PURESTORAGE®
Uncomplicate Data Storage, Forever

WHITE PAPER

MongoDB on Portworx

Design Guide

Contents

Introduction

MongoDB has emerged as a leader among database management systems evidenced by its DB-Engines ranking as of September 2023. This leadership is attributed to the ability to scale cheaper, query faster, pivot easier and develop faster while driving significant innovation for the business processes making use of its data management capabilities.

Containerization continues to rise in popularity due to its ability to create and deploy applications faster and more securely. Organizations that wish to take advantage of containerization for MongoDB will be presented with any number of decisions as to how this adoption will be implemented.

This design guide will assist with understanding how Portworx® can be used to implement a fully integrated solution for persistent storage, backup, disaster recovery, data security, cross cloud data migrations and automated capacity management for MongoDB running on Kubernetes.

How to Use This Guide

This Design Guide functions as a comprehensive asset aimed at architects, administrators and engineers who are eager to understand how to design MongoDB in Kubernetes with Portworx to meet business requirements.. Within this guide, you will find invaluable insights into crafting a resilient, scalable, and secure ecosystem meticulously suited for executing MongoDB workloads within the dynamic realm of Kubernetes. This environment is powered by the dynamic storage capabilities that Portworx has to offer.

MongoDB

MongoDB is an open-source, NoSQL database designed to store and manage large volumes of unstructured and semi-structured data. It belongs to the family of document-oriented databases, where data is stored in flexible and schema-less documents instead of traditional relational tables. MongoDB is a popular choice with developers for developing scalable applications with varying data schemas.

The following are key features of MongoDB:

Document-Oriented: MongoDB organizes data in flexible, JSON-like documents called BSON (Binary JSON). These documents can vary in structure, allowing for easy handling of evolving and dynamic data.

Scalability: MongoDB is built to handle horizontal scalability, meaning it can distribute data across multiple servers or clusters, enabling high availability and accommodating large-scale applications with ease.

High Performance: MongoDB's architecture and indexing mechanisms allow for efficient read and write operations, providing low latency and high throughput. It supports various indexing options to optimize query performance.

Querying and Aggregation: MongoDB provides a powerful query language that supports a wide range of query operations, including filtering, sorting and field projection. It also offers an aggregation framework for performing complex data analysis and aggregations.

Flexible Data Model: MongoDB's schema-less nature allows for flexible data modeling, making it well-suited for rapidly changing or evolving data structures. This flexibility enables developers to iterate quickly and adapt to evolving application requirements.

Replication and Fault Tolerance: MongoDB supports replication, where data can be replicated across multiple servers to provide high availability and fault tolerance. It ensures that even if one server fails, the system can continue to operate without data loss.

Automatic Sharding: MongoDB can automatically partition data across multiple machines using sharding. Sharding enables horizontal scaling by distributing data based on a chosen shard key, allowing efficient distribution and management of large datasets.

Integration and Ecosystem: MongoDB integrates with popular programming languages and frameworks, providing native drivers and libraries for easy application development. It also integrates with other tools, such as data visualization and analytics platforms.

Challenges

MongoDB supports deployment into Kubernetes environments. Kubernetes has solidified its position as the conventional standard for orchestrating and managing containerized applications, spanning the entire spectrum from development stages to full-fledged production environments

A noteworthy trend is the growing comfort of enterprises to embrace cloud-native environments for the deployment and management of applications and databases. This alignment with modern business requirements emphasizes agility, cost-effectiveness and scalability, enabling them to leverage the full potential of cloud technology to foster innovation and provide top-quality services.

This trend encompasses contemporary distributed databases and data services, such as MongoDB. To meet the needs of stateful and distributed applications such as MongoDB databases, organizations require a Kubernetes storage solution that optimizes performance within the underlying infrastructure while ensuring the deployment is rooted in high availability and replication principles.

Effectively navigating the management of MongoDB within Kubernetes introduces a series of hurdles necessitating specialized strategies and expertise. These challenges span diverse facets of deployment, operation, scaling, and maintenance.

Challenges faced by organizations adopting Kubernetes for stateful applications includes:

Data Persistence: Ensuring steadfast and dependable storage for MongoDB in a Kubernetes setting introduces complexity. The configuration and upkeep of storage solutions aligned with MongoDB's requisites for endurance and uniformity hold paramount importance.

Deployment: Defining and creating multiple kubernetes objects such as configmaps, secrets, statefulsets, services etc. will be another challenging factor.

High Availability: Adjustments to MongoDB's inherent replication and failover mechanisms may be imperative for optimal performance within a Kubernetes cluster. Crafting seamless failover mechanisms and orchestrating data replication across nodes and pods mandates meticulous configuration.

Data Consistency: The dynamic nature of Kubernetes poses the challenge of upholding data consistency across MongoDB replicas and shards. Coordinating data synchronization and managing coherence become intricate endeavors.

Data Protection: The creation of effective backup and recovery strategies tailored to MongoDB instances managed by Kubernetes is intricate due to the distribution of data and the imperative for uniform snapshots.

Scaling Challenges: The orchestration of MongoDB's horizontal scaling whilst preserving data integrity and minimizing disruptions necessitates meticulous deliberation. Determining the optimal timing and methodology for scaling operations requires thoughtful planning.

Observability: The oversight of the health, performance and behavior of MongoDB clusters within Kubernetes necessitates the application of specialized tools and configurations. The establishment of effective alerting mechanisms is indispensable for prompt issue identification.

Upgrades and Maintenance: The coordination of MongoDB version upgrades and Kubernetes maintenance tasks to minimize downtimes and data loss mandates meticulous planning and rigorous testing.

