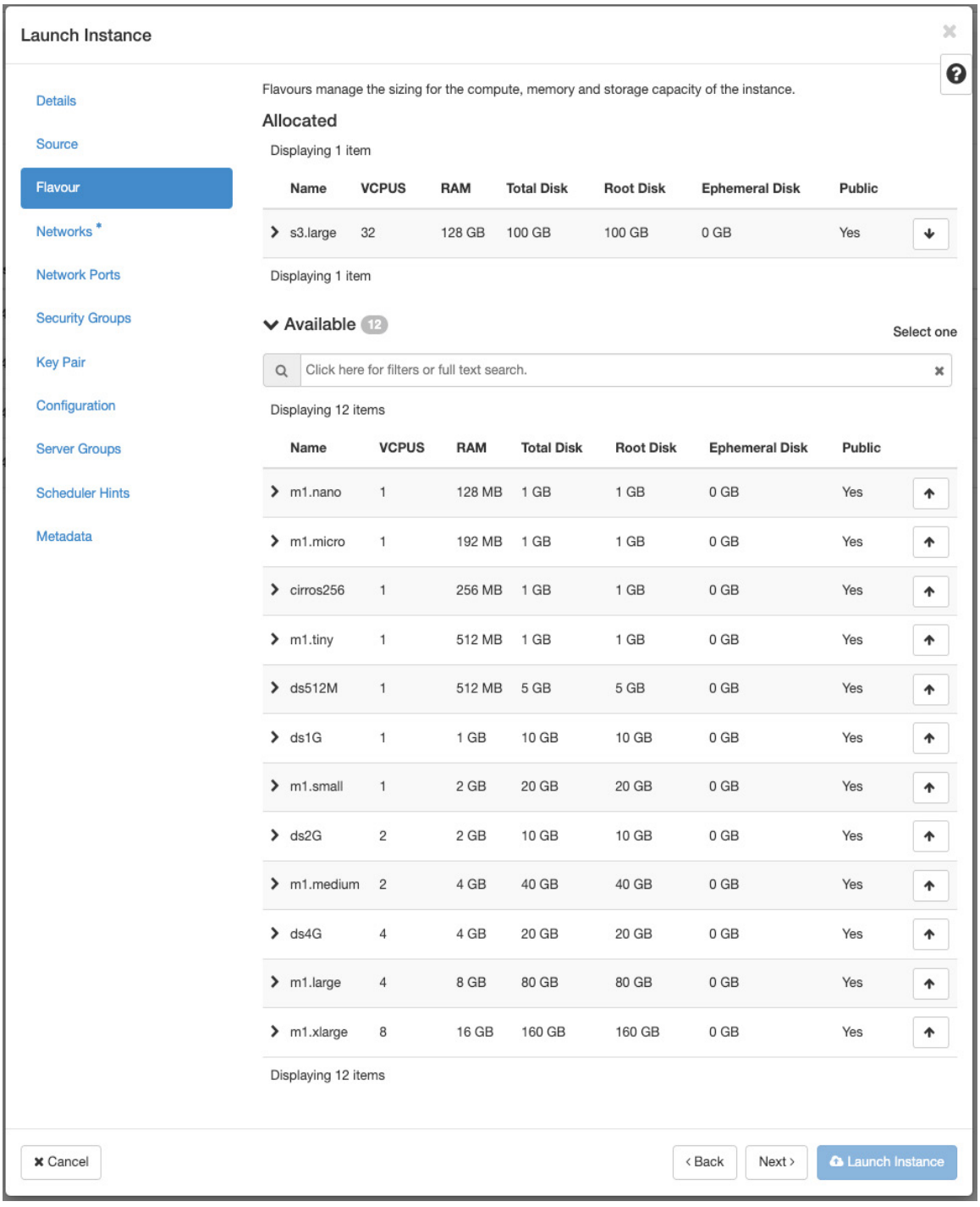
Fill out the “Instance Name” field and provide the number of instances to be provisioned in the “Count” field and click on “Next”.



