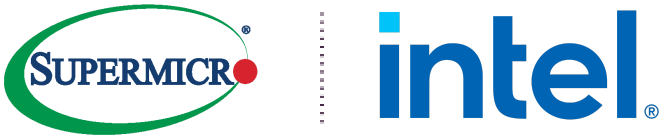


# Reference Architecture:

# Kubernetes with Block Storage Solutions for High-Performance Workloads

June 2025

Rev 1.0

This reference architecture provides a comprehensive guide to deploying a Kubernetes environment optimized for any workload - from general-purpose applications to high-performance workloads such as databases and AI/ML. Leveraging the combined power of Ceph and Lightbits storage, this architecture ensures robust storage solutions. It covers key aspects like hardware setup, cluster configuration, storage integration, application deployment, monitoring, and cost optimization. A key advantage of this architecture is that Lightbits storage can be added to an existing Kubernetes deployment without significant rearchitecting - enabling a seamless upgrade path to software-defined high-performance storage. By following this architecture, organizations can build highly available and scalable Kubernetes platforms to meet the diverse needs of modern applications running in containers, as well as legacy applications running as KubeVirt Virtual Machines (VMs).

# Table of Contents

# 1. Executive Summary

This reference architecture outlines a comprehensive approach to deploying a Kubernetes environment optimized for both general-purpose and high-performance workloads, such as databases and AI/ML. By leveraging the combined power of Ceph and Lightbits storage, this architecture ensures robust storage solutions and seamless integration with existing Kubernetes deployments.

Key features and benefits of this architecture include:

- **Comprehensive Coverage:** The architecture covers all essential aspects of Kubernetes deployment, from cluster configuration to application deployment, monitoring, and optimizations.
- **Optimized Storage Solutions:** Leveraging Ceph and Lightbits storage, the architecture provides a robust and scalable storage solution, capable of handling both general-purpose and high-performance workloads.
- **Seamless Integration:** Lightbits storage can be seamlessly integrated into existing Kubernetes environments without significant rearchitecting, enabling a smooth transition to high-performance storage.
- **Efficient Data Management:** Implementing effective strategies for data placement and relocation between Ceph and Lightbits.
- **Performance and Scalability:** The architecture is designed to deliver high performance and scalability, making it suitable for demanding applications such as databases and AI/ML workloads.
- **Sustainability:** The architecture emphasizes cost optimization strategies, including the efficient utilization of resources and the selection of cost-effective storage solutions.
- **Versatility:** Supporting both containerized and virtualized (using KubeVirt) applications.

By following this architecture, organizations can build highly available, scalable, and sustainable Kubernetes platforms to meet the diverse needs of modern applications, including both containerized and virtualized (using KubeVirt) legacy applications - all orchestrated by Kubernetes.

# 2. Introduction

Kubernetes has increasingly become the industry standard for container orchestration, enabling organizations to deploy, scale, and manage applications more efficiently. As enterprises seek alternatives to traditional infrastructure platforms such as VMware, many are turning to

Kubernetes - not only for cost reduction but also for its cloud-native design, ecosystem maturity, and operational flexibility. This shift is fueling modernization efforts across infrastructure, storage, and application delivery pipelines.

However, deploying and operating production-grade Kubernetes environments - particularly those supporting high-performance, stateful container workloads - presents several challenges. These include maintaining storage performance at scale, ensuring consistent data resilience, and simplifying operations across diverse hardware and network topologies. Organizations need robust, scalable architectures that address these complexities, while meeting performance and availability expectations.

To support these requirements, this reference architecture presents an integrated solution built on Kubernetes with Lightbits and Ceph as storage backends. This combination delivers a unified infrastructure platform capable of supporting containerized applications with high-performance NVMe/TCP storage (Lightbits) and general-purpose, scalable storage for secondary workloads and data services (Ceph). The architecture is designed for flexibility, scalability, and seamless integration into existing data center environments.

This paper is organized into several key sections. It begins with an overview of Kubernetes fundamentals, including its core components and cluster configuration best practices. It then introduces the Lightbits cluster model and recommended network design to support high-performance NVMe/TCP storage. A high-level architectural view of the integration between Kubernetes, Lightbits, and Ceph is presented, followed by detailed steps for deploying and configuring Kubernetes alongside Lightbits. The paper concludes with a performance evaluation section that presents benchmark results from synthetic workload testing using the FIO tool, highlighting a comparative analysis between Lightbits and Ceph.

Note that while Ceph is included in this reference architecture as part of an integrated solution providing comparative evaluation and context, its deployment and configuration are outside the scope of this paper. More detailed information for Ceph installation can be found here.

# 3. Kubernetes Fundamentals

Kubernetes is an open-source platform designed to automate the deployment, scaling, and operation of application containers across a cluster of servers, providing a robust foundation for cloud-native applications and virtual machines. This section delves into the core components of Kubernetes, discusses best practices for cluster configuration, outlines effective application

deployment strategies, and highlights essential monitoring and logging techniques. More detailed information can be found here.

# 3.1. Core Components

## 3.1.1. Control Plane

The control plane is the brain of the Kubernetes cluster, responsible for making global decisions about the cluster (e.g., scheduling), as well as detecting and responding to cluster events (e.g., starting up a new pod when a deployment's replicas field is unsatisfied).

- kube-apiserver: Acts as the front-end to the control plane, exposing the Kubernetes API and handling and processing REST requests, updating the corresponding objects in the database (etcd).
- etcd: A reliable, distributed data store that serves as Kubernetes' backing store for all cluster data.
- kube-scheduler: Watches for newly created pods that have no node assigned, and selects a node for them to run on.
- kube-controller-manager: Runs controller processes, which handle routine tasks in the cluster.
- cloud-controller-manager: Lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that just interact with your cluster.

## 3.1.2. Node Components

These components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

- kubelet: An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.

- kube-proxy: Maintains network rules on nodes. These network rules allow network communication to your pods from network sessions inside or outside of your cluster.

- Container Runtime: The software that is responsible for running containers.

### 3.1.3. KubeVirt Extension

KubeVirt is an extension for Kubernetes that allows it to run virtual machines alongside container workloads, providing a unified development platform for mixed environments. By integrating directly with the existing Kubernetes ecosystem, KubeVirt makes it possible to manage virtual machine infrastructure with the same resources and tools that are used for container orchestration. This integration simplifies operations for teams that need to run both containers and virtual machines on the same cluster, enhancing the cluster's flexibility and utilization.

KubeVirt operates by introducing new resource types through Kubernetes' Custom Resource Definitions (CRDs), which are then managed by the standard Kubernetes API. This approach allows users to apply familiar Kubernetes techniques and patterns to manage virtual machines, including configuration, networking, storage, and security policies that are consistent with those used for containerized workloads. As a result, KubeVirt is an ideal solution for organizations looking to modernize their legacy applications by migrating them to a cloud-native environment without sacrificing the traditional VM-based applications they rely on..

Note: While KubeVirt is supported in this architecture, the performance benchmarking presented in this paper was conducted exclusively using containerized workloads to maintain test consistency and focus.

## 3.2. Cluster Configuration Best Practices

Configuring a Kubernetes cluster appropriately is vital to ensure that it meets the demands of your applications while maintaining efficiency and reliability. The following are some best practices for configuring the number of control plane nodes and worker nodes.

### 3.2.1. Control Plane Nodes

The control plane manages the cluster's global state, and its configuration directly impacts the cluster's performance and stability.

- High Availability: For production environments, it is recommended to have at least three control plane nodes to ensure high availability. This setup helps in preventing downtime and makes the cluster more resilient to failures.

- Load Balancing: Use a load balancer in front of the control plane nodes to distribute traffic and requests evenly across them, enhancing the responsiveness and reliability of cluster operations.

- Resource Allocation: Control plane nodes should be provisioned with sufficient CPU and memory resources to handle the cluster management workload. These nodes primarily deal with API requests, etcd storage, scheduling, and other management tasks.

### 3.2.2. Worker Nodes

Worker nodes are where your applications and workloads actually run. Scaling and configuring these nodes depend largely on the applications' requirements.

- Scaling Based on Workload: Start with a baseline configuration based on your expected workload and adjust as necessary. It's common to scale out with more nodes rather than scaling up with more powerful hardware, as this provides better resilience and fault tolerance.

- Diverse Workloads: For clusters supporting a diverse range of applications, consider segregating workloads across different pools of worker nodes. This segregation can be based on resource intensity, security requirements, or compliance needs.

- Resource Matching: Ensure that the worker nodes' hardware matches the requirements of the applications they are meant to support. For instance, workloads requiring intensive computational power or memory might need nodes with higher specs, whereas lighter applications can run on standard nodes.

### 3.2.3. Pods

A Kubernetes pod is the smallest deployable and manageable unit in a Kubernetes cluster. Each pod consists of one or more containers (the basic building blocks of applications in Kubernetes) that share the same resources and are deployed together. When an application is deployed in Kubernetes, it is encapsulated within a pod.

