MICROSERVICE ARCHITECTURE FOR STATEFUL APPLICATIONS

by

Andrew Edwards

May, 2024

Director of Thesis: Nic Herndon, PhD

Major Department: Computer Science

The widespread adoption of microservices in modern software development has required efficient solutions for managing databases within Kubernetes environments. While Kubernetes offers some mechanisms for handling stateful applications, challenges exist in performance and reliability, especially for time-critical workloads such as Geographic Information System (GIS) data processing. This study investigates the limitations of existing approaches and a solution leveraging Ceph as a backend storage system integrated with a Kubernetes cluster. Through the creation of an External Ceph Storage Cluster and reconfigured Helm charts, the proposed solution enhances data management for reliability, scalability, and performance. Results demonstrate improvements in data processing speed and fault tolerance, contributing to the evolution of Kubernetes-based databases. Additionally, a deployment script is provided to facilitate easy experimentation and adoption of the solution, expanding exploration and integration of advanced storage technologies within Kubernetes ecosystems.

MICROSERVICE ARCHITECTURE FOR STATEFUL APPLICATIONS

A Thesis

Presented to The Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Computer Science

by

Andrew Edwards

May, 2024

Director of Thesis: Nic Herndon, PhD

Thesis Committee Members:

Qin Ding, PhD

Rui Wu, PhD

Acknowledgements

I would like to express my sincere thanks to my advisor, Dr. Nic Herndon, as well as BAE Systems and my colleagues Jeffrey Burrell, Keith Lee, and Carmen Bentley for their invaluable guidance and support throughout this research. I would also like to thank my friends and family for their unwavering support and encouragement.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Background

In the ecosystem of modern software development, the usage of microservices architecture has become popular to facilitate scalability, resiliency, and ease of deployment. This shift towards using microservices is fueled by the need for independently deployable components that can be efficiently developed, tested, and maintained. One of the leading technologies of this architectural style is Kubernetes, a powerful container orchestration platform that facilitates the deployment, scaling, and management of containerized applications.

As organizations increasingly adopt Kubernetes, the role of databases within this ecosystem becomes essential. Kubernetes was originally designed to support stateless applications, which typically, do not store data. The now more commonly developed stateful applications require a means of storing, managing, and retrieving data essential to operations over their complete life cycles. This led to the decision of developers to host stateless services within Kubernetes, while stateful applications were hosted outside of Kubernetes. Kubernetes has evolved to support stateful applications through the use of persistent volumes and statefulsets to provide a method to store the data generated by an application. Persistent volumes are managed by Kubernetes itself and are not dependent on the pod lifecycle, allowing the data to be

reused, which addresses the gap between stateless and stateful applications.

Nevertheless, opposing viewpoints exist regarding this approach. For instance, some argue against using database containers in production [17], and others suggest that running databases in Kubernetes is not fully prepared yet [19]. There are two main methodologies proposed for managing databases within a Kubernetes environment:

1. The first method is to leverage Kubernetes' StatefulSets to manage database state, automatically provisioning persistent volumes for each pod [5]. But, this depends on external volumes for the data persistence which can lead to complications such as dangling volumes [9] and disk I/O performance bottlenecks [20]

2. The second method is to address the constraints of StatefulSets by involving the usage of third-party operators to extend Kubernetes APIs. This easily allows the creation of custom resources and controllers [16]. However, the majority of current operators are still immature and unsuitable for critical workloads [12].

1.2 The Purpose of this study

Kubernetes (K8s) poses a significant challenge for effectively managing databases within its environment. Most notably, there is a lack of a standardized and optimal solution to address the complexities associated with large-scale data storage. This issue becomes noticeable when attempting to run databases in a K8s environment for time-critical Geographic Information System (GIS) data processing, demanding near real-time and high-volume throughput.

The absence of a standard solution is rooted in the foundational nature of Kubernetes, which was never explicitly created to provide with persistent data storage

solutions directly. Google engineers have confirmed that the practical implementation of running databases in Kubernetes remains distant and suggest waiting for further evolution of these technologies and tools [19]. The orchestration complexities related to dynamic databases, combined with the demands of real-time processing, have highlighted challenges in the development of storage solutions surrounding the Kubernetes ecosystem. This complexity is why some advise against using containers for production databases as they are critical services that demand significant effort to operate reliably. By minimizing the complexity, fewer unknowns will be present in your stack, making it simpler to manage operations and incidents [17]. Existing approaches often fall short, struggling to provide efficient storage, speed, and reliability comparable to traditional bare-metal configurations. Moreover, some of these solutions are more complex and difficult to set up, further complicating a standard storage solution.

1.3 Motivation of Project

The motivation behind our project stems from the benefits of near real-time data processing combined with ensuring data security and fault tolerance. Recognizing the inherent limitations of conventional storage solutions utilizing mounts like NFS, we wanted to explore innovative approaches to database management within Kubernetes clusters. We wanted to explore the transition from these conventional storage mechanisms using deployments with persistent volumes to CephFS within our Kubernetes cluster to elevate our data management for reliability, scalability, and performance.

While our current solutions are sufficient, there is a continuous need to evolve our data processing solution to keep up with the speed at which data processing occurs. Being able to integrate a file system within Kubernetes, we anticipate an improve-

ment in data processing speed, allowing us to handle time-critical data processing requirements with ease. Enhancing speed is crucial for maintaining a competitive edge where timely insights and actions can make a huge difference.