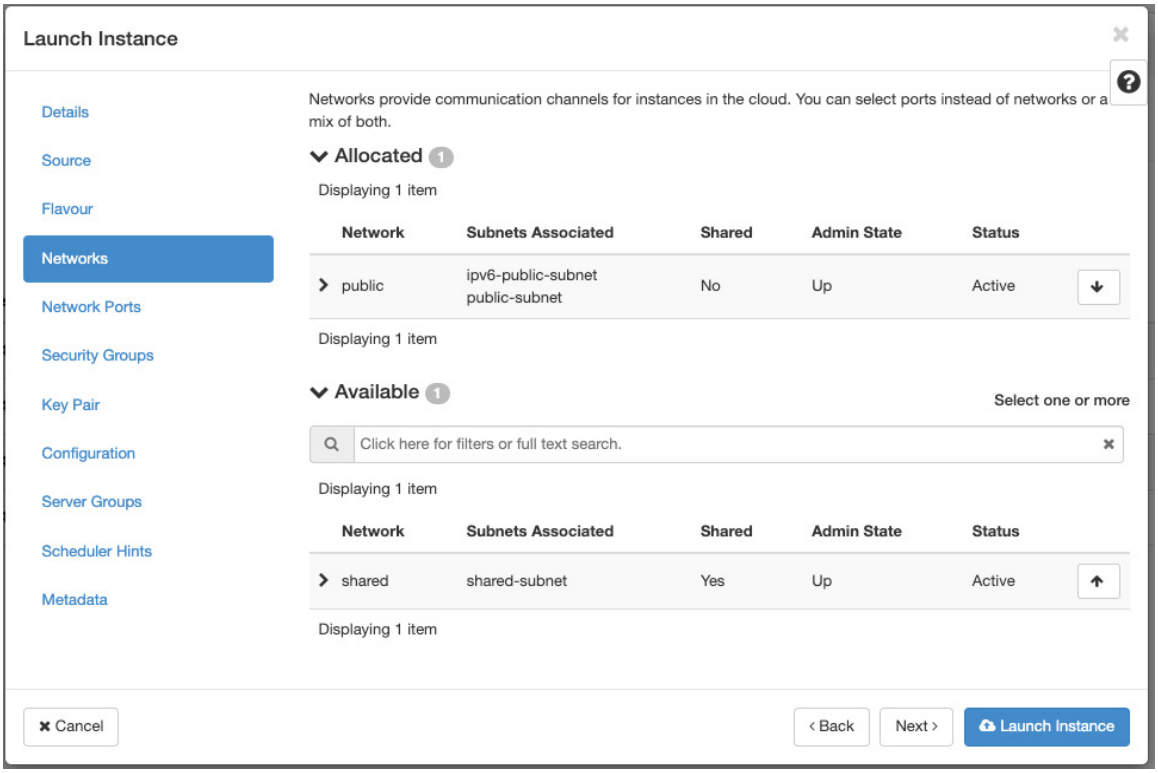
2. Select appropriate values in “Boot Source” and “Volume Size” and select an image from the “Available” list. Click on “Next” to choose the required Flavour.



3. In the Flavour page, compute, memory and disk related settings can be configured by selecting required flavour from already existing flavours. (Additionally, users have the option to create a new customized flavor if needed.)