### 3.2.4. General Considerations

- Redundancy: Always plan for more resources than your estimated needs to handle unexpected spikes or failures.
- Network Configuration: Proper network setup is crucial, especially for clusters with high communication demands between nodes. Implementing network policies and segmentations can help in managing traffic and securing cluster communications.
- Monitoring and Adjustment: Continuously monitor the performance and health of both control plane and worker nodes. Use monitoring insights to adjust resources and configurations dynamically based on the changing demands.

### 3.2.5. Node Pools

- Define groups of nodes within a cluster that have similar configurations. Using node pools, you can specify versions and size of resources for nodes grouped together.

### 3.2.6. Network Configuration

- Ensure proper network setup to allow communication between node components through the control plane, often involving network policies that specify how pods communicate with each other and other network endpoints.

## 3.3. Application Deployment

### 3.3.1. Containers

- A container is a lightweight, standalone, and portable package that includes everything needed to run an application and is deployed within a pod.

### 3.3.2. KubeVirt

- Extends Kubernetes by adding virtual machine support to its resource model. It allows existing virtualized applications to run alongside containerized applications in a unified cloud-native platform. For more information on how to deploy applications on virtual machines using KubeVirt, please refer to the KubeVirt documentation.

## 3.4. Monitoring and Logging

### 3.4.1. Monitoring Tools

- Utilize tools like Prometheus for monitoring Kubernetes nodes and applications, ensuring you can track everything from CPU to memory usage across your cluster dynamically.

### 3.4.2. Logging

- Implement centralized logging with tools such as open-source Fluentd, which aggregates logging data from nodes and sends it to a central data store such as Elasticsearch for analysis.

# 4. Optimized Storage Solutions

## 4.1. Ceph Storage Solution

Ceph is a unified, distributed storage system designed for performance, reliability, and scalability. The core of Ceph's architecture is the Reliable Autonomic Distributed Object Store (RADOS), which manages the storage of data objects across multiple servers, ensuring high availability and redundancy through data replication and striping. RADOS is the foundation for several key components of Ceph, each catering to different storage needs:

1. Ceph Block Device (RBD): RBD provides block storage capabilities that are ideal for environments like virtual machines and database systems - where direct, unstructured, byte-level access to data is required. It leverages RADOS for data replication and fault tolerance, ensuring robustness and high performance.

2. Ceph File System (CephFS): This is a POSIX-compliant file system built on top of Ceph's distributed object store, RADOS. CephFS endeavors to provide a state-of-the-art, multi-use, highly available, and performant file store for a variety of applications, including traditional use cases such as shared home directories, HPC scratch space, and distributed workflow shared storage.

3. Ceph Object Storage (RADOS Gateway): Known as RGW, it provides interfaces compatible with APIs from major cloud storage services like Amazon S3 and OpenStack Swift. This makes Ceph a good choice for cloud storage applications, offering robust scalability and compatibility with a broad ecosystem of applications.

Additionally, Ceph's architecture is supported by several daemon processes that maintain the cluster's health and performance:

- Ceph Monitors (ceph-mon): Ceph monitors serve as the single source of truth for the cluster map. They maintain a master copy of the cluster map, monitor the health of the cluster, and provide consensus among the nodes about the state of the cluster.

- Ceph Managers (ceph-mgr): These run alongside monitor daemons to collect and provide metrics, manage system resources, and perform essential non-data-path operations that help in the overall management of the cluster.

- Ceph OSDs (Object Storage Daemons): These are responsible for managing data on local storage, handling data replication, recovery, rebalancing, and providing detailed information to Ceph Monitors and Managers about operational status.

More information about Ceph can be found here.

### 4.1.1. Ceph Storage for Kubernetes

For many Kubernetes deployments where cost-effectiveness and scalability are prioritized over performance, Ceph serves as a commonly chosen storage solution.

Lightbits offers high-performance NVMe/TCP storage solutions that significantly boost throughput and reduce latency, making it ideal for latency-sensitive, stateful applications such as databases, real time analytics and AI/ML. Organizations can enhance their current storage deployments (such as Ceph) with Lightbits, creating a more robust, scalable, and performant software defined solution tailored to specific application needs. Workloads can also be gradually migrated from their existing storage systems to Lightbits for improved performance.

The following details how Ceph integrates within Kubernetes environments:

1. Dynamic Volume Provisioning: Through the use of Ceph as a storage backend, Kubernetes can leverage dynamic volume provisioning capabilities. This allows for automatic creation and association of storage resources on demand without manual administrator intervention, thereby improving operational efficiency and reducing time-to-deployment for applications that require persistent storage.

2. Scalability and Flexibility: As Kubernetes clusters grow, Ceph can scale out to manage increasing data This ensures that storage capacity can grow with the needs of applications without requiring significant architectural changes.

3. High Availability: Ceph's replication features ensure that data stored in Kubernetes applications is highly available and resilient against hardware failures. Ceph automatically distributes and replicates data across the cluster, providing fault tolerance and reducing the risk of data loss.

4. Flexible Storage Options: Ceph offers block, object, and file storage within a unified platform, its versatility supports Kubernetes environments that handle diverse workloads with varying storage needs.

5. Software Defined By using Ceph, organizations can leverage commodity hardware to build out their storage solutions, avoiding the need for expensive proprietary storage hardware.

## 4.2. Lightbits Storage Solution

Lightbits represents a paradigm shift in how storage resources are managed and utilized in modern data centers. Lightbits' unique disaggregated, software-defined, NVMe over TCP, block storage architecture - combined with Intelligent Flash Management and enterprise-rich data services - makes it the only complete cloud data platform that's easy to provision and simple to manage, while also highly scalable, performant, and cost-efficient.

At its core, it leverages NVMe over TCP (NVMe/TCP) storage protocol technology, which was invented by Lightbits and then ratified and given to the open-source community, to deliver low-latency, high-performance block storage over a regular Ethernet network. This not only reduces the cost and complexity compared to other technologies like Fibre Channel or Infiniband, but also democratizes access to high-speed NVMe storage.

The software-defined nature of Lightbits' solution means it can be deployed on commodity servers and scaled across a distributed architecture without the need for specialized networking equipment or proprietary hardware. This flexibility combined with advanced features such as Intelligent Flash Management, ElasticRAID, in-line compression, software encryption, thin provisioning, snapshots, and clones produces a solution that enhances data durability, optimizes storage efficiency, and reduces overall operating costs.

## 4.2.1. Lightbits Storage for Kubernetes

Integrating Lightbits with Kubernetes offers a compelling advantage for environments that demand high performance and efficiency in handling storage operations. Lightbits provides a software-defined storage solution that is natively designed with NVMe/TCP, which is a key consideration for data-intensive applications requiring low latency and high throughput. Even if a single application doesn't initially demand significant performance, at scale, with hundreds or thousands of applications running concurrently, storage performance requirements can become extremely high. Lightbits can sustain such loads without needing to scale the storage infrastructure simply to meet performance demands.

The following are the key benefits of integrating Lightbits with Kubernetes:

1. High Performance: Lightbits leverages NVMe over TCP to deliver high-speed, low-latency storage, which is ideal for performance-critical applications such as databases and real-time analytics running in Kubernetes. This ensures that applications can maximize their performance without the traditional bottlenecks associated with networked storage.

2. Simplified Scalability: With Lightbits, Kubernetes environments can easily scale storage independently of compute resources. This decoupling allows for more flexible resource management, where storage can be added or adjusted without impacting the compute nodes. Such scalability is crucial for dynamic containerized environments - where workloads can change rapidly.

3. Enhanced Data Durability: Lightbits provides built-in data protection and resilience mechanisms that enhance the durability of data in Kubernetes clusters. This is particularly valuable in cloud-native environments where data integrity and availability are paramount. Lightbits' advanced features like thin provisioning, snapshots, encryption, and replication help ensure that data is protected against failures and is easily recoverable.

4. Cost Efficiency: By optimizing data storage with compression, Lightbits helps reduce the overall storage footprint and costs. This efficiency makes it possible to manage more data with less storage, which is economically beneficial for growing organizations looking to optimize their IT investments.

5. Ease of Integration: Lightbits integrates with Kubernetes through the Container Storage Interface (CSI), making it straightforward to add Lightbits storage solutions to existing Kubernetes clusters. This seamless integration simplifies the deployment and management of storage resources, allowing IT teams to focus on deploying and scaling applications rather than managing underlying storage infrastructure. Furthermore, Lightbits provides

multi-tenancy to enable Kubernetes administrators to efficiently manage storage access, further enhancing the operational efficiency and security of the cluster.

6. Cloud-Native Compatibility: Lightbits is designed to support cloud-native applications with its software-defined approach that aligns well with the principles of microservices and containerization used in Kubernetes. This compatibility ensures that storage services provided by Lightbits are as agile and portable as the containerized applications they support.

By integrating Lightbits into Kubernetes, organizations can create a high-performance, scalable, and cost-effective storage environment that supports the advanced needs of modern, data-driven applications. This integration not only enhances the operational efficiency of storage management, but also improves the overall performance and reliability of applications deployed in Kubernetes clusters.