To address the dilemma faced by companies when evaluating the feasibility of adopting complex workloads and setups, we further explore streamlined and configurable scripts for easily deployable solutions. By doing so, we aim to improve access to cutting-edge technologies, allowing companies to explore and integrate new solutions without taking on unnecessary time and finances.

1.4 Proposed solution

Our proposed solution addresses the limitations of Kubernetes in handling persistent data by leveraging Ceph. Ceph will be acting essentially as a backend object store for the Kubernetes environment to overcome the limitations faced with database. Through the creation of an External Ceph Storage Cluster, integrating into the Kubernetes (K8s) environment via Rook and reconfigured Helm charts, this approach transforms the database storage management within Kubernetes.

By incorporating Ceph's distributed and fault-tolerant architecture, this configuration revolutionizes the original database storage within Kubernetes, offering a standardized and optimal solution that is personalized for managing multiple types of databases and scales. These enhancements not only overcome the prevalent storage challenges encountered in large-scale Kubernetes deployments but also elevate database performance, responsiveness, and reliability. This method can be used by many applications; however, this was evaluated for a specific GIS application.

This approach represents an important milestone in the evolution of Kubernetes-based databases, effectively closing the gap between Kubernetes' fundamental design

principles and the intricate demands of data-intensive applications requiring persistent storage. By extending these capabilities, we allow for Kubernetes to better handle complex requirements of data-intensive workloads across multiple industries and use cases.

1.5 Contributions

The following outline contributions made to the field of data processing within the Kubernetes ecosystem given our proposed solution.

1. `Increased Efficiency for Ingests and Data Analysis:` The implementation of Ceph as a back-end storage solution significantly enhances the efficiency in data ingestion and analysis. By leveraging Ceph, data processing speed is notably improved. This improvement translates into faster ingest times, improving data analysis workflows. Additionally, decision-making processes are quicker as the data can be accessed and interpreted faster. Specifically, the average ingest time is reduced to 787 seconds compared to 909 seconds with the original method, depicting a 13% increase in data ingest speed, contributing to more timely and informed decision making process.

2. `Improved Fault Tolerance:` This improvement is attributed to Ceph's distributed architecture, which disperses data across multiple nodes, thus reducing the risk of single points of failure. Through various data redundancy and replication mechanisms, Ceph ensures that event in the event of hardware failures or network failures, data integrity is preserved. This continuous re-balancing improves the system resilience by dynamically adjusting data distribution to maintain optimal performance. As a result, organizations can rely on this infrastructure to upload data availability, mitigating the risk of data loss and thus

enhancing the overall system reliability.

3. `Automated Solution Deployment:` The automated solution framework brings an improvement in streamlining the integration of complex storage technologies. By creating a customizable script tailored for similar deployments, the accessibility to these complex storage solutions is improved for both organizations and developers. This approach not only simplifies the deployment process but also significantly reduces the associated challenges of time and financial investment required for learning and adapting these technologies. Additionally, it provides a beneficial environment for experimentation and innovation within Kubernetes data storage, removing the challenges of entering and progressing in this field. By offering a simplified and cost-effective way to deploy and explore storage solutions, this automation empowers organizations to optimize their Kubernetes environments efficiently, driving advancements and innovation in the Kubernetes ecosystem for data storage management.

4. `Centralizing Information:` In response to challenges stemming from outdated, inconsistent documentation and fragmented resources, efforts were made to consolidate and ease access to essential information. This aims to address issues of confusion arising from unclear instructions and the need to sift through multiple different documents and forms. By establishing a centralized repository, it mitigates the need for developers to face the overwhelming task of searching through various sources for critical details. Instead, they benefit from improved clarity and accessibility, finding relevant information available in one location. This consolidation saves time and also enhances workflow efficient, allowing developers to focus more on the tasks and goals rather than navigating through various documentation structures. This centralizing information

significantly contributes to a smoother and more productive environment.

1.6 Structure of Thesis

The remainder of the thesis is organized as follows: We start with the related works which is described in Chapter 2, which explores current current works in the area of data storage in Kubernetes environments. Following that, in Chapter 3, we present our implementation of the Ceph cluster and its integration within a Kubernetes environment. In Chapter 4, we describe the experiments and results for processing geospacial data within our Ceph configuration. Lastly, Chapter 5, we conclude our work and propose future research.

Chapter 2

Related Works

There are several storage options with Kubernetes. We begin with an examination of **read-only federated** databases in Section 2.1. Moving forward, Section 2.2 goes into a common approach to **scaling databases**, discussing the simplicity of this widely used method. We then pivot towards the cutting-edge technology of **edge computing** in Section 2.3, where we explore an approach aimed at bringing data closer to its source. Lastly, Section 2.4 looks at the existing work surrounding **Hadoop on Kubernetes**.

2.1 Federated Databases

Federated databases have been used in distributed database management systems, as they enable the integration of scattered databases into a unified system. Traditional databases are centralized and do not allow seamless communication between separate databases. Zheng et al. [14] introduce the idea of implementing a federated database to utilize and facilitate geospatial data. The goal is to containerize multiple distinct database instances that utilize Kubernetes to manage the container and database instance.

This approach focuses on emphasizing read-only queries and employing a Kubernetes-aided federated database system (FDBS). Building upon previous research on container-

native read-only databases for production [13], the proposed infrastructure distinguishes between stateless data access and stateful data processing, optimizing and reducing complexity.