Solution Overview

The above mentioned challenges can be easily addressed through the use of Portworx, the leading Kubernetes data services platform that provides a fully-integrated solution for persistent storage. Portworx is the most complete Kubernetes data services platform that also offers data protection, disaster recovery, data security, and automated capacity management. **Portworx Enterprise** offers a range of significant benefits for deploying MongoDB on Kubernetes.

Persistent Storage Solution for Stateful Workloads: Portworx Enterprise delivers Kubernetes-native persistent storage, effectively addressing MongoDB's essential requirement for data persistence in a stateful application context.

Enhanced Data Resilience and High Availability: Portworx ensures the resilience of data by replicating it across nodes and availability zones, thereby augmenting MongoDB's inherent high availability within Kubernetes clusters.

Seamless Dynamic Scalability: Portworx enables the effortless scaling of MongoDB clusters within Kubernetes, smoothly accommodating increased workloads and data expansion while preserving optimal performance levels.

Efficient Data Protection Capabilities: Leveraging Portworx's dedicated, efficient and consistent backup solution (Portworx Backup) designed for Kubernetes, it protects applications and data with automated backup and recovery capabilities.

Augmented Data Security: Portworx integrates encryption and access control mechanisms, fortifying data security measures and effectively safeguarding MongoDB instances and their associated data within Kubernetes environments.

Streamlined Day-2 Operations: Portworx simplifies post-deployment operations, automating the expansion of storage pools and volumes. This automation reduces the need for manual intervention and optimizes the utilization of available resources.

Optimal Performance Enhancement: Portworx enhances storage performance specifically for MongoDB workloads. This enhancement guarantees low latency and impressive throughput, aligning seamlessly with the rigorous requirements of demanding applications.

Customizable StorageClass Objects: Portworx empowers organizations to craft personalized Kubernetes StorageClass objects. These can be tailored to accommodate unique MongoDB requirements, encompassing factors such as replication ratios, snapshot scheduling and more.

Efficient Deployment Process: The incorporation of Portworx into Kubernetes streamlines the deployment of MongoDB, enhancing operational efficiency. This streamlined process is made possible by integrating Portworx's operator and custom resource, which simplifies the deployment of MongoDB clusters.

Portworx's Data Services(PDS), Database Platform-as-a-Service (DBPaaS) for Kubernetes which is built on top of Portworx Enterprise, reduces the complexity of deploying and managing data services such as MongoDB on top of Kubernetes clusters with few clicks and without the lock-in of a cloud platform or a Kubernetes distribution. It tackles a range of issues associated with data management and storage in containerized environments, including Kubernetes. Below are some of the key challenging factors that PDS effectively resolves.

Multi-Cloud and Hybrid Cloud Support: PDS is designed to span across multiple cloud providers and on-premises infrastructure, providing flexibility to organizations with diverse deployment environments. It seamlessly supports hybrid and multi-cloud strategies.

Data Portability: Applications and their associated data can be migrated across different Kubernetes clusters and environments with minimal modifications, streamlining migration and disaster recovery strategies.

Data Services Customization: Organizations have the flexibility to define custom StorageClass objects within PDS, tailoring replication factors, snapshot schedules and other parameters to meet the specific requirements of their applications.

Integration with Kubernetes: PDS integrates seamlessly with Kubernetes, providing an efficient pathway for setting up and managing data services in containerized environments.

Load Balancing: PDS balances the load of application pods across nodes, improving performance and availability.

MongoDB Design options on Kubernetes

There are multiple options available for designing a new MongoDB on Kubernetes deployment, each with its own considerations. The selection of the deployment method hinges on factors such as requirements, infrastructure, scalability needs and operational preferences.

The following are some MongoDB deployment options :

Helm Charts

Helm, a Kubernetes package manager, offers MongoDB Helm Charts for straightforward deployment. Helm charts provide flexibility and come with preconfigured settings, reducing the need for manual configuration. Tasks like configuring Replica Sets can be customized to meet specific business requirements.

Key considerations for the use of Helm Charts

Selecting appropriate image: Appropriate MongoDB container image selection from trusted image repositories, like Helm charts/Bitnami charts or the official MongoDB Docker image on Docker Hub. Verify its compatibility with the MongoDB version intended to use.

Selecting appropriate topology: Select a MongoDB deployment topology that aligns with unique business use cases and performance needs. Choose between single-node or multi-node Replica Sets, with the latter being recommended for enhanced availability and data redundancy.

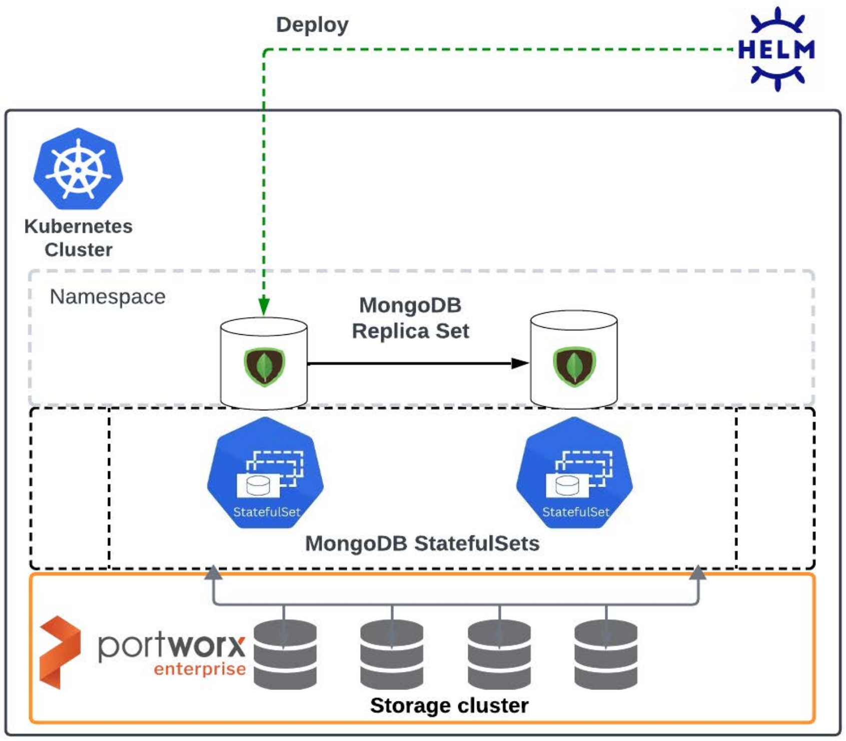

FIGURE 1 MongoDB on Kubernetes using Helm charts

Helm Chart Challenges

Complexity: Helm charts for MongoDB can be complex due to the various configuration options and dependencies, making it challenging to understand and customize them for specific use cases.