# 5. Reference Architecture

## 5.1. High-Level Architecture

In the ever-changing realm of technology, the need for scalable, reliable, and high-performance storage solutions is more critical than ever, particularly in environments leveraging containerized applications managed by Kubernetes. To address this need, we propose a reference architecture that integrates both Lightbits and Ceph storage solutions with Kubernetes. This architecture aims to harness the strengths of each storage technology, providing a versatile and robust infrastructure optimized for a variety of application demands.

Kubernetes serves as the orchestration layer, managing containerized applications across a cluster of servers. Its dynamic nature requires a flexible approach to storage that can adapt to changing conditions and workloads. In this context, Ceph offers a resilient and scalable storage solution that is ideal for data-intensive applications requiring traditional file and object storage capabilities. It provides high availability and data durability through its self-healing and self-managing features, making it suitable for applications that need reliable, long-term data retention.

Concurrently, Lightbits introduces a high-performance, software-defined storage layer optimized for NVMe/TCP, delivering low-latency and high-throughput block storage capabilities. This is particularly beneficial for stateful applications and databases within Kubernetes that demand fast storage operations to meet their performance and efficiency targets. Lightbits' integration

through the Container Storage Interface (CSI) ensures seamless interaction with Kubernetes, allowing for dynamic provisioning and management of storage resources.

The following diagram illustrates how Kubernetes can effectively integrate with both Lightbits and Ceph, creating a unified storage solution that supports the diverse needs of modern applications. The diagram below illustrates a typical high-level deployment scenario involving Kubernetes, Lightbits, and Ceph, showcasing how these technologies can work together in a scalable architecture. Note that this is a conceptual representation and was not used in the performance validation.

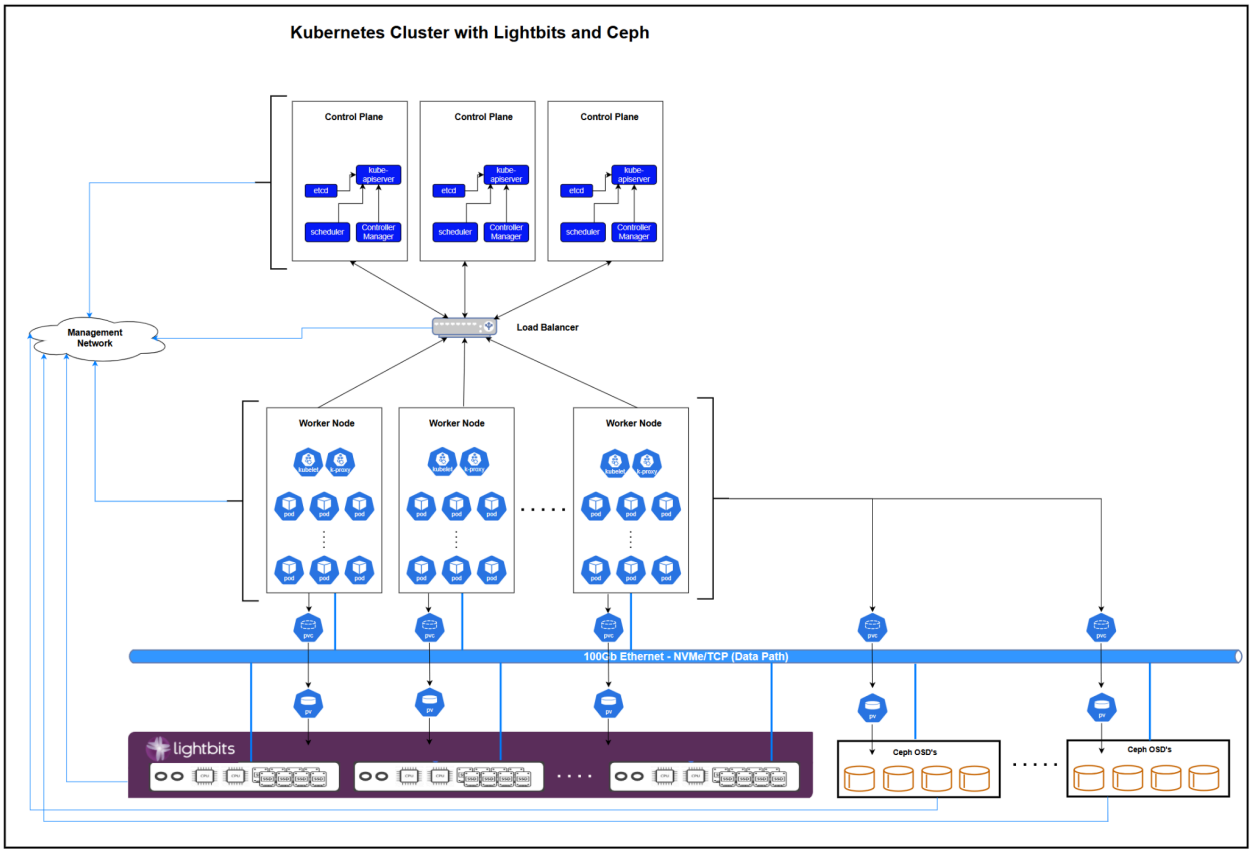



**Figure 1: High-Level Architecture**

# 5.2. Detailed Design

## 5.2.1. Hardware Setup and Configuration for Validation

After presenting the high-level architecture and its integration with Kubernetes, Lightbits, and Ceph, the focus now turns to the specific hardware and software components used during the validation. This section details the server specifications, storage configurations, and supporting software elements that comprised the tested environment, offering insight into a representative deployment optimized for performance and reliability.

For the validation, a total of six physical servers were used—three dedicated to running Lightbits storage nodes, and three configured as Kubernetes control-plane and worker nodes.

The tables below summarize the hardware and software components involved in the validation effort.

**Table 1: Hardware Specifications for Lightbits Cluster**

| Component | Description |
|---|---|
| Purpose | Lightbits targets (3 Servers) |
| Chassis Type | SuperMicro - SYS-222BT-HNR |
| CPU | 2 x Intel(R) Xeon(R) 6710E  per server |
| Memory | 512GB 5600 MT/s per server |
| MGMT NIC | On-board 1GbE per server |
| Data NIC | 1 x BCM57508 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet per server |
| Boot Disk | 1 x SAMSUNG MZ1L2960HCJR-00A07 960GB per server |
| NVMe SSDs | 6 x KIOXIA KCMYXRUG3T84 3.84TB (on server1) |
| NVMe SSDs | 6 x SAMSUNG MZWLO3T8HCLS-00A07 3.84TB (per server2 & server3) |
| OS | Rocky 9.4 |

**Table 2: Hardware Specifications for Kubernetes Control-Plane and Worker Nodes**

| Component | Description |
|---|---|
| Purpose | Kubernetes - 1 x control-plane/worker node and 2 x worker nodes (3 Servers) |
| Chassis Type | SuperMicro - SYS-222BT-HNR per server |
| CPU | 2 x Intel(R) Xeon(R) 6710E per server |
| Memory | 512GB 5600 MT/s per server |
| MGMT NIC | On-board 1GbE per server |
| Data NIC | 1 x BCM57508 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet per server |
| Boot Disk | 1 x SAMSUNG MZ1L2960HCJR-00A07 960GB per server |
| NVMe SSDs | 6 x SAMSUNG MZWLO3T8HCLS-00A07 3.84TB (Not used for storage) |
| OS | Rocky 9.4 |

**Table 3: Software Components**

| Software | Version |
|---|---|
| Kubernetes | 1.32 |
| Lightbits | 3.14.1 |
| Ceph | 17.2.8 (Quincy) |

## 5.2.2. Cluster Size and Topology Guidelines

The configuration of cluster size and topology is critical for achieving optimal performance, managing resources efficiently, and ensuring high availability and fault tolerance within the infrastructure. This section examines the cluster configurations for Kubernetes, Lightbits, and Ceph - detailing how each is designed to meet specific operational requirements.

- Kubernetes Cluster: The size and topology of the Kubernetes cluster are determined based on the expected load and redundancy requirements. Typically, a Kubernetes cluster in production environments should have at least three master nodes to ensure high availability, and multiple worker nodes to handle varying workloads. The distribution of these nodes can be aligned to provide resilience against hardware and network failures.

- Lightbits Cluster: For the Lightbits cluster, it is recommended to have a minimum of four nodes to ensure robust performance and high availability. Each node should run a stable and secure Linux distribution, such as RHEL 8+, AlmaLinux 8+, or Rocky Linux 8+. The network configuration is crucial for maintaining high throughput and resilience:

    - Network Interface Cards (NICs): Each node should be equipped with 2 x dual-port 100GbE NICs if configuring the cluster with dual instances. For a single instance configuration, 2 x single-port 100GbE NICs are sufficient. This setup ensures ample bandwidth and network redundancy, crucial for handling high data volumes and providing continuous service without disruptions.