Taking advantage of containerization principles and Kubernetes orchestration, the proposed FDBS aims to achieve improved performance and scalability, allowing the handling of large volumes of geospatial data. By emphasizing on read-only user queries this approach optimizes the use of pre-obtained and unchanged data, enhancing efficiency in applications. A typical example is the GIS domain where various instances of geospatial data can be acquired and utilized by different organizations via different channels, such as satellite, aerial imagery, topographic maps, GPS data, each providing insights and information for geospatial analysis. [3]. This simplified Kubernetes-aided FDBS allows for easier implementation, making it more accessible while also reducing the complexity and overhead involved. There are also benefits to setting a database for read-only data for both database administrators and programmers. For database administrators, utilizing a read-only database eliminates the overhead and challenges associated with transactions and concurrency control, including issues like locking [11] and multi-versioning [6]. For programmers, this simplifies multi-process data access, removing the need for message passing or shared memory [4] enhancing the efficiency and reducing the complexity associated with managing data access in a multi-process environment.

The main limitation of this approach is the emphasis on read-only queries. It has significant advantages when applications can utilize pre-obtained data. However, this design imposes constraints in applications requiring extensive read-write operations or real-time data interactions. When applications heavily rely on data manipulation or real-time updates, this read-only approach can restrict their functionality and responsiveness. When handling tasks such as frequent database updates, transactional

processing, and user interactions that require immediate data modifications, the reliance on read-only access can restrict this functionality. Additionally, the usage of pre-obtained data implies that the dataset is static. This could limit the scalability in environments where data may have frequent changes or additions.

2.2 Database Scaling

Auto-scaling has been a popular concept in cloud computing and distributed systems, allowing developers to adjust the resources allocated to their applications in response to a changing workload. Using this approach for resource management allows for optimal performance, scalability, and cost-effectiveness by automatically scaling components such as containers and application services up and down based on a prediction when looking at the current workload. By utilizing auto-scaling mechanisms, developers can more effectively handle various levels of demand while maintaining high availability.

This approach shifts the focus to applying the concept of auto-scaling to PostgreSQL databases within a Kubernetes environment. While auto-scaling has been utilized in various aspects of cloud computing and distributed systems, its application to databases, particularly within Kubernetes clusters, presents its own set of challenges like data synchronization and an inability to maximize availability [18]. By addressing these challenges, Perera et al. [16] aimed to provide a novel solution for managing highly available databases on Kubernetes through auto-scaling. This solution introduced a tailored auto-scaling algorithm designed to account for database workload characteristics and synchronization requirements. By integrating this solution with Kubernetes, developers can easily scale their PostgreSQL databases based on real-time demand, improving performance, data integrity, and availability.

When performing database scaling, two main strategies are often used: scaling and sharding. Scaling can be implemented in two ways: vertical and horizontal scaling. Sharding involves partitioning a database into smaller, independent parts referred to as shards, where each shard contains a subset of the data. This involves distributing the workload across multiple database instances, allowing for better parallelism and performance. Alternatively, scaling refers to adjusting the capacity of individual database instances to handle an increase in demand. Vertical scaling handles increasing the resources like CPU and memory of a single database instance, whereas horizontal scaling involves adding more database instances to the system. Horizontal scaling is what offers more flexibility and scalability by distributing the workload across multiple nodes, allowing for better resource utilization and fault tolerance.

Managing highly available PostgreSQL databases on Kubernetes through auto-scaling can offer several advantages, considering the prominence of SQL and relational databases being one of the most widely used database technologies [8]. By utilizing a tailored auto-scaling algorithm that takes into account database workload characteristics and synchronization requirements, scaling decisions can be made intelligently to maintain data availability. This can address the issue of fault tolerance in Kubernetes environments by mitigating the risk of data loss that can occur with sudden failures. By scaling database resources to maintain a high availability, one can minimize the likelihood of downtime and data loss incidents. Scaling a database may involve adding more PostgreSQL pods to the cluster where Kubernetes' orchestration will facilitate the deployment and management of these new database pods, evenly distributing them across the cluster and improving fault tolerance and system reliability. This scalability also allows the system to dynamically adjust resources in response to workloads to improve performance and resource utilization. This also allows the system to handle sudden spikes in demand without user intervention.

Although this approach offers several advantages, it demonstrates a few limitations. The main limitation of this approach is that its applicability is constricted solely to PostgreSQL databases, limiting its utility for developers employing several database technologies. While this method addresses fault tolerance to mitigate the risk of data loss due to node failures, catastrophic simultaneous failures across multiple nodes can still lead to data loss and corruption. Additionally, there exists a dilemma regarding the scaling of database containers together with their volumes [10]. Scaling out database instances may lead to unexpected dangling volumes which pose data consistency issues when different data is stored across various volumes. However, opting for a single database instance to avoid these issues will reverse the benefits of containerization, failing to leverage the capabilities of Kubernetes, like automated container deployment, scaling, and management [13]. Lastly, auto-scaling does not inherently enhance database access speed as it just facilitates resource allocation in response to the workload. Factors like network latency and throughput are not directly optimized, leaving some performance bottlenecks unaddressed.

2.3 Edge Computing

Edge computing is a cutting edge concept that focuses on processing data closer to the source of creation. This is typically done at the edge of the network, rather than relying solely on centralized data centers. The purpose of this approach is to minimize latency by reducing the distance data needs to travel, enhancing real-time processing and responsiveness. When deploying edge applications with containers, one should consider using a layered architecture as it's possible to divide it into different levels. This architecture can include the infrastructure layer which comprises the underlying computer, storage, and network resources, which can support the

container orchestration layer tasked with deploying and managing the life cycles of containers. This infrastructure layer then supports the application layer [7].