Launch Instance
Details
Source
Flavour
Networks *
Network Ports
Security Groups
Key Pair
Configuration
Server Groups
Scheduler Hints
Metadata
Flavours manage the sizing for the compute, memory and storage capacity of the instance.
Allocated
Displaying 1 item
Name VCPUS RAM Total Disk Root Disk Ephemeral Disk Public
s3.large 32 128 GB 100 GB 100 GB 0 GB Yes
Displaying 1 item
Available 12
Select one
Click here for filters or full text search.
Displaying 12 items
Name VCPUS RAM Total Disk Root Disk Ephemeral Disk Public
m1.nano 1 128 MB 1 GB 1 GB 0 GB Yes
m1.micro 1 192 MB 1 GB 1 GB 0 GB Yes
cirros256 1 256 MB 1 GB 1 GB 0 GB Yes
m1.tiny 1 512 MB 1 GB 1 GB 0 GB Yes
ds512M 1 512 MB 5 GB 5 GB 0 GB Yes
ds1G 1 1 GB 10 GB 10 GB 0 GB Yes
m1.small 1 2 GB 20 GB 20 GB 0 GB Yes
ds2G 2 2 GB 10 GB 10 GB 0 GB Yes
m1.medium 2 4 GB 40 GB 40 GB 0 GB Yes
ds4G 4 4 GB 20 GB 20 GB 0 GB Yes
m1.large 4 8 GB 80 GB 80 GB 0 GB Yes
m1.xlarge 8 16 GB 160 GB 160 GB 0 GB Yes
Displaying 12 items
Cancel
Back
Next
Launch Instance

4. Next, select appropriate networks to be allocated from the existing networks list and click the “Launch Instance” button.



The above steps deploys a Nova Instance in the OpenStack environment.

Cinder

Cinder is the block storage service within the OpenStack suite of services. It allows users to manage volumes, which are block storage devices that can be attached to instances running in the OpenStack compute service (Nova).

iSCSI facilitated through Cinder block storage service is a widely used storage protocol within OpenStack and allows seamless integration with storage arrays that present iSCSI targets. It provides flexible and scalable storage solutions for cloud deployments.

The Pure Storage FlashArray Cinder iSCSI driver enhances connectivity between OpenStack deployments and iSCSI targets offered by Pure Storage FlashArray. This driver serves as a bridge between OpenStack Cinder services and the FlashArray storage infrastructure, facilitating efficient provisioning and management of iSCSI volumes within the OpenStack environment. Through this integration, users can leverage the advanced features and performance capabilities of Pure Storage FlashArray while benefiting from the centralized management and automation capabilities of OpenStack.

This section provides an overview of the prerequisites to configure the Pure Storage FlashArray Cinder iSCSI driver.

1. Create volume types to associate specific performance, resilience, and other settings as key-value pairs.

Execute the following commands to create a volume type and set it up as the volume backend.

```
cinder type-create pure1
cinder type-key pure1 set volume_backend_name=puredriver-1
```

2. Create volumes using Horizon Dashboard or CLI command and attach it to Nova Instances.

For best practices on OpenStack Cinder configuration on FlashArray for the deployed version of OpenStack, see the Pure Storage documentation.

Kubernetes

Leveraging Nova instances as the underlying infrastructure provides control over compute resources while ensuring compatibility with OpenStack's management and networking capabilities. Deploying a Kubernetes cluster on Nova instances within an OpenStack environment offers a flexible and scalable approach to container orchestration.

Kubernetes requires careful setup of master and worker nodes, along with networking configurations like DNS and CNI plugins. Deploying Kubernetes in an OpenStack environment involves several steps to ensure a smooth integration.

- Ensure the OpenStack environment is running with sufficient resources to deploy the Kubernetes cluster.
- Select a Kubernetes deployment tool that is compatible with OpenStack. Kubespray is most commonly used.
- Ensure the latest version of Ansible is installed.

Here are the steps to set up the Kubernetes cluster using Kubespray.