    - Bonding Configuration: Network bonding is highly recommended, as it enhances the fault tolerance and increases the availability of the network interface by bonding two or more network interfaces together. This is particularly important in high-availability (HA) environments where network reliability is paramount but not a requirement.

  The table below shows the minimum requirements for the Lightbits storage nodes.

**Table 4: Minimum Lightbits Hardware Specifications**

| Lightbits Cluster Servers (x4)<br>2U SuperMicro or OEM equivalent | |
|---|---|
| Component | Description |
| CPU | Recommended dual-socket with 32 cores/socket with hyper-threading (high clock speed preferred). |
| Memory | Minimum 256GB per socket (scale with total NVMe capacity). |
| Data NIC | Recommended 2 x dual-port 100GbE per node(bonded) for NVMe/TCP data path for network redundancy. |
| Mgmt NIC | 1 x 1GbE or 10GbE for management and monitoring. |

| | |
|---|---|
| NVMe SSD | Minimum of 6 NVMe SSDs per node (or per socket with dual-socket configuration), to enable Erasure Coding. |
| Operating System | RHEL 8+, AlmaLinux 8+, or Rocky Linux 8+ |
| Power Supply | Redundant power supplies. |

Implementing these specifications for the Lightbits cluster optimizes storage performance and aligns with best practices for enterprise-level deployments, ensuring that the storage infrastructure is both scalable and resilient.

- Ceph Storage Cluster: In the case of Ceph, the cluster size and topology should consider factors like data replication and fault domains. Ceph clusters are typically configured with multiple replicas (usually three) to ensure data durability and availability. The topology should also consider the separation of Ceph monitors and OSDs (Object Storage Daemons) across different physical machines or racks to safeguard against single points of failure. More detailed Ceph documentation can be found here.

By carefully planning and configuring the size and topology of these clusters, the architecture can leverage the strengths of Kubernetes, Lightbits, and Ceph to create a cohesive, resilient, and scalable environment. This setup supports the current demands and is adaptable to future growth and changes in application requirements.

## 5.2.3. Network Configuration and Optimization

Achieving high performance and resilience in your storage network is crucial for ensuring optimal data throughput and reliability within your Kubernetes environment. This section provides a detailed look at the recommended network architecture for integrating Lightbits on Intel-based servers and explaining the concept of a Lightbits instance as it relates to socket configuration.

### 5.2.3.1. Lightbits Network Design

**Intel-Based Servers:**

- Configuration Options: Lightbits supports both single-instance and dual-instance setups on Intel-based processors.
    - Single Instance: A single Lightbits instance can utilize either a single-port or dual-port 100GbE NIC, with bonding recommended to enhance network resilience.

    - Dual Instances: For dual-instance setups, it is advisable to use dual-port 100GbE NICs for each instance, ensuring that each socket has dedicated network resources. This leverages CPU and memory locality effectively - especially for NVMe/TCP traffic, optimizing performance.

- Network Resilience: In dual-instance configurations, ensure that each instance (or socket) has its network interfaces bonded for redundancy and load balancing, ideally placing each bonded interface on a separate subnet to maximize fault tolerance and network path efficiency.

#### 5.2.3.2. Practical Guidelines

- Bonding Practices: Whether you're using single or dual-port NICs, configure NIC bonds using LACP or ALB to improve resiliency and performance. This setup helps manage traffic more effectively across the network interfaces.

- Subnet Configuration: Placing each bonded NIC on a separate subnet helps in reducing congestion and enhances performance consistency across the network.

The figure below shows an example of a Lightbits storage server with dual-socket configuration on an Intel-based server.

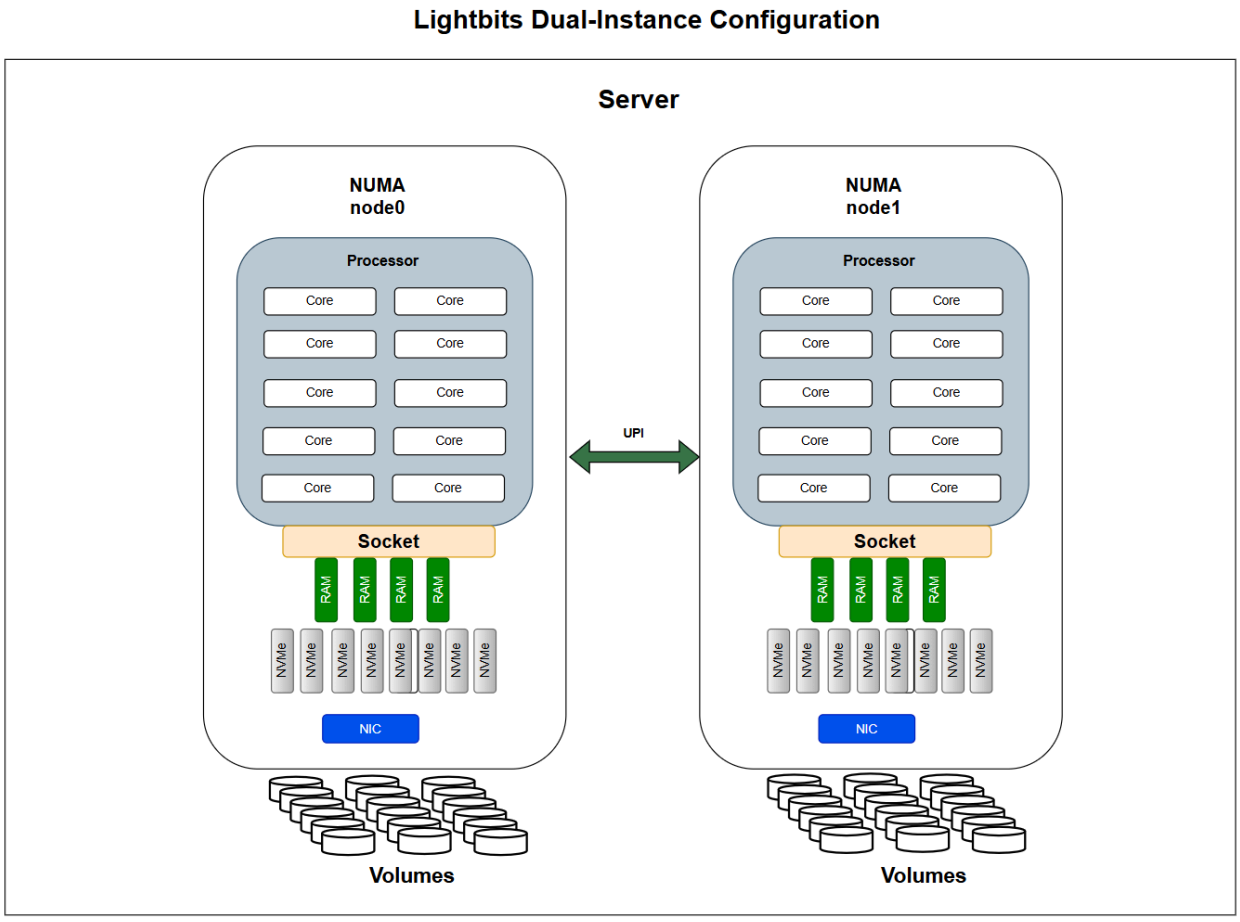



**Figure 2: Dual-Instance Architecture**

The diagram below illustrates a high-level view of the Kubernetes worker nodes integrated with a Lightbits storage cluster over a high-speed NVMe/TCP data network. Note that the diagram shows one bonding option.