Customization: Helm charts may not cover every possible customization option for MongoDB deployments. Advanced configurations may require manual modifications, which can be error-prone.

Compatibility: Ensuring compatibility between Helm charts, MongoDB versions and Kubernetes versions can be challenging. Upgrading any of these components may require adjustments to the Helm chart.

Security: Helm charts may not always provide the most secure default configurations. It's crucial to review and adjust security settings to meet organization's standards.

MongoDB Enterprise Kubernetes Operator

Kubernetes operators are a powerful concept and framework for extending the functionality of Kubernetes and automating complex application management tasks. An operator is essentially a method of packaging, deploying and managing a Kubernetes application as a custom resource, using its own domain-specific language.

Operators enable the automation of application-specific operational tasks that would otherwise require manual intervention. They encapsulate expert knowledge and best practices for managing a particular application or service, allowing for consistent and reliable deployment, scaling, and maintenance.

MongoDB provides an official kubernetes operator (Licensed/Enterprise) designed to streamline the deployment and administration of MongoDB on Kubernetes. This operator enhances Kubernetes' capabilities to manage MongoDB tasks tailored to its requirements including tasks like backups, scaling, and configuration adjustments.

The operator constantly monitors the desired state of the MongoDB cluster and ensures that it aligns with the actual state. It automates various tasks such as creating and scaling Replica Sets, managing data distribution across shards, handling failover scenarios and applying updates or patches to the MongoDB deployment.

The MongoDB Enterprise Kubernetes Operator for MongoDB offers several advantages. Refer to the official MongoDB documentation for more information.

Simplified Deployment: The operator abstracts away the complexities involved in setting up a MongoDB cluster, simplifying the deployment process. Users can define the desired configuration declaratively and the operator takes care of the implementation details.

Scalability and High Availability: The operator enables effortless scaling of MongoDB clusters by automating the addition or removal of replicas and shards as required. It also ensures high availability by managing failover and maintaining data redundancy.

Self-Healing: Various operational tasks including node failure detection, replica synchronization and data recovery are automated by the operator. It actively monitors the health of the MongoDB deployment and takes proactive measures to maintain the desired state.

Integration with Kubernetes Ecosystem: The MongoDB Enterprise Kubernetes Operator seamlessly integrates with other Kubernetes features and tools leveraging functionalities such as storage volumes, networking and authentication mechanisms. This allows for consistent management and smooth integration with the broader Kubernetes ecosystem.

Enterprise Support: MongoDB provides regular updates to support new MongoDB versions, incorporate bug fixes, apply security patches and introduce feature enhancements.