1. Clone the git repository.
2. Copy ``inventory/sample`` as ``inventory/mycluster`` using this command:

```
cp -rfp inventory/sample inventory/mycluster
```

3. Update the Ansible inventory file with the inventory builder:

```
declare -a IPS=(<node1_IP>, <node2_IP>, <node3_IP>)
CONFIG_FILE=inventory/mycluster/hosts.yaml python3 contrib/inventory_builder/inventory.py
${IPS[@]}
```

4. Review and change parameters under ``inventory/mycluster/group_vars``.

```
cat inventory/mycluster/group_vars/all/all.yml
cat inventory/mycluster/group_vars/k8s_cluster/k8s-cluster.yml #specify kubernetes version
```

5. Deploy Kubespray with Ansible Playbook and run the playbook as root. The option `--become` is required, as for example writing SSL keys in /etc/, installing packages and interacting with various systemd daemons. (Note that without "--become" the playbook will fail to run!)

```
ansible-playbook -i inventory/mycluster/hosts.yaml --become --become-user=root cluster.yml
```

Refer Kubespray documentation for prerequisites and installation instructions.

Portworx Enterprise

Portworx Enterprise can be installed and configured using the Portworx Operator on Kubernetes. It leverages the Kubernetes Operator framework for simplified lifecycle management.

Portworx seamlessly integrates with OpenStack environments to provide efficient, scalable, highly available storage solutions to any AlloyDB Omni statefulsets running on Kubernetes. One of its key features is the ability to create a storage cluster pool using Cinder volumes attached to Nova instances. This process involves several steps.

- Portworx discovers available Cinder volumes attached to Nova instances, imports these volumes into its storage cluster, and identifies them as viable storage resources.
- Portworx creates a unified storage resource by combining these volumes that will be ready to serve the cluster's needs.
- The Portworx storage cluster will be initialized to prepare volumes for data storage and ready to handle application workloads effectively.
- Portworx applies data replication, protection, and observability mechanisms to the storage pool to ensure high availability, data resilience, operational status, and proactive management.
- Portworx dynamically allocates storage resources to statefulsets from the storage pool to meet the requirements of Kubernetes workloads running on Nova instances within the OpenStack environment by provisioning PersistentVolumes (PVs) and/or PersistentVolumeClaims (PVCs).

Here are the requirements to install Portworx Enterprise.

- A valid account is needed to log in and access Portworx Central (product catalog) to generate manifest/spec.
- A secret in the target cluster with credentials (encoded to base64 format) to access the infrastructure resources to deploy the storage cluster needs to be created.

This process deploys the Portworx Operator and creates StorageCluster in Kubernetes.

1. Installation prerequisites include system, network, software requirements and supported Kubernetes versions for Portworx enterprise can be found in Portworx Documentation. Portworx enterprise setup involves two steps, spec generation and applying specs.

Portworx's deployment manifest (spec generation) can be personalized and obtained from PX-Central, a web-based interface. Through PX-Central, users can select from the Portworx product catalog and configure platform-specific parameters to align with their requirements.

Applying the specs involves two steps: Portworx operator deployment and StorageCluster Deployment.

a. To deploy a Portworx operator, execute the below command by replacing the appropriate Portworx version.

```
kubectl apply -f 'https://install.portworx.com/<version-number>?comp=pxoperator'
```

b. To deploy the StorageCluster, execute the below command by replacing the appropriate Portworx version.

```
kubectl apply -f 'https://install.portworx.com/<version-number>?operator=true&mc=-
false&kbver=&b=true&c=px-cluster-0d8dad46-f9fd-4945-b4ac-8dfd338e915b&stork=true&c-
si=true&mon=true&tel=false&st=k8s&promop=true'
```

2. Detailed setup and validation instructions for Portworx enterprise are available in the Portworx Setup Documentation.
3. Define a Kubernetes StorageClass that utilizes Portworx as the storage provider. This StorageClass will be used to provision PVs for AlloyDB pods.

Optimal configurations for Portworx can be implemented during the creation of a StorageClass in Kubernetes.

StorageClass

A StorageClass is a provisioning system within Kubernetes that enables the dynamic allocation of persistent volumes (PVs) in a Kubernetes cluster. Kubernetes administrators define different storage classes, and pods can then request the precise storage type required on the fly.