For this approach, Kubernetes can serve as the framework for deploying and managing containerized applications, which can be used for processing data at the edge. For this type of setup, the infrastructure layer consists of the physical and virtual resources that will be deployed at the edge location. The container orchestration layer, provided by Kubernetes, will oversee deployment, scaling, and monitoring of containers across the infrastructure. At the application layer, data processing tasks can be conducted, such as real-time analytics, machine learning, and IoT, through the use of containers. Kubernetes can facilitate these deployments and manage the data processing workloads, running at the edge.

Implementing edge computing combined with Kubernetes can offer several advantages, addressing the issue of data transfer latency, cost, and throughput. As organizations rely on cloud services, the distance between their local networks and cloud servers encompasses several challenges. Large volumes of data need to traverse large distances, leading to slow transfer times, escalating costs, and bottlenecks in throughput, as cloud providers charge based on data transfer volume. Utilizing an edge computing infrastructure means data processing tasks can be executed locally, eliminating the need for extensive data transfers to distant cloud servers. Deploying Kubernetes in this architecture also allows for efficient use of containerized applications, allowing to process data closer to its source without extensive latency or expenses, while reducing network usage to improve throughput. This shines in scenarios where fast data processing and high throughput are important, such as real-time analytics applications.

While this approach can offer significant advantages for processing data, it has some noteworthy limitations. Depending on the use, there are scenarios where de-

ploying servers close to the data may not be feasible due to constraints like power, space, or the environment. For instance, in environments like airplanes or vehicles, there can be limitations on the power and space available for hosting edge servers. This can cause challenges for real-time processing tasks that require significant computational resources. Furthermore, the mobile environments also introduce the risk of data loss if the edge server onboard the vehicle experiences failures.

2.4 Hadoop On Kubernetes

Hadoop on Kubernetes is where the deployment and management of Hadoop clusters occurs within Kubernetes. Hadoop, like Kubernetes, also focuses on distributed computing that requires several nodes to work together as a cluster. It has several components like Hadoop Distributed File System (HDFS), YARN, and MapReduce that can integrate well with Spark, a high-performance data analytics framework in a cluster [2]. Utilizing Kubernetes offers add-on services, such as logging, security, and monitoring [2]. This can allow companies to leverage the resources of distributed computing provided by a Hadoop cluster while taking advantage of the flexibility, efficiency, and diversity of Kubernetes to process and analyze large volumes of data more effectively.

This approach offers the ability to deploy HDFS within a Kubernetes cluster. Hadoop implements distributed file systems, where it uses the storage in each worker node to be part of the file system. This allows the application to have better read-write operation as the data is closer to the executor [2]. This can be combined with Kubernetes' persistent volume features by allowing HDFS nodes to be provisioned as containers with attached persistent volumes which allows for reliable storage within the cluster. This provides a separation between computation and storage while also

taking advantage of Kubernetes' built-in scheduling and resource allocation to have high availability and fault tolerance.

Implementing Hadoop on Kubernetes also offers several other benefits for data processing. Hadoop requires manual intervention to scale clusters up and down; however, when deployed on Kubernetes, it gains the capability to scale dynamically based on demand to ensure resource utilization for handling various workloads. It also provides flexibility, allowing for the deployment of multiple Hadoop clusters on the same infrastructure, enabling isolated workloads and better control over data processing environments. Furthermore, it allows the deployment of Hadoop clusters on any Kubernetes-compatible infrastructure like the cloud.

This approach presents several challenges that need to be considered before adapting this technique. Firstly, there's a high complexity in setting up, configuring, and managing the environment due to the integration of two separate complex systems. This complexity can lead to a long development time and expertise needed for both Hadoop and Kubernetes. Uber had several teams spend 2 years to re-architect their Hadoop deployment to be 60% within Docker containers with several challenges [15]. Additionally, there can be a trade-off in performance due to the added overhead of managing containers within Kubernetes compared to a traditional Hadoop deployment. Maintaining this complex system also requires a large team to ensure smooth operation and address issues. NameNode, responsible for storing metadata about the file system, also presents another challenge, as containerizing the NameNode and maintaining its stateful file system adds complexity to achieving high availability of the HDFS. Furthermore, integrating Zookeeper's Quorum-based Journal Manager (QJM) for NameNode competes with Kubernetes' native etcd for leader election [1].

Chapter 3

Implementation

3.1 Ceph

Ceph is a distributed storage system designed to meet the substantial demands of modern infrastructures. Its architecture focuses on scalability, fault tolerance, and high performance by utilizing a peer-to-peer model to help eliminate single points of failure, ensuring data reliability when hardware failures or network failures occur.

One of the unique features of Ceph is its support for multiple storage interfaces, including object storage, block storage, and file storage. This allows Ceph to address a variety of use cases, including traditional storage solutions and cloud-based infrastructures. These multiple storage options are:

- `RADOS (Reliable Autonomic Distributed Object Store):` Facilitates the storage and retrieval of unstructured data, ideal for cloud storage applications.
- `RBD (RADOS Block Device):` Offers block-level access to storage volumes, ideal for integration with virtualization platforms.
- `CephFS (Ceph File System):` Offers a distributed file system with POSIX standards, allowing users to access files in a hierarchical structure across the

entire cluster.

3.2 Script

Creating a script to deploy a Ceph cluster on Red Hat Enterprise Linux (RHEL) 8 can streamline the process of setting up a testing environment. A testing environment allows organizations and developers to experiment with various configurations, functionalities, and failures without risking production data.