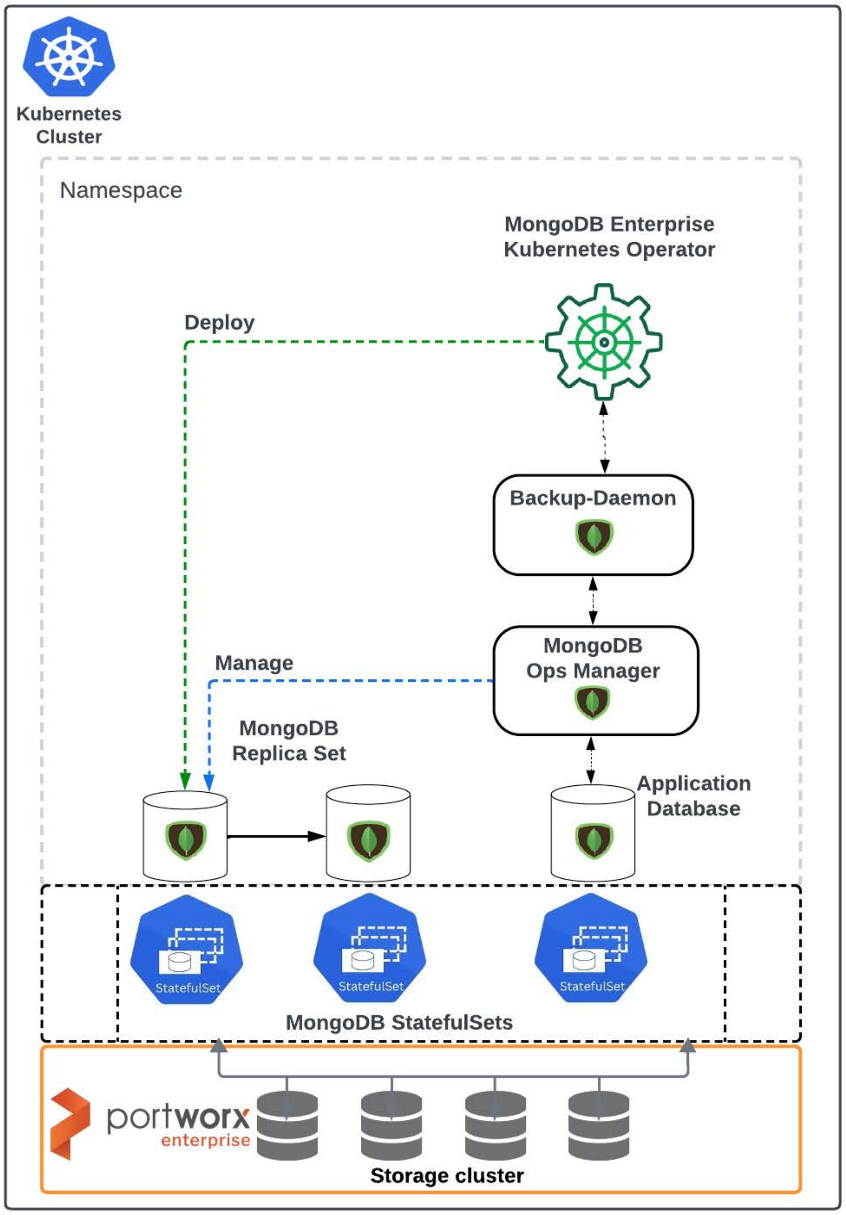

FIGURE 2 MongoDB on Kubernetes using MongoDB Enterprise Kubernetes operator

Design Architecture

The Design Architecture provides an overview of the conceptual and structural elements within an environment. It acts as a blueprint for creating, comprehending and conveying the design of MongoDB on Portworx, encompassing a range of associated benefits.

Design Benefits

Portworx offers several benefits to address various aspects of data management and storage in containerized environments, particularly within Kubernetes clusters. They cater to various aspects of data resilience, backup, migration and provisioning empower organizations to deploy and manage containerized workloads with confidence and efficiency.

Here are key benefits for MongoDB deployed on Portworx:

High Availability for MongoDB Persistent Storage

Portworx provides high availability (HA) for containerized environments on Kubernetes by ensuring data resilience and application uptime. It ensures that data remains accessible and applications stay operational, even in the case of hardware failures, node outages or other disruptions.

By leveraging Portworx, MongoDB attains High Availability through the following capabilities:

Data Replication: Portworx employs synchronous replication to create consistent and durable copies of MongoDB data. It replicates data across multiple nodes, ensuring that each replica remains up to date and consistent. This replication mechanism safeguards data against node failures, eradicating single points of failure within the MongoDB deployment.

Auto-Failover: In the event of a MongoDB hosted node failure, Portworx promptly detects the issue and initiates an automated failover process with help of STORK scheduler. The workload of the failed node seamlessly transitions to a healthy node, minimizing downtime and upholding MongoDB's availability.

Dynamic Provisioning and Scaling: Portworx enables dynamic provisioning and scaling of storage resources for MongoDB. As the workload grows, Portworx automatically allocates additional storage capacity to accommodate the increasing demands of the database. This flexibility empowers MongoDB deployments to scale effortlessly without disrupting ongoing operations.

Data Resynchronization: Portworx ensures efficient synchronization of a restored MongoDB node with replicated data from other nodes. This resynchronization process minimizes data inconsistencies and preserves the integrity of the MongoDB deployment.

Portworx simplifies the management and operation of MongoDB, delivering a reliable solution for organizations with critical data requirements.

Figure 3 illustrates the mechanism for ensuring high availability in the event of a node failure.

In this illustration, you can observe how Portworx STORK health monitoring plays a vital role in rescheduling MongoDB pod(s) onto healthy nodes following a failure. STORK takes charge in scenarios where the storage driver on a node encounters errors or becomes unavailable, automatically facilitating the failover of MongoDB pod(s). This process ensures that applications achieve true High Availability without requiring any user intervention.

Below are comprehensive steps outlining the process to ensure High Availability during failure scenarios.

1. Kubernetes monitors nodes/pods health.
2. If a node failure occurs, the Kubernetes scheduler, with assistance from the STORK scheduler, will initiate the creation of new pods on nodes where replicated volumes are available.

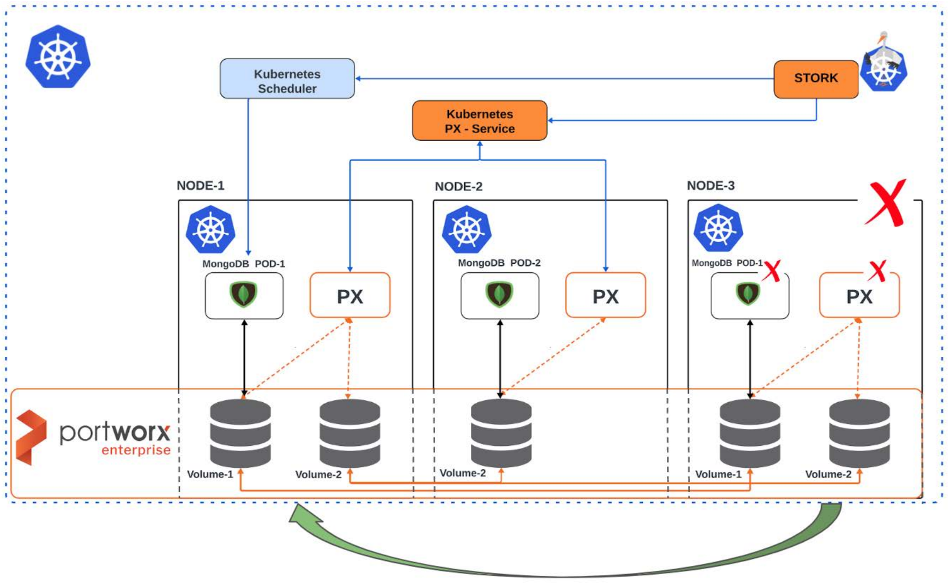

FIGURE 3 High Availability with Portworx

Moreover, PDS reinforces data protection with high availability at the database layer, ensuring seamless and fully automated recovery from instance failures. This is accomplished through the provisioning of the Enterprise MongoDB cluster consisting of 2 or more MongoDB nodes.