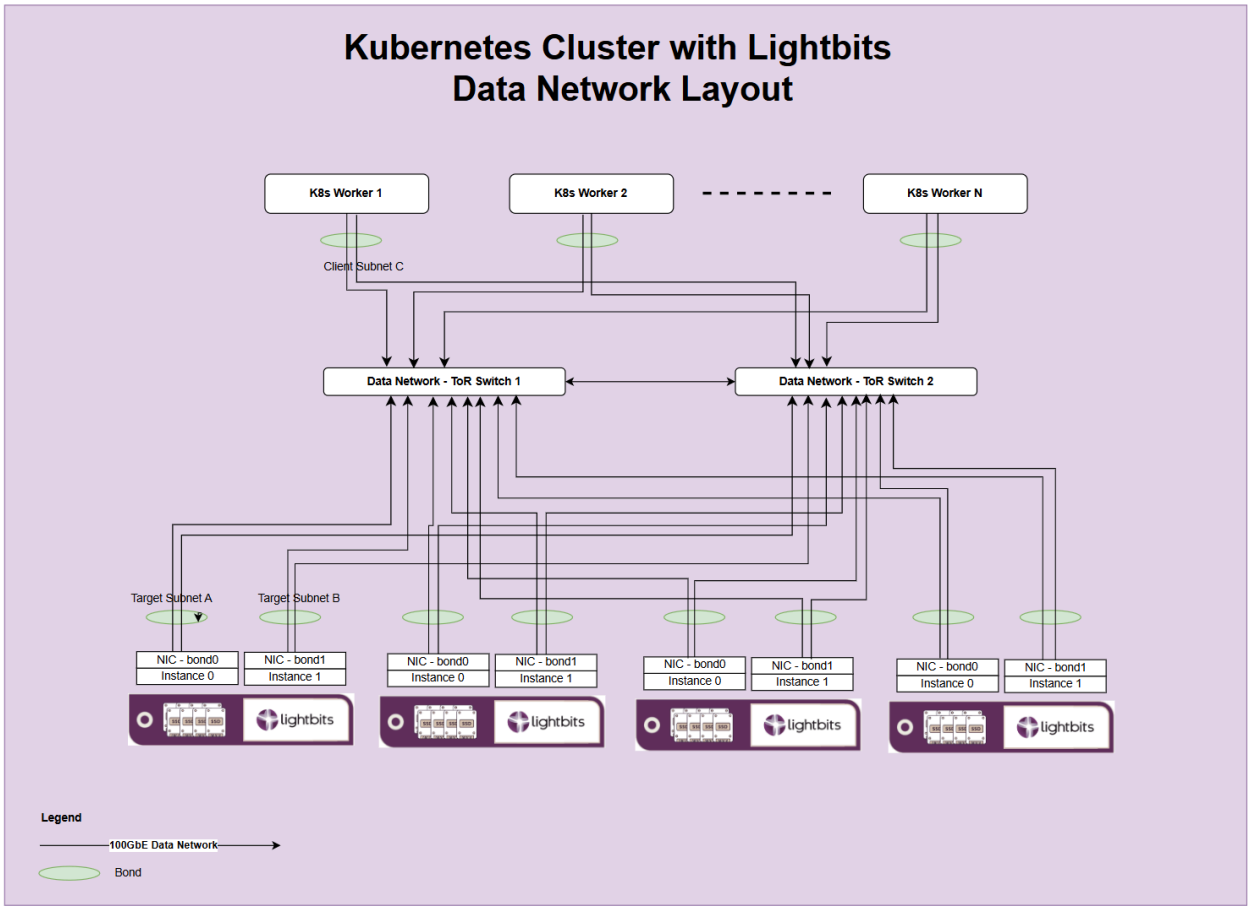



**Figure 3: Lightbits storage cluster networking layout**

# 5.3. Deployment and Configuration

With a comprehensive understanding of the architectural design and specific configurations of our Kubernetes environment integrated with Lightbits and Ceph, we now shift our focus to the practical aspects of deployment. The following sections will guide you through a detailed, step-by-step process for setting up and configuring Kubernetes, Lightbits, and Ceph. This systematic approach ensures that each component is optimally configured to function cohesively within the broader infrastructure - thereby maximizing performance, reliability, and scalability.

## 5.3.1. Kubernetes Deployment

### 5.3.1.1. Installing Kubernetes

Deploying Kubernetes on-premise - particularly on bare-metal servers - requires a thorough approach tailored to the specific needs of your infrastructure. This section briefly focuses on the manual installation method, which is particularly suited for environments where customization and control over the installation process are paramount. We will specifically address setting up Kubernetes on AlmaLinux, a robust and secure choice for enterprise environments that demand stability and extensive support. More information on how to install Kubernetes can be found here.

1. Begin by setting up the bare-metal servers with an operating system that is compatible and well-suited for your Kubernetes environment. Recommended choices include Ubuntu, Red Hat Enterprise Linux (RHEL), or AlmaLinux. These operating systems offer robust support and compatibility with Kubernetes. Ensure that you select the same operating system across all control plane and worker nodes to simplify management and reduce compatibility issues. Follow the manufacturer's guidelines for a clean installation, configuring necessary network settings and security patches during setup to lay a strong foundation for your Kubernetes cluster.
2. Once the operating system is installed, it's important to ensure that all system packages are up to date by using the “update” command.. This will provide the latest security patches and performance improvements available, making your Kubernetes nodes more secure and efficient. Furthermore, ensure that kernel settings are configured correctly for Kubernetes by referring to the Kubernetes documentation for more details.
3. After ensuring that your AlmaLinux system is up to date, proceed with the installation of Docker, which provides a user-friendly interface for building and managing container images.
4. Following the Docker installation, install containerd.io, which is a critical component of the container runtime environment used by Kubernetes.
5. After setting up the container runtime, continue by installing the essential Kubernetes components: kubeadm, kubelet, and kubectl.
    - kubeadm: This tool helps you bootstrap the Kubernetes cluster efficiently, handling the complexities of setting up the initial cluster and managing its lifecycle.
    - kubelet: Running on all cluster nodes, kubelet is the primary "node agent" that ensures that each container is running in a pod.
    - kubectl: This command-line tool lets you interact with your Kubernetes cluster, providing powerful control over its resources and deployments.

6. Once the installation of Docker, containerd, and Kubernetes components is complete, the next step is to ensure that these services are enabled to start on boot and are actively running on your system. Use the systemctl command to manage these services effectively.
7. To establish a resilient and highly available Kubernetes cluster, begin by initializing the cluster on one of the control plane nodes. This is done using the kubeadm init command, which sets up the foundational components of the cluster.

   For a high-availability configuration, specify the HA proxy IP address as the --control-plane-endpoint. This ensures that the cluster management is accessible and resilient to individual node failures. Use the following command format to initialize your cluster:

   ```
   sudo kubeadm init --control-plane-endpoint="HA_PROXY_IP:PORT" --upload-certs
   ```
8. Once the Kubernetes cluster initialization is complete, proceed with establishing the Calico network or another type such as Flannel to handle pod networking across the cluster. This is achieved by applying a Calico configuration file using the kubectl command. Run the following command on your first control plane node to deploy Calico:

   ```
   sudo kubectl apply -f <calico-config-file.yaml>
   ```

   After setting up the Calico network on your control plane node, it's essential to confirm that the Calico pods are properly running within the kube-system namespace. To check the status of these pods, use the following command:

   ```
   sudo kubectl get pods -n kube-system
   ```

   This will list all pods in the kube-system namespace, allowing you to review the READY and STATUS columns for each Calico pod. Ensure that the READY column indicates that all necessary containers within the pods are up and running, and that the STATUS column shows Running.
9. After initializing the cluster and setting up the Calico network, the next step is to expand your cluster by adding additional control plane nodes and worker nodes. This is crucial for enhancing the resilience and scalability of your Kubernetes environment.
   - Joining Additional Control Plane Nodes: To integrate additional control plane nodes, use the kubeadm join command provided at the end of the initial cluster initialization. This command includes a token and a certificate key necessary for securely joining the node to the cluster as a control plane node. Execute the command on each additional control plane node:

     ```
     sudo kubeadm join <control-plane-endpoint>:<port> --token <your-token> \
     ```

```
--discovery-token-ca-cert-hash sha256:<hash> \ --control-plane --certificate-key
<certificate-key>
```

- Joining Worker Nodes: Similarly, to add worker nodes to the cluster, use the kubeadm join command tailored for worker nodes, which does not include the --control-plane and --certificate-key options. This command should also be provided at the end of the initial setup:

```
sudo kubeadm join <control-plane-endpoint>:<port> --token <your-token> \
--discovery-token-ca-cert-hash sha256:<hash>
```

10. Once all control plane and worker nodes have been successfully joined to the cluster, it's important to confirm that the cluster is functioning correctly. To verify that all nodes are online and correctly integrated into the cluster, execute the following command on any of the control plane nodes:

```
sudo kubectl get nodes
```

This command will display a list of all nodes in the cluster - along with their status, roles, and readiness. Ensure that each node's status is Ready, indicating that it is configured correctly and actively participating in the cluster operations.

## 5.3.2. Lightbits Deployment

### 5.3.2.1. Installing Lightbits SDS

Following the successful deployment of Kubernetes, the next step involves integrating a robust storage solution to support your containerized applications. Lightbits SDS offers a high-performance, software-defined storage system optimized for environments requiring fast access and efficient data management. This section will focus on the manual installation of Lightbits SDS, which is particularly beneficial for on-premise, bare-metal deployments where precise configuration and control are essential. We will outline the steps to integrate Lightbits within your existing Kubernetes setup on AlmaLinux, ensuring that your storage infrastructure is as resilient and scalable as your computational resources.

Lightbits SDS can be deployed using either an online or offline installation method, depending on your network environment and accessibility:

- Online Method: This approach requires that the Lightbits servers have internet access. During installation, the software is directly downloaded from online repositories. This method is convenient if your servers are connected to the internet, as it ensures that you receive the latest Linux packages and the Lightbits software.

- Offline Method: For environments without direct internet access, the offline method is ideal. Prior to installation, the necessary software packages are manually downloaded and transferred to the servers. This method ensures that you can install Lightbits SDS without requiring an internet connection, making it suitable for secure or isolated networks.

Both methods offer flexibility in how Lightbits SDS can be integrated into your infrastructure, allowing you to choose the best option based on your specific operational requirements. In this reference architecture, we will primarily focus on a high-level overview of the online installation method. For detailed installation instructions, refer to the Lightbits website.