The initial steps of the script involve the creation of a virtual machine (VM), where an admin node and a specified number of worker nodes are provisioned. The script attaches a user-provided ISO image to each VM and configures a network address translation (NAT) device to facilitate initial SSH access via port forwarding. The VMs can then be configured on a NAT network, the IP addresses are pulled from the machine, and the SSH connections are adjusted to utilize the IP address of its respective machine. This information is stored in a dictionary for future reference, while the hostname of each VM is set to its designated name.

The script then proceeds to setup a YUM repository from the user-specified location, and certificates are retrieved and installed in the PKI-CA-trust anchors. Essential packages and dependencies required for Ceph, including Python 3.8, systemd, Docker, Chrony, and LVM2, are installed. Chrony is then configured with the user's provided network time protocol (NTP) servers for time synchronization across the cluster.

Now, Cephadm can be installed on the admin node, utilizing the specified Cephadm URL. The cluster is then bootstrapped, and the Ceph key is copied for later use on the worker nodes. Additionally, the Ceph command line interface (CLI) is enabled on the admin node for cluster management tasks.

Before adding the worker nodes to the cluster, they must have a last preparation step. The Ceph key is added to an `authorized_keys` file within the `ssh` directory on each worker node to grant the admin node permission to interact with the worker nodes securely. Finally each worker node can be added to the cluster with the designated labels such as `mon`, `mgr`, `osd`, and `mds`, facilitating their roles within Ceph:

- `mon`: A monitor that maintain maps of the cluster state and coordinates changes to the cluster map and provide fault tolerance through quorum.
- `mgr`: A manager that provides cluster-wide management and collects cluster-wide statistics and metrics.
- `osd`: A object storage daemon that stores data, handle data replication, recovery, and rebalances the cluster.
- `mds`: A metadata server that manages metadata for CephFS, handling operations such as directory listings, file creation, file deletion, and permissions.

This script was designed to automate the deployment of a Ceph cluster on RHEL8, providing users with a straightforward and efficient means to set up a testing environment and familiarize themselves with Ceph's functionalities and operations. This script offers extensive customization options to cater to the unique requirements of each developer. Each person can tailor the deployment according to their specific needs by adjusting parameters such as the Cephadm version, ISO image, repository location, certificates, number of nodes, memory allocation, CPU resources, VM names, SSH ports, Ceph username and password, NAT network name, network CIDR,