Volume Snapshots for Database Replicas

Database replicas of databases have a significant role in tackling various challenges including scaling, ensuring data availability across multiple clouds, supporting analytics/reporting, creating backup copies and facilitating the seeding or deployment of new systems for development and testing purposes.

Portworx snapshots are a valuable feature within the Portworx data management platform, offering users a reliable and efficient way to protect, manage and restore their data. With Portworx snapshots, users can create point-in-time copies of their data volumes, ensuring data integrity and providing an additional layer of data protection.

These snapshots are created instantaneously and with minimal impact on the performance of running applications or workloads, allowing businesses to maintain continuous operations. Portworx snapshots also optimize storage utilization by utilizing a copy-on-write mechanism, storing only the changes made to the original data after the snapshot creation.

Portworx snapshots seamlessly integrate with other Portworx Data Services, such as backup and disaster recovery solutions, providing a comprehensive data management ecosystem. They can also be automated and orchestrated through APIs and integration with popular frameworks like Kubernetes, simplifying snapshot creation and management at scale. Snapshots with Portworx for MongoDB can be application consistent or crash consistent, which is dependent on the requirements of the snapshot being created.

Another key benefit of Portworx snapshots is their cross-cloud and multi-cloud support. Whether on-premises or on various cloud environments, users can leverage Portworx snapshots to ensure consistent data protection and portability across different infrastructures.

By utilizing these snapshots within the Portworx platform, businesses can enhance their data management strategies, improve resiliency and confidently safeguard their valuable data assets.

For data services managed by PDS, like MongoDB, it incorporates an integrated backup and restoration functionality that utilizes Portworx's volume snapshot process. Users also have the option to utilize the Portworx CLI.

Figure 4 illustrates how MongoDB leverages Portworx volume snapshot feature for creating the procedure for generating new MongoDB instance(s) and protecting data in the event of application, database, or infrastructure-level disasters.

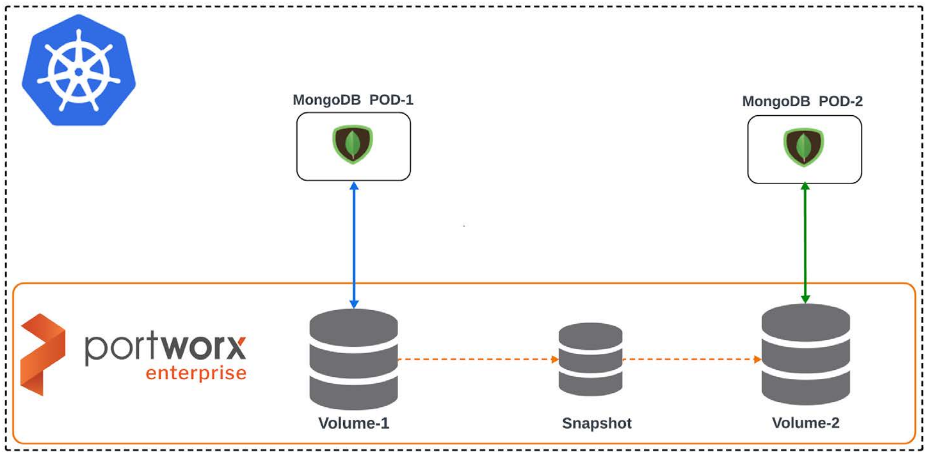

FIGURE 4 Portworx Volume snapshot

Data Protection

Portworx offers robust capabilities for backup and restore, providing organizations with dependable and efficient data protection for their critical workloads. With Portworx's backup and restore functionality, organizations can safeguard their data, recover from potential disasters and ensure uninterrupted operations.

Key features of Portworx's backup and restore include:

Point-in-Time Backups: Portworx enables the creation of backups that capture the exact state of data volumes at specific moments, serving as reliable restore points for recovery purposes.

Incremental Backups: Portworx performs incremental backups, capturing only the changes made to the data since the last backup. This approach reduces backup time and minimizes storage space requirements.

Application Consistency: Portworx ensures that backups are consistent with the applications running on the volumes, guaranteeing data integrity and recoverability, particularly for transactional systems like databases.

Flexible Restore Options: PX-Backup provides flexible restore options, allowing users to restore entire volumes and application specific resources. This granularity enables quick and targeted recovery, minimizing downtime.

Recovery capability: Portworx's backup and restore capabilities are vital for disaster recovery scenarios. In the event of data loss or system failure, organizations can rely on Portworx backups to recover critical data swiftly and resume operations.

Cross-Cloud and Multi-Cloud Support: Portworx's backup and restore operations are not limited to specific cloud providers or infrastructures. It facilitates backup and restore operations across multiple cloud platforms and on-premises environments, enabling data portability and flexibility.

Portworx simplifies backup management and ensures continuous business operations, even in the face of potential data loss or system failures.

Furthermore, PDS hosted data services like MongoDB can benefit from PDS's integrated backup and restore functionality, which safeguards the data service specific deployment components including data. Backup options can be activated and scheduled either during the initial deployment configuration or after the deployment is finalized.

Figure 5 illustrates the key components and capabilities of Portworx backup, facilitating the backup and restoration of kubernetes objects and data across diverse kubernetes clusters deployed on multiple cloud platforms (such as AWS, Azure, Google Cloud and on-premises) to S3/Blob storage repositories.