### 5.3.2.2. Prerequisites

**Ansible Host**

The installation of Lightbits is facilitated through Ansible, which simplifies the deployment process. To proceed, you will need a separate Linux server or a client machine that has Ansible capabilities. Lightbits offers a convenient solution by providing Ansible within a Docker container, which minimizes the setup requirements on the client machine.

To simplify the installation process further, follow these steps:

1. Install Docker: Ensure that docker-ce (Docker Community Edition) is installed on the client machine. This will be used to run the Lightbits Ansible container. The below packages are required for Docker. Note that existing Docker packages that come with the operating system installation should be removed first.
    a. Docker-ce
    b. Docker-ce-cli
    c. containerd.io
    d. Docker-compose-plugin
2. Download Ansible Container: After installing Docker, log in to Lightbits Docker Hub and download the Ansible container. You will need a token to access the Docker Hub, which will be provided by Lightbits. Select the version that matches the Lightbits software version. More information can be found here.
3. Run Lightbits Ansible Container: After pulling down the Ansible container - which contains all the necessary scripts and tools to install Lightbits on your target servers - you will be ready to proceed. Detailed steps for utilizing this container to install Lightbits will be covered in the “Running Ansible” section.

This approach not only streamlines the installation but also ensures that the environment is consistent and controlled, reducing potential issues related to software dependencies.

**Configuration Requirements for Lightbits Target Servers**

To prepare the Lightbits target servers for operation, several key settings must be configured to ensure optimal performance and connectivity:

1. Operating System:
    - Install one of the following Linux distributions on the Lightbits target servers: Alma Linux, Red Hat Enterprise Linux, or Rocky Linux.
2. Network Configuration:
    - Management IP Address: Assign a management IP address for administrative access.
    - Data IP Address and Interface Bonding: Configure at least two dual-port 100Gb Ethernet interfaces, one for each NUMA node in a dual-socket server. Each dual-port network interface should be bonded for enhanced reliability and performance.

      Note: Bonding is not a requirement, but is recommended for network redundancy.
    - Link Aggregation Control Protocol (LACP): It is recommended to use LACP for network interface bonding with the following settings:
        - Mode: 802.3ad - This setting allows for the aggregation of multiple network connections in parallel, to increase bandwidth and provide redundancy.
        - Transmit Hash Policy: layer3+4 - This policy helps in load balancing outbound traffic.
        - MII Monitor Interval: 100 milliseconds - This setting specifies the frequency at which MII link monitoring occurs, aiding in the detection of network failures.
3. Firewall Configuration:
    - Start the Firewalld Service: Ensure that the firewalld service is active to manage network traffic securely.
    - Install iptables: Confirm that iptables is installed for additional network packet filtering and security measures.

**Ports Required for Installation**

TCP port 22 must be open between the Ansible host (client machine) and the Lightbits target servers, to facilitate the installation process.

| Component | Management/Data NIC | Port (TCP) |
|---|---|---|
| Ansible operation over SSH | Management | 22 |

**Ports Required for Operation**

The following TCP port must be open on the Lightbits target servers to ensure that the Lightbits software installs and functions correctly.

| Component | Management/Data NIC | Port (TCP) | Default Location |
|---|---|---|---|
| Management CLI | Management | 443 | None |
| ETCD peer port | Data | 2380 | roles/etcd/defaults/main.yml |
| Exporter port | Management | 8090 | roles/install-lightos/defaults/main.yml |
| Duroslight port. Note: NVMe client connects to Duroslight via this port. | Data | 4420, 8009 | roles/install-lightos/defaults/main.yml |
| Replicator port. Note: Other nodes connect for replication to the node via this port. | Data | 22226 | roles/install-lightos/defaults/main.yml |
| Encryption at rest port | Data | 4007 | |

## 5.3.2.3. Ansible Setup for Installation

The following instructions detail the steps to deploy the Ansible container and configure the installation files necessary for setting up Lightbits.

1. On the Ansible host, as the root user, extract the tarball: light-app-install-environment-<version>.tgz.
2. Go to ansible/inventories/ and you will find a folder called cluster_example. You can make a copy of this folder and give it a name. For this installation example, we'll make a copy of this folder and name it cluster_1.

3. Go into cluster_1.
4. Edit the hosts file.
    a. The top section lists the Lightbits target servers and their credentials.
        - server00: Replace "server00" with the actual hostname of the machine.
        - ansible_host: Specify the Fully Qualified Domain Name (FQDN) or management IP address of the machine.
        - ansible_ssh_user: Enter the username for SSH access.
        - ansible_ssh_pass: Provide the password for the SSH user.
        - ansible_become_user: Indicate the user account that has privileges to execute commands requiring elevated rights.
        - ansible_become_pass: Enter the password for the privileged user account.
    b. duros_node section
        - Remove any unneeded lines; this section should only contain hostnames that correspond to the entries in the top section.
    c. duros_nodes:vars section
        - Change the URL of the local_repo_base_url to reflect the correct URL for the Lightbits version.
    d. etcd section
        - Ensure that this section only contains entries that match the hostnames specified in the top section of the file.
    e. Initiators section
        - Remove the entry labeled client_0.
    f. Save and exit.

The hosts file should look something like this:

```
lbit01    ansible_host=10.10.10.1    ansible_connection=ssh ansible_ssh_user=root  ansible_ssh_pass=passwd
ansible_become_user=root ansible_become_pass=passwd
lbit02    ansible_host=10.10.10.2    ansible_connection=ssh ansible_ssh_user=root  ansible_ssh_pass=passwd
ansible_become_user=root ansible_become_pass=passwd
lbit03    ansible_host=10.10.10.3    ansible_connection=ssh ansible_ssh_user=root  ansible_ssh_pass=passwd
ansible_become_user=root ansible_become_pass=passwd
lbit04    ansible_host=10.10.10.4    ansible_connection=ssh ansible_ssh_user=root  ansible_ssh_pass=passwd
ansible_become_user=root ansible_become_pass=passwd

[duros_nodes]
lbit01
lbit02
lbit03
lbit04

[duros_nodes:vars]
local_repo_base_url=https://dl.lightbitslabs.com/<token>/lightos-3-12-1-rhl-9/rpm/el/9/$basearch
auto_reboot=true
cluster_identifier=ae7bdeef-897e-4c5b-abef-20234abf21bf

[etcd]
```

```
lbit01
lbit02
lbit03
lbit04

[initiators]
```

5. Edit the group_vars/all.yml file to ensure that enable_iptables is set to false, and persistent_memory is also set to false. It should look something like this:

```
---

enable_iptables: false
persistent_memory: false
start_discovery_service_retries: 5
#nvme_subsystem_nqn_suffix: "some_suffix"
chrony_enabled: true
```

6. Within the host_vars directory, you will find sample configuration files named server00.yml, server01.yml, and server02.yml. Follow these steps to properly configure each server:
   a. Rename the sample files:
      - Rename these files to reflect specific server identifiers for clarity, such as lbit01.yml, lbit02.yml, etc. This naming convention helps in easily associating the configuration files with their respective servers.
   b. Edit the configuration files:
      - Open each renamed file and update the following parameters to suit your deployment needs:
        1. data_ip: Specify the data IP address used by the server.
        2. initialDeviceCount: Set the initial number of NVMe storage devices.
        3. maxDeviceCount: Define the maximum number of NVMe storage devices that can be configured.
        4. allowCrossNumaDevices: Indicate whether or not devices across NUMA nodes are permissible. This should be set to false if configuring two NUMA nodes.

Below is a sample configuration for the lbit01.yml file, set up with two NUMA nodes:

```
---
name: lbit01
nodes:
-  instanceID: 0
   data_ip: 20.20.20.1
   failure_domains:
   - lbit01
   ec_enabled: true
```

```
  lightfieldMode: SW_LF
  storageDeviceLayout:
   initialDeviceCount: 12
   maxDeviceCount: 12
   allowCrossNumaDevices: false
   deviceMatchers:
#    - model =~ ".*"
     - partition == false
     - size >= gib(300)
#    - name =~ "nvme0n1"
-  instanceID: 1
  data_ip: 20.20.21.1
  failure_domains:
  - lbit01
  ec_enabled: true
  lightfieldMode: SW_LF
  storageDeviceLayout:
   initialDeviceCount: 12
   maxDeviceCount: 12
   allowCrossNumaDevices: false
   deviceMatchers:
#    - model =~ ".*"
     - partition == false
     - size >= gib(300)
#    - name =~ "nvme0n1"
```

### 5.3.2.4. Running Ansible

At this stage, you are ready to begin the Ansible deployment.

1. Navigate back to the base directory where you extracted the light-app-install-environment-<version>.tgz file. Within this directory, you will find several subdirectories, including:
   - Ansible
   - Playbooks
   - Plugins
   - Roles
2. You need to create a directory named lightos-certificates. This directory is crucial as it will house the self-signed certificates generated during the installation of the Lightbits cluster.