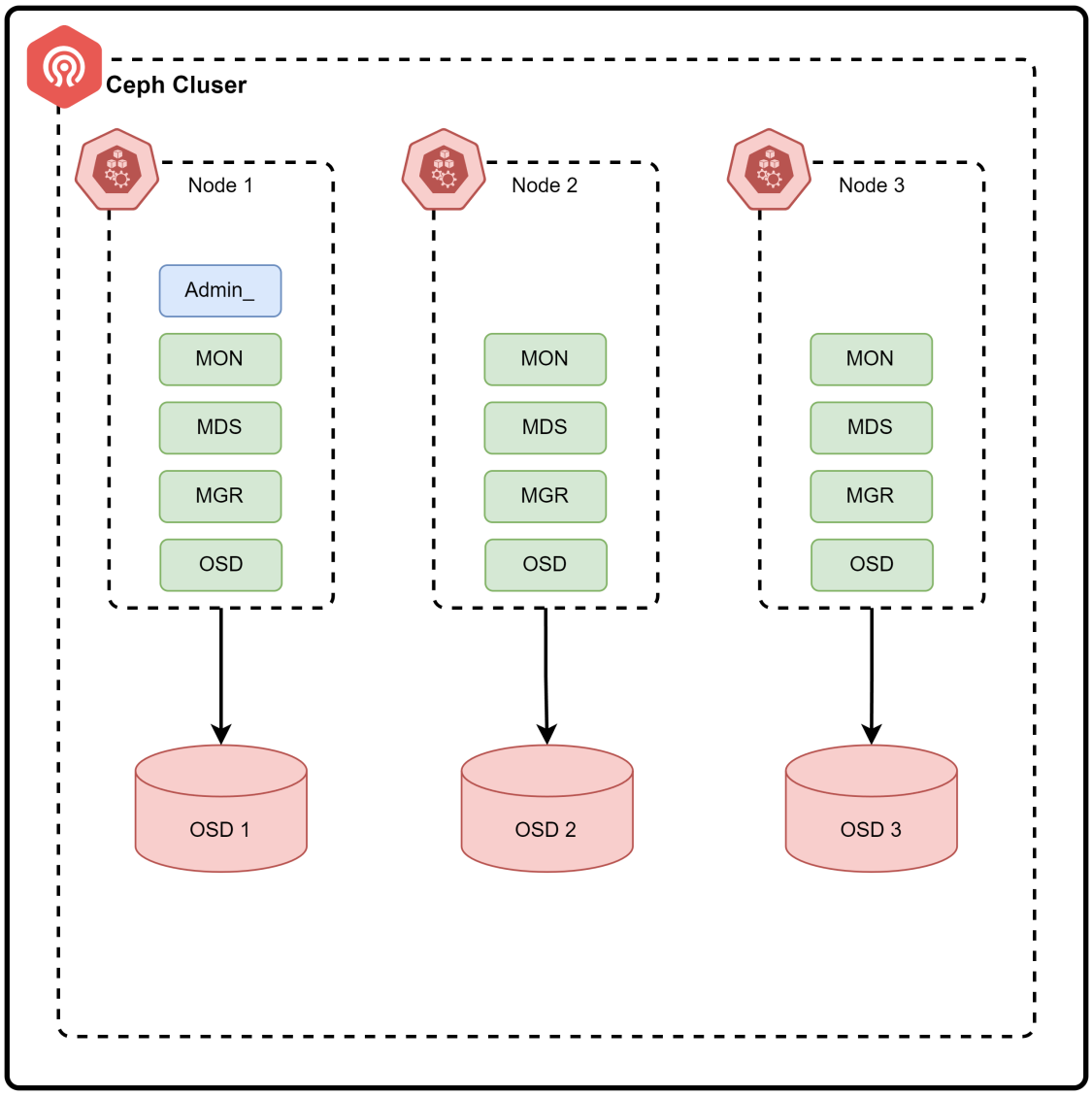

Figure 3.1: Illustrates a Ceph cluster consisting of 3 nodes. Each node handles critical Ceph services, including monitor (MON), metadata server (MDS), manager (MGR), and object storage daemon (OSD), with a dedicated OSD connected to every node.

and operating system type. This flexibility allows developers to optimize their Ceph infrastructure for their precise specifications and use case.

3.3 Single-Node K3s Cluster

For our purposes, our Ceph cluster will be connecting to a single-node K3s VM. This offers a lightweight platform for deploying and managing our containerized workloads. K3s is a Kubernetes distribution for resource-constrained environments, allowing us to run it on a single node within a VM.

We can utilize a batch processing system, Volcano, to handle computational workloads effectively within this environment. To help the data processing and batch scheduling tasks, an operator is utilized to help managing these workflows. This operator is used for various aspects of data processing, including job scheduling, resource allocation, monitoring, and more.

While the K3s cluster handles containerized workloads and batch processing tasks, our databases such as PostgreSQL, MySQL, and neo4j are used for persistent data storage and management. In this setup, databases are stored outside the k3s cluster but within the VM hosting the cluster. This architecture allows databases to maintain their own dedicated resources and storage while remaining accessible to applications running within the K3s environment.

3.4 Connecting K3s to External Ceph Cluster

To connect K3s to an external Ceph cluster, a series of steps needs to be followed for easy integration and proper functioning. Before starting, you should ensure that no Helm charts are running on the cluster that will be connected to the Ceph cluster. This prevents conflicts and ensures a clean environment for the integration.

Firstly, the K3s instance needs to be configured to pull from the appropriate private repository. This involves creating a `registries.yaml` file within the k3s directory, `/etc/rancher/k3s/`, and then restarting the K3s service to apply the changes.