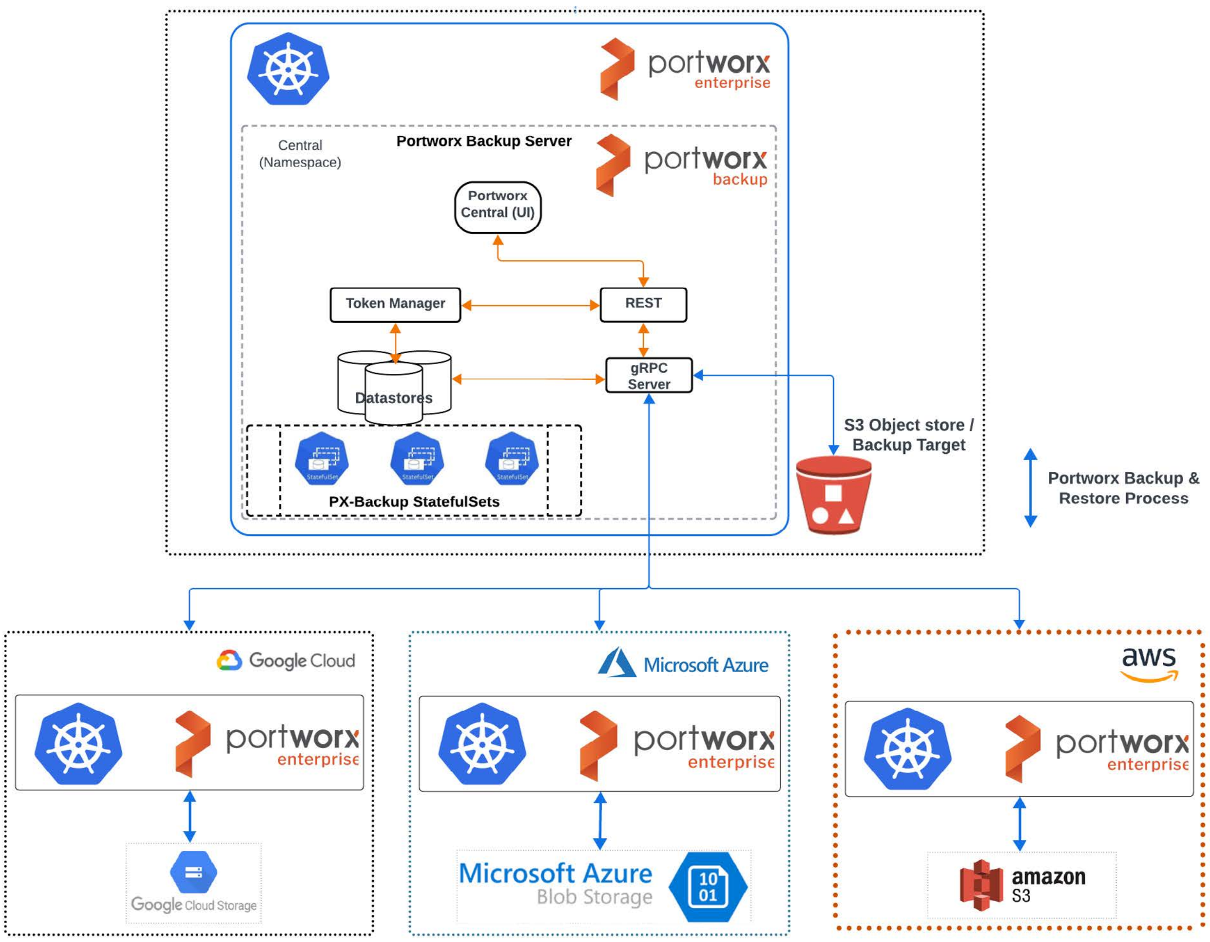

FIGURE 5 Portworx Backup & Restore

NOTE: *For detailed information on the above solution benefits using Portworx refer to MongoDB on Portworx Deployment guide.*

MongoDB on Portworx

This section outlines the design factors to setup and deploy MongoDB on Portworx. Below are the fundamental design considerations.

Infrastructure Choice

Choose the infrastructure provider that best aligns with your organization's requirements and budget, focusing on top-tier compute and networking capabilities. This can include options like AWS, Azure, Google Cloud, or on-premises solutions.

Storage

Storage can be any of block storage devices that are unmounted. These devices can include raw disks, drive partitions, LVM volumes, or cloud-based block storage(s).

Kubernetes

Choose any of Kubernetes platforms such as EKS, AKS, GKE, Redhat OpenShift etc.. to configure Helm charts, MongoDB Enterprise kubernetes operator which are prerequisites for MongoDB deployments.

Portworx

Portworx uses block storage devices to create and manage distributed storage pools. You should have one or more block storage devices attached to each node in the kubernetes cluster. These devices can be physical disks, NVMe drives or cloud-based storage volumes.

- Portworx Enterprise setup requires the necessary StorageClass definitions to create a storage cluster.
- Portworx Backup to be configured for data protection capabilities.
- Optional configuration of Portworx Data Services for deploying and managing MongoDB deployments. (Applies only if MongoDB is deployed using Portworx Data Services).
- Any S3/ S3 compatible object storages to be configured as backup targets to leverage Portworx's data protection capabilities.

MongoDB

For MongoDB, Kubernetes nodes should utilize processors based on the x86-64/AMD64 architecture. The StorageClass definition chosen for MongoDB deployments must be compatible with either XFS or EXT4 file systems. The MongoDB Enterprise Kubernetes Operator is designed to work seamlessly with actively supported versions of MongoDB, which depends on the base image used for the MongoDB database resource.

There are no specific restrictions regarding the selection or restriction of servers, OS distributions, Kubernetes variants, database versions or Kubernetes operators.

Customers are encouraged to utilize their chosen servers, storage solutions and the latest versions of Linux-based OS, Kubernetes distributions, database versions and Kubernetes operators based on their preferences.