Portworx recommended parameters to configure within the StorageClass definition for deploying database workloads:

- **allowVolumeExpansion**: Setting to true enables dynamic/online resizing of volumes supported by the StorageClass.
- **provisioner**: Setting to kubernetes.io/portworx-volume.Portworx provisioner is the volume plugin utilized for provisioning persistent volumes.
- **repl**: Determines the number of replicas of Portworx volumes to maintain for high availability, with the minimum value set to 2.
- **io_profile**: Set to db_remote.Recommended IO profile setting for databases/StatefulSet workloads.
- **io_priority**: Set to high. Recommended setting for databases/StatefulSet workloads specifies the priority of IOs.
- **fs**: Set to xfs.Recommended setting for AlloyDB Omni. Specifies the type of file system to be provisioned.

Below is the StorageClass definition for reference. It establishes a StorageClass that creates Persistent Volumes (PVs) with XFS filesystem, high I/O priority, db_remote as I/O profile, and three replicas to guarantee high availability at the storage layer.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: postgres-sc
provisioner: kubernetes.io/portworx-volume
parameters:
 repl: "3"
 io_priority: "high"
 fs: "xfs"
 io_profile: "db_remote"
allowVolumeExpansion: true
```

AlloyDB Operator and AlloyDB Omni

AlloyDB Omni offers advanced features for high availability, scalability, and data management, making it suitable for modern cloud-native architectures. The process of deploying AlloyDB Omni on Kubernetes demonstrates how database management integrates seamlessly with container orchestration, resulting in streamlined operations and improved performance. This setup involves two steps.

1. Deploy and configure AlloyDB Operator using YAML or Helm charts.

- Setup environment variables:

```
export GCS_BUCKET=alloydb-omni-operator
export HELM_PATH=$(gsutil cat gs://alloydb-omni-operator/latest)
export OPERATOR_VERSION="${HELM_PATH%%/*}"
```

- Download AlloyDB Operator:

```
gsutil cp -r gs://$GCS_BUCKET/$HELM_PATH ./
```

- Install AlloyDB Operator:

```
helm install alloydbomni-operator alloydbomni-operator-${OPERATOR_VERSION}.tgz
--create-namespace --namespace alloydb-omni-system --atomic --timeout 5m
```

2. Create instances for CRDs to deploy AlloyDB clusters.

After setting up the AlloyDB Operator in the Kubernetes cluster, create an AlloyDB Omni database cluster by applying a manifest similar to the following:

```
apiVersion: alloydbomni.dbadmin.goog/v1
kind: DBCluster
metadata:
 name: alloydb
spec:
 availability:
 numberOfStandbys: 2
 enableAutoFailover: true
 databaseVersion: "15.5.0"
 primarySpec:
 features:
 memoryAgent:
 enabled: true
 adminUser:
 passwordRef:
 name: password-alloydb
 resources:
 cpu: 8
 memory: 200Gi
 disks:
 - name: DataDisk
 size: 200Gi
 storageClass: postgres-sc
```

This will create a DBCluster named “alloydb” with HA configured at the database layer (one primary and two replica pods) in the specified namespace. The DBCluster needs to be started using the “kubectl patch” command.

Conclusion

This reference architecture meticulously outlines an advanced, integrated solution designed to support AlloyDB Omni on Kubernetes, leveraging Portworx and OpenStack. By integrating Pure Storage FlashArray, the architecture achieves exceptional performance, scalability, and resilience, essential for handling both transactional and analytical workloads in container-based environments. The design takes full advantage of modern cloud-native capabilities, ensuring that AlloyDB Omni is not only adaptable and efficient but also robust against the dynamic demands of enterprise computing. Organizations adopting this architecture can expect a seamless, flexible deployment of databases with assured high availability and robust data protection, making it a compelling choice for enterprises striving to maximize operational efficiency and technological investment in their cloud-native transformation journeys.

Additional Resources

More details on managing AlloyDB DBClusters on Kubernetes can be found in the AlloyDB Documentation.

The Portworx documentation can be used to provide a deeper understanding of the different technical aspects not mentioned in this reference architecture.

PS2630-01-en 06/24