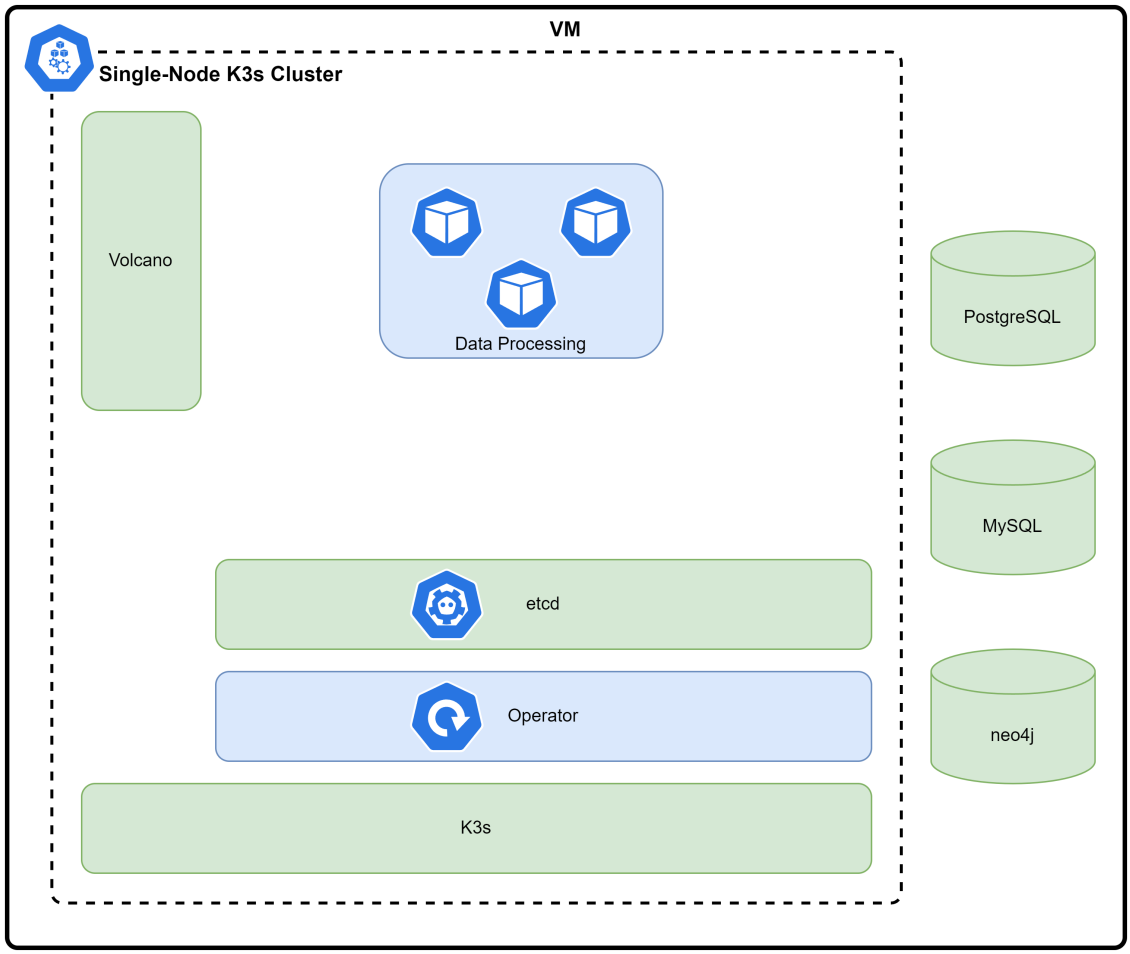

Figure 3.2: Illustrates a single-node K3s virtual machine cluster for data processing. This setup showcases the essential components such as K3s, Operator, etcd, and Volcano to facilitate various data processing tasks within the cluster. Additionally, databases are placed external to the K3s cluster but within the VM environment.

By utilizing a private repository, we gain better control over the container images. This can offer several benefits:

- Reducing the risk of security vulnerabilities associated with third-party images.
- Faster and more reliable deployments as the images are stored internally and do not have to interact with an external network.

- Complete control of when and how images are updated, minimizing the risk of issues.

Listing 3.1: registries.yaml

```
mirrors:
    registry.example.com:
        endpoint:
            - "https://registry.example.com:5000"
```

Once the K3s cluster is prepared to be connected, the export data from the Ceph cluster must be obtained and given to the K3s cluster. This data includes information about Ceph Monitors, file system ID (FSID), secrets, RADOS block device (RBD), CephFS, and other relevant details. Run the `create-external-cluster-resources.py` script to create the export data and create all users and keys on the Ceph cluster, as shown in Listing 3.2.

Listing 3.2: Command to Create External Cluster Resources

```
python3 create-external-cluster-resources.py --rbd-data-pool-
    name <pool_name> --cephfs-filesystem-name <filesystem_name
    > --namespace <namespace_name> --format bash
```

After obtaining the export data from the Ceph cluster, the next step is to integrate it into the K3s cluster. This is done by importing the obtained data, done by the execution of the ‘create-external-cluster-resources.py‘ script above. The output produced by this script needs to be directly pasted into the current shell of K3s. This important step sets environmental variables necessary for Rook to establish resources and connectivity between the two environments. In addition to pasting the output into the shell, it’s needed to execute the import script, `import-external-cluster.sh`.

This script orchestrates the integration of the Ceph cluster resources into the K3s environment. Running this script, ensures the necessary configurations and settings are applied.

After the external cluster resources have been imported, YAML files need to be added to the appropriate helm directory such as:

- `values.yaml`
- `values-external.yaml`

This directory structure is utilized by Helm to provide a standardized way to package and manage Kubernetes deployments. This directory usually contains the following structure:

- `values.yaml`: This file contains the default configuration values for a chart.
- `templates/`: This directory contains templates for Kubernetes manifest files used to deploy applications.
- `Charts/`: This directory contains dependencies required by the chart.

Additionally, a Rook Helm script ('rook_helm.sh') needs to be included in the home directory and given execute permissions. This script orchestrates the installation of Rook, which facilitates the Ceph integration within Kubernetes via Helm. This script will need to correctly map to the helm directory where the YAML files are located. Depending on the environment, additional configurations such as TLS might be needed, or you may opt to utilize the '–insecure-skip-tls-verify' flag in the helm commands to proceed without TLS verification.

After Rook has been deployed into your cluster, the CephCluster object will need to be modified to utilize the local Docker registry. This entails updating image locations and references within the CephCluster configuration to accurately point to the correct docker registry, as shown in Listing 3.3.

Listing 3.3: Command to Edit CephCluster

```
kubectl -n <rook_namespace> edit CephCluster
```

Finally, a verification step is performed to ensure the successful connection between K3s and the Ceph cluster. This includes checking the state of the CephCluster object to confirm connectivity and verifying the creation of the storage class based on the provided RBD pools and file system specified in Listing 3.2.

- `kubectl -n <rook_namespace> get CephCluster`
- `kubectl -n <rook_namespace> get sc`

3.5 Ceph Cluster Specs

Our Ceph cluster comprises of three distinct nodes, with one designated as an administrator node, overseeing the management and coordination of the cluster, while the remaining two are worker nodes, handling tasks.

Each node hosts its own dedicated Object Storage Daemon (OSD), a component responsible for managing storage operations within the cluster. This creates a distributed architecture with data storage, ensuring redundancy and fault tolerance, and enhancing the overall reliability of storage.

Among the three nodes, two of them have uniform hardware specifications, featuring a configuration of 20 CPUs, each containing 10 cores for parallel processing.