   It should look something like this:

```
drwxr-xr-x.  3 root root     25 Nov 25 13:55 ansible
-rw-r--r--.  1 root root  11032 Jul 24 00:18 ansible.cfg
-rw-r--r--.  1 root root  64336 Nov 22 04:24 light-app-install-environment-v3.12.1~b114.tgz
drwxrwxrwx.  2 root root   4096 Nov 26 10:46 lightos-certificates
drwxr-xr-x.  2 root root   4096 Nov 22 02:43 playbooks
drwxr-xr-x.  3 root root     38 Jul 24 00:18 plugins
drwxr-xr-x. 11 root root    181 Nov 22 02:43 roles
```

3. Use the docker image ls command to list available Docker images and identify the image name needed for the docker run command.

   It should look something like this:

```
REPOSITORY                                                    TAG     IMAGE ID      CREATED      SIZE
docker.lightbitslabs.com/lightos-3-12-1-rhl-9/lb-ansible   v9.1.0   4de44e36bed2   5 weeks ago
    697MB
```

4. Run the below command in a tmux session. Note the Docker image in the command line:

```
docker run -it --rm --net=host \
  -e UID=`id -u` \
  -e GID=`id -g` \
  -e UNAME=$USER \
  -v `pwd`:/ansible \
  -w /ansible \
  -e ANSIBLE_LOG_PATH=/ansible/ansible.log \
  docker.lightbitslabs.com/lightos-3-12-1-rhl-9/lb-ansible:v9.1.0 \
  sh -c 'ansible-playbook \
    -e system_jwt_path=/ansible/lightos_jwt \
    -e certificates_directory=/ansible/lightos-certificates \
    -i ansible/inventories/cluster_1/hosts \
    playbooks/deploy-lightos.yml -vv'
```

## 5.3.3. Lightbits Container Storage Interface (CSI) Deployment

### 5.3.3.1. Installing Lightbits CSI

With the Lightbits software successfully installed on your target servers, the next crucial step involves integrating the Lightbits Container Storage Interface (CSI). The Lightbits CSI is essential for facilitating communication between your Kubernetes cluster and the Lightbits storage solution, enabling advanced storage features directly from within the Kubernetes environment. This interface enhances how storage resources are managed and utilized, ensuring optimal performance and scalability of your applications. In the following section, we will guide you through the detailed process of installing and configuring the Lightbits CSI, ensuring seamless integration with your newly established storage infrastructure.

The following instructions outline the steps required to deploy the Lightbits CSI for Kubernetes.

1. Download the latest Lightbits CSI driver from the Lightbits public repository and extract the file.

For example:

On one of the control plane nodes, run:

```
# curl -1sLf -O 'https://dl.lightbitslabs.com/public/lightos-csi/raw/files/lb-csi-bundle-1.17.0.125.tar.gz'

# tar xzf lb-csi-bundle-1.17.0.125.tar.gz
```

2. Install the plugin with the discovery client. This plugin is in the “k8s” directory.

For example:

```
# cd k8s

# kubectl create -f lb-csi-plugin-k8s-v1.30-dc.yaml

– To verify

# kubectl get -n kube-system statefulset lb-csi-controller ; echo ; kubectl get -n kube-system daemonsets lb-csi-node ; echo ; kubectl get -n kube-system pod --selector app=lb-csi-plugin -o wide
NAME              READY   AGE
lb-csi-controller   1/1     4m37s

NAME         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
lb-csi-node   2         2         2       2            2           <none>          4m37s

NAME                 READY   STATUS    RESTARTS   AGE     IP         NODE              NOMINATED NODE   READINESS GATES
lb-csi-controller-0   5/5     Running   0          4m37s   x.x.x.x    rack03-server59   <none>           <none>
lb-csi-node-c28jd     3/3     Running   0          4m37s   x.x.x.x    rack03-server59   <none>           <none>
lb-csi-node-qdb5b     3/3     Running   0          4m37s   x.x.x.x    rack03-server58   <none>           <none>
```

3. Install the secret and the storage class. You will first need to convert the JWT strings to the base64 format and enter it in the secret section at the “jwt” parameter. In addition to the JWT strings, you will need to enter the mgmt-endpoint IP address and port 443, replica-count, and compression. Note that the IP addresses are your Kubernetes control plane management IP addresses. The YAML file below shows an example of the secret and the storage class. More information on installing the Lightbits secret and storage class can be found here.

```
---
```

```
# Source: lb-csi-workload-examples/charts/storageclass/templates/secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: lightbits-secret
  namespace: default
  labels:
    helm.sh/chart: "storageclass-0.1.0"
    app.kubernetes.io/instance: "RELEASE-NAME"
    app.kubernetes.io/version: ""
    app.kubernetes.io/managed-by: "Helm"
type: lightbitslabs.com/jwt
data:
  jwt: |-
     ZXlKaGJHY2lPaUpTVXpJMU5pSXNJbXRwWkNJNkluTjVjM1JsYlRweWIyOTBJaXdpZEhsd0lqb2lT
---
# Source: lb-csi-workload-examples/charts/storageclass/templates/storageclass.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: lightbits-sc
provisioner: csi.lightbitslabs.com
allowVolumeExpansion: true
parameters:
  mgmt-endpoint: X.X.X.X:443,Y.Y.Y.Y:443,Z.Z.Z.Z:443
  replica-count: "3"
  compression: enabled
  project-name: default
  mgmt-scheme: grpcs
  csi.storage.k8s.io/controller-publish-secret-name: lightbits-secret
  csi.storage.k8s.io/controller-publish-secret-namespace: default
  csi.storage.k8s.io/controller-expand-secret-name: lightbits-secret
  csi.storage.k8s.io/controller-expand-secret-namespace: default
  csi.storage.k8s.io/node-publish-secret-name: lightbits-secret
  csi.storage.k8s.io/node-publish-secret-namespace: default
  csi.storage.k8s.io/node-stage-secret-name: lightbits-secret
  csi.storage.k8s.io/node-stage-secret-namespace: default
  csi.storage.k8s.io/provisioner-secret-name: lightbits-secret
  csi.storage.k8s.io/provisioner-secret-namespace: default
```

```
– Install the secret and storage class

# kubectl create -f secret-and-storage-class.yaml

– To verify

# kubectl get secret,sc
NAME                  TYPE                  DATA  AGE
secret/example-secret  lightbitslabs.com/jwt  1     63s

NAME                                PROVISIONER          RECLAIMPOLICY  VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION  AGE
storageclass.storage.k8s.io/example-sc  csi.lightbitslabs.com  Delete       Immediate       true            63s
```

## 5.4. Seamless Integration of Lightbits

Integrating Lightbits storage solutions into an existing Kubernetes cluster can significantly enhance storage performance and scalability without the need for extensive rearchitecting. Lightbits' technology is designed to be compatible with Kubernetes through the use of its CSI plugin, which facilitates straightforward deployment and management. This process involves installing the Lightbits CSI driver into your Kubernetes environment, which then allows the cluster to communicate directly with Lightbits storage, leveraging its advanced NVMe/TCP capabilities.

The integration is designed to be non-disruptive, with Lightbits acting as an additional storage resource that can be dynamically provisioned and managed just like any other Kubernetes storage resource. This approach not only simplifies the operational complexity but also ensures that existing applications can benefit from improved performance and storage efficiency without alteration to their deployment configurations. As a result, Kubernetes administrators can deploy Lightbits rapidly, achieving immediate benefits in terms of latency, throughput, and storage density - all while maintaining the existing architecture and operational workflows.

# 6. Performance Evaluation

With Lightbits seamlessly integrated into the Kubernetes environment via its CSI plugin, the next logical step is to validate the benefits it brings to real-world workloads. To assess the impact of Lightbits on performance, a series of benchmark tests using the FIO tool with a queue depth of 32

were conducted – after preconditioning the storage volumes to ensure consistent performance – and compared against a Ceph-based (RBD) deployment using the same Kubernetes infrastructure.

These benchmarks simulate common data access patterns - including random read, random write, and mixed read/write (70/30 and 50/50) workloads - executed across a 15-pod Kubernetes cluster. Each pod was provisioned with a 1 x 500GB block storage volume from either the Lightbits or Ceph backend.

To focus on architectural advantages rather than raw numbers, results are presented in normalized form, with Ceph used as the baseline (1.0) and Lightbits shown as a relative performance multiplier.

The following charts display performance trends across:

- Random Read and Random Write workloads using multiple block sizes (4K, 8K, 32K, 64K, 128K).
- 70/30 and 50/50 Read/Write Mixed workloads that better represent real-world application behavior.
- Varying levels of concurrency (1, 8, 16, and 24 jobs).