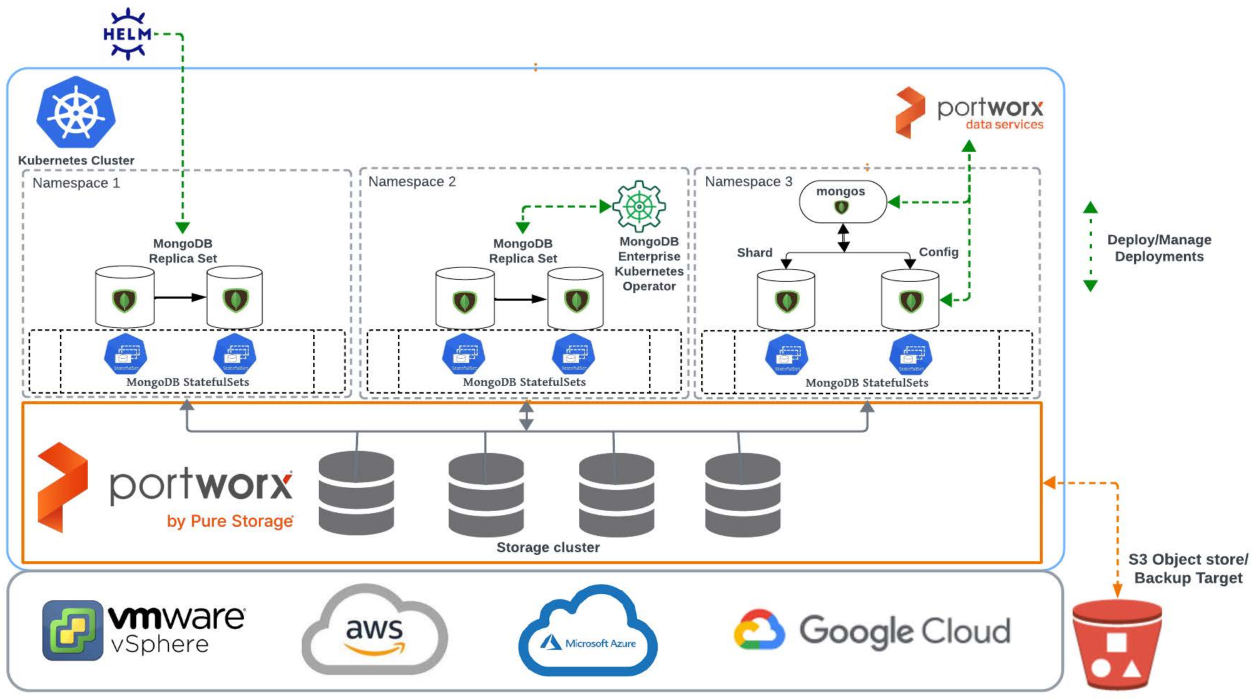

FIGURE 6 Logical architecture of the MongoDB on Portworx

MongoDB using Portworx Data Services (PDS)

Organizations have the option to seamlessly incorporate MongoDB Enterprise Advanced, a self-managed database, in conjunction with Portworx to facilitate its deployment in private cloud, on-premises or hybrid environments.

PDS provides cloud-native Database Platform As a Service that allows the deployment and management of several data services including MongoDB. A self-managed, single platform interface that provides automated life cycle management of data services, enhances availability, performance and reduces infrastructure costs.

PDS boasts a minimalist interface that simplifies the setup and execution of MongoDB workloads, delivering an excellent developer experience. PDS offers support for the following features.

Deployment & Configuration: PDS significantly simplifies the configuration and deployment of MongoDB clusters when compared to traditional MongoDB Kubernetes operators that often demand substantial manual intervention for setup, configuration and deployment. PDS streamlines the process by guiding users through a simple setup procedure. This involves steps like designating a deployment target, choosing the MongoDB version, specifying resource allocations (CPU, memory and storage), defining cluster size and configuring backup schedules. The result is the effortless deployment of a MongoDB instance complete with connection strings.

High Availability: PDS ensures data protection through a two-fold approach, addressing both the database and storage layers. Within the database layer, PDS sets up an Enterprise MongoDB Shard cluster with configured shards, config servers, and MongoDB routers. This setup operates as a self-healing Replica Set, guaranteeing automatic failover in response to unexpected events, thus enhancing high availability. Additionally, at the storage layer, PDS utilizes Portworx volume replication strategies to fortify data protection and prevent data loss. In the event of instance failures, recovery is seamless and entirely automated.

Scalability: PDS provides a horizontal scaling solution for databases by seamlessly incorporating sharding into applications. In this approach, MongoDB distributes data across multiple Replica Sets, referred to as shards. MongoDB's automated balancing mechanism ensures equitable data distribution across these shards, whether data volumes increase or the cluster size changes. Sharding liberates MongoDB deployments from the constraints of a single server, such as potential bottlenecks in RAM or disk I/O, all while maintaining the simplicity of application management. Moreover, PDS offers the capability to vertically scale workloads, adapting to changing demands by dynamically adjusting resources like CPU, memory and storage. This can be achieved effortlessly by selecting and applying predefined or customized templates.

Data Protection: PDS prioritizes data protection through a fully managed backup service, offering continuous, consistent backups, point-in-time recovery and customizable retention policies. These functionalities can be effortlessly initiated with just a few clicks.

Observability: Deployment and database metrics are systematically arranged to enable the monitoring of infrastructure and database-level parameters through Prometheus and Grafana dashboards. This setup offers the flexibility to configure and tailor observability metrics using APIs.

Seamless Patching/Upgrades: PDS offers certified upgrades and patches for MongoDB deployments, streamlining the upgrade process by automating intricate tasks, ensuring data integrity and providing user-friendly interfaces and support. This ensures a seamless transition to a new MongoDB version with ease and confidence.

Copilot - AI Powered Tool: PDS strives to narrow the divide between end users and database administrators, facilitating a user-friendly and intuitive database interaction. Through Copilot, it offers guidance and information about the current data service using natural language queries, sparing users the complexity of writing intricate SQL commands. Copilot comprehends and interprets natural language input, transforming it into the relevant database queries, reducing the necessity for end users to possess extensive database system knowledge.