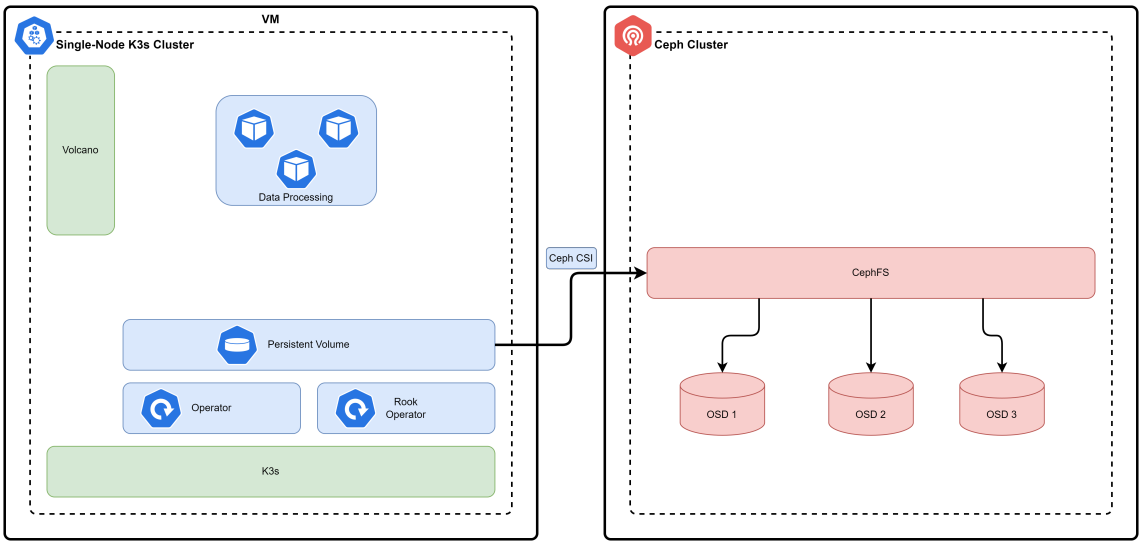

Figure 3.3: Illustrates the integration of K3s with the Rook Operator and Ceph. This showcases the addition of the Rook Operator to the K3s cluster setup, enabling the utilization of persistent volumes through the Ceph Container Storage Interface (CSI). This establishes a connection to the CephFS within the Ceph cluster infrastructure, the data is stored within the CephFS which is distributed and managed across Object Storage Devices (OSDs).

These nodes contain 125.1 GiB of memory capacity and a large 2.8 TiB of storage space. The OSD deployed on these nodes have a dedicated 1.8 TiB NVME SSD, ensuring optimal performance and reliability for storage workloads.

Alternatively, the third node contains a slightly different hardware configuration with 12 CPUs, each comprising 6 cores. It also possesses a smaller memory allocation of 62.3 GiB; however, it still contains a generous storage of 2.3 TiB. For the OSD deployment, this node also utilizes a dedicated 1.8 TiB NVME SSD for the OSD configuration.

Despite the variations in hardware specifications across the nodes, the operating system environment is uniform across the nodes. All nodes within the Ceph cluster operate on the RHEL 8.9 operating system, providing a consistent environment for operations and management.

3.6 Transitioning Storage Infrastructure

In our solution, we operate within a singular network environment where both our K3s single-node virtual machine (VM) and the external Ceph cluster exist. This network architecture ensures easy communication and data transfer between the K3s environment and the Ceph storage backend without having to go over multiple networks, thus reducing overhead during the data processing and storage management.

Originally, our K3s environment processes data while storing its databases directly on the VM but outside the cluster. Initially, the databases are stored using a hostpath storage, a method where storage volumes are mounted directly from the host file system into a container. While this approach works, it lacks the scalability, redundancy, and resilience needed for large-scale data processing and storage.

To address these limitations, we transitioned the storage backend of our K3s environment to leverage Rook to simplify the deployment and management of Ceph within the Kubernetes cluster. One of the main steps in this transition was rewriting our Helm charts to utilize Persistent Volume Claims (PVCs) and Persistent Volumes (PVs) connected to the CephFS Container Storage Interface (CSI). The CephFS CSI driver allows integration between Kubernetes and Ceph, providing dynamic provisioning and management of storage resources within our cluster.

However, in some environments, Rook may not accurately create storage classes for CephFS and RBD. In these cases, it may be necessary to manually create the StorageClass. Below is an example of how to create a StorageClass for CephFS:

Listing 3.4: Create CephFS StorageClass: storageclass-cephfs.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
```

```
  name: <sc_name>
provisioner: <rook_namespace>.cephfs.csi.ceph.com
parameters:
  clusterID: <rook_namespace>
  fsName: <fs_name>
  pool: <fs_pool>

  csi.storage.k8s.io/provisioner-secret-name: rook-csi-cephfs
    -provisioner
  csi.storage.k8s.io/provisioner-secret-namespace: <
    rook_namespace>
  csi.storage.k8s.io/controller-expand-secret-name: rook-csi-
    cephfs-provisioner
  csi.storage.k8s.io/controller-expand-secret-namespace: <
    rook_namespace>
  csi.storage.k8s.io/node-stage-secret-name: rook-csi-cephfs-
    node
  csi.storage.k8s.io/node-stage-secret-namespace: <
    rook_namespace>

reclaimPolicy: Delete
allowVolumeExpansion: true
```

Next, to create a Persistent Volume Claim for CephFS, you can use the following YAML code:

Listing 3.5: Create PVC for CephFS: pvc-cephfs.yaml

```
---
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: <pvc_name>
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 7Gi
  storageClassName: <cephfs_storageclass>
```

Additionally, with the updated Helm charts, our databases now automatically create PVCs connected to the CephFS rather than relying on hostpath storage, or utilize already created PVCs connected to the CephFS. This transition improves the scalability and reliability of our storage system, as the file system is now directly connected to the cluster via CSI, removing the need to rely on external storage solutions like NFS. This ensures that the storage is fully integrated within the cluster itself, optimizing performance for the data processing tasks.

Also, as part of this migration, we also had to adjust other Helm deployments to utilize this storage solution