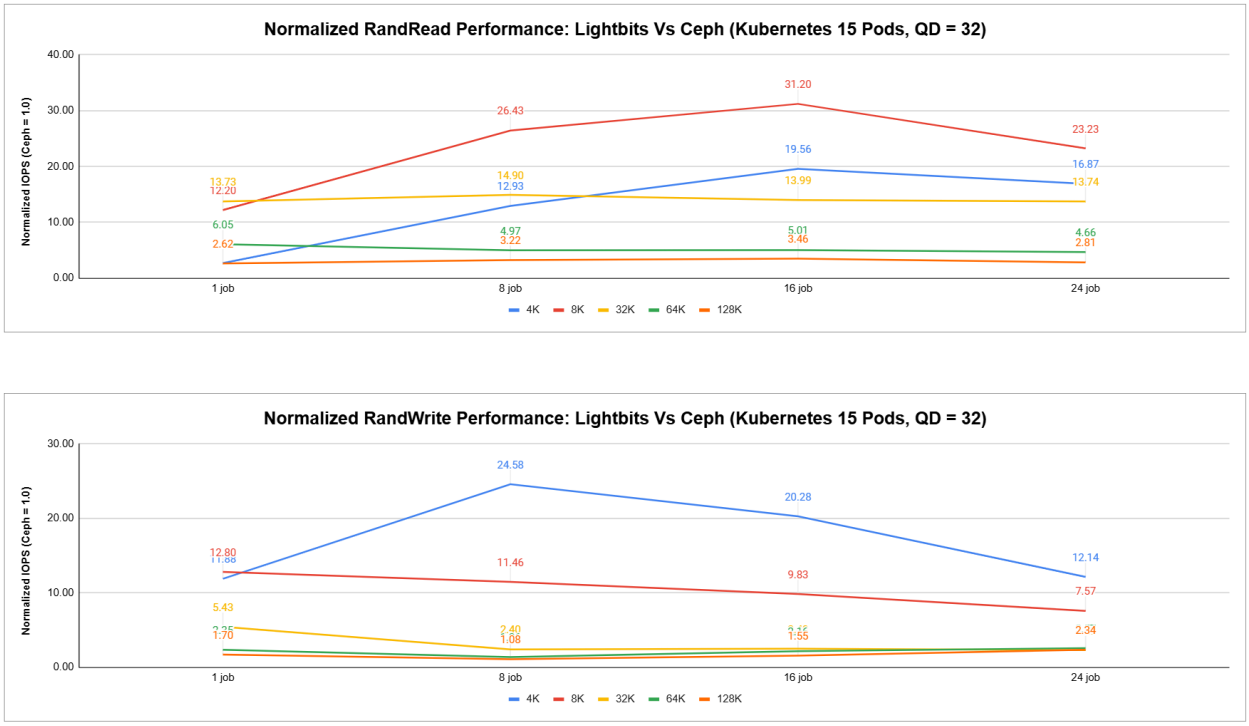
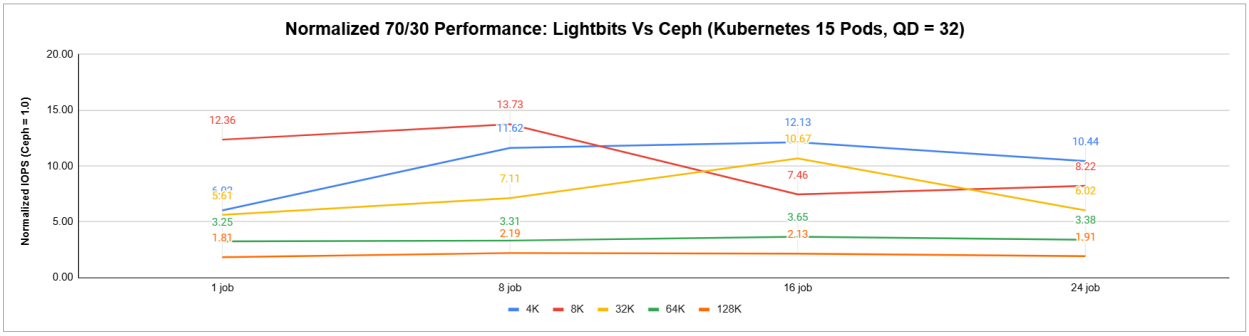



**Figure 4: Random Read Random Write Charts**



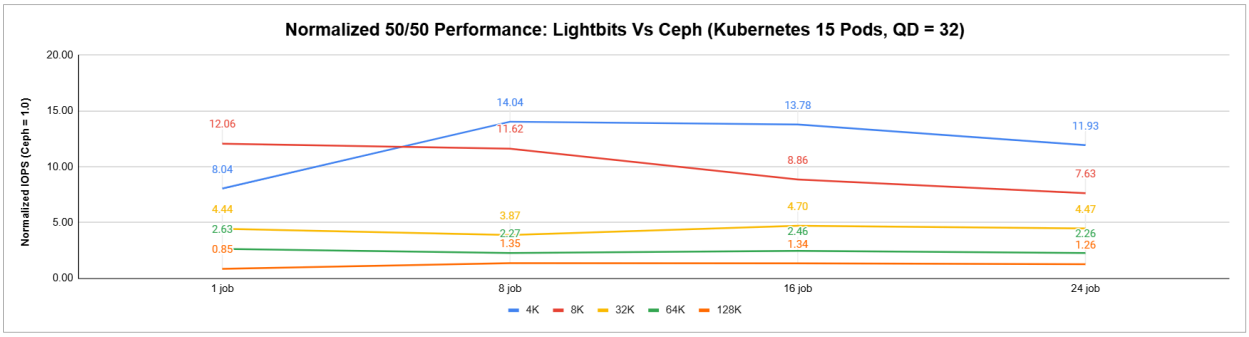




**Figure 5: 70/30 and 50/50 ReadWrite Mixed Charts**

Across all scenarios, Lightbits consistently outperforms Ceph. In random read and write benchmarks, Lightbits demonstrates performance gains ranging from 1X to over 31X compared to Ceph, particularly in high-concurrency environments with small block sizes. In the mixed workload tests (70/30 and 50/50), Lightbits also delivers strong performance, with improvements ranging from 0.8X to 14X over Ceph, highlighting its ability to efficiently handle both read- and write-intensive application demands.

These results reinforce the architectural takeaway: Lightbits is ideally suited for latency-sensitive, high-performance Kubernetes workloads, such as databases, AI/ML pipelines, and transactional applications. Meanwhile, Ceph remains a solid choice for general-purpose workloads where unified object/file/block access or lower performance demands are acceptable.

# 7. Conclusion

This paper presents a comprehensive reference architecture for deploying Kubernetes environments optimized for various workloads - from general-purpose applications to high-performance applications such as databases and AI/ML. By leveraging Ceph and Lightbits storage, this architecture ensures robust storage solutions, high performance, and cost efficiency. Organizations can follow this architecture to build highly available, scalable, and cost-effective Kubernetes platforms to meet the diverse needs of any application. Additionally, this paper provides guidance on hardware setup and configuration to optimize performance and cost.

# 8. Appendix

The following appendix provides supporting materials referenced throughout this paper. It includes sample Kubernetes StatefulSet YAML manifests and the template FIO job file.

A. Kubernetes StatefulSet YAML used for FIO Benchmarking:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: fio-block-test1
spec:
 serviceName: "fio"
 replicas: 15
 selector:
  matchLabels:
   app: fio
 template:
  metadata:
   labels:
    app: fio
  spec:
   containers:
   - name: fio
    image: almalinux:9
    command: ["/bin/sh", "-c"]
    args:
     - |
      dnf install -y epel-release &&
      dnf install -y fio &&
      dnf install -y ncurses &&
      sleep infinity
    securityContext:
     privileged: true
    volumeDevices:
```

```
    - name: data
      devicePath: /dev/xvdb
  restartPolicy: Always
volumeClaimTemplates:
- metadata:
   name: data
 spec:
   accessModes: [ "ReadWriteOnce" ]
   volumeMode: Block
   storageClassName: lb-sc
   resources:
    requests:
     storage: 500Gi
```

B. FIO Job File Template

The following FIO configuration was used for benchmarking. Adjust the bs (block size), rw (read/write mode), numjobs (parallel jobs), and iodepth (queue depth) parameters to reflect the desired workload characteristics.

```
[global]
ioengine=libaio
direct=1
prio=0
rw=randread
bs=4k
numjobs=1
iodepth=1
time_based
ramp_time=5
norandommap
randrepeat=0
runtime=120
group_reporting=1
[job]
filename=/dev/xvdb
```

# About Lightbits Labs

Lightbits Labs™ (Lightbits) is leading the digital data center transformation by making high-performance elastic block storage available to any cloud. Creators of the NVMe® over TCP (NVMe/TCP) protocol, Lightbits software-defined storage is easy to deploy at scale and delivers performance equivalent to local flash to accelerate cloud-native applications in bare metal, virtual, or containerized environments. Backed by leading enterprise investors including Cisco Investments, Dell Technologies Capital, Intel Capital, JP Morgan Chase, Lenovo, and Micron, Lightbits is on a mission to make high-performance elastic block storage simple, scalable and cost-efficient for any cloud.


🌐 www.lightbitslabs.com ✉ info@lightbitslabs.com

US Offices
1830 The Alameda,
San Jose, CA 95126, USA

Israel Office
17 Atir Yeda Street,
Kfar Saba 4464313, Israel





LBRA03/2025/06