This section outlines the environment utilized in our lab to validate MongoDB deployment using Portworx Data Services.

1. Add/register deployment target (a kubernetes cluster) to host MongoDB workloads.
2. Deploy 2-node MongoDB shard cluster with mongos routers and config server through the PDS portal.

Figure 7 illustrates MongoDB cluster deployment on kubernetes using Portworx Data Services.

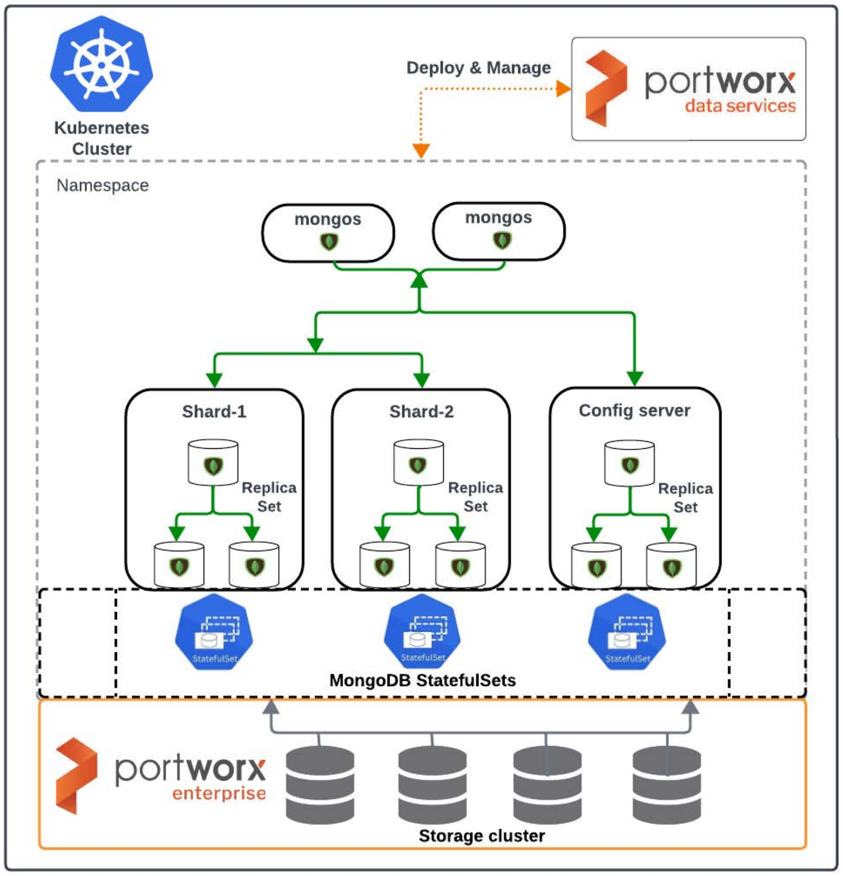

FIGURE 7 Logical architecture of the MongoDB deployment using PDS.

Best Practices and Design Recommendations

Deploying MongoDB on Kubernetes can be a complex task, but following best practices can help ensure a smooth and reliable deployment. Here are some best practices for deploying MongoDB on Kubernetes:

Use StatefulSets: StatefulSets are ideal for running stateful applications like MongoDB on Kubernetes. They provide stable network identities and persistent storage for each pod, ensuring that each MongoDB instance has a unique hostname and stable storage. This helps with data persistence and recovery during pod rescheduling.

Distinct Namespaces: Establish distinct namespaces and allocate all essential resources, including Configmaps and services, within these designated namespaces. This approach ensures segregation and streamlines resource management.

ConfigMaps: Store all custom scripts and containers/pods configuration within ConfigMaps.

Secret store: Use of appropriate secret stores like Hashicorp Vault, AWS Secret store etc., are recommended to store secrets, such as API keys, passwords and certificates which are sensitive pieces of information. Storing them in a secret store encrypts them, making it more difficult for unauthorized users.

PersistentVolumes (PVs) and PersistentVolumeClaims (PVCs): Implementing various PVCs to handle persistent storage for both data and log files in MongoDB is considered a best practice. This approach ensures data availability even during pod restarts or rescheduling, enhances performance and optimizes resource allocation, particularly when the appropriate storage class is chosen. This practice is particularly important in production database environments where performance, reliability and recoverability are critical considerations.

Resource Requests and Limits: Set resource requests and limits for CPU and memory to prevent overcommitment of resources. Understand workload's resource requirements and adjust these settings accordingly.

Node Affinity/Anti-Affinity: Use node affinity and anti-affinity rules to control pod placement. Ensure that MongoDB pods are placed on nodes with suitable resources and avoid scheduling multiple pods on the same node.

Authentication and Authorization: Enhance the security of your MongoDB deployment by activating both authentication and authorization. For users, Enable Role-Based Access Control (RBAC) authentication.

Network Policies: Implement Kubernetes network policies to control traffic to and from MongoDB pods. This helps enforce security and restricts unauthorized access. Configure load balancer services to access MongoDB externally.

Monitoring and Alerting: Set up monitoring and alerting solutions to track the health and performance of MongoDB deployment. Tools like Prometheus and Grafana can be used to monitor MongoDB metrics. Refer official documentation from MongoDB on how to set up Prometheus for MongoDB.

Security: Follow MongoDB's security best practices, including enabling SSL/TLS for data in transit, limiting network exposure and regularly updating MongoDB to address security vulnerabilities.

Designing MongoDB schemas/objects appropriately and creating suitable indexes is critical for achieving optimal performance, scalability, storage efficiency and a positive user experience in applications. It's a foundational practice in MongoDB database administration and application development.

These best practices allow deploying MongoDB on Kubernetes in a secure, scalable and reliable manner, ensuring the smooth operation of the database in a containerized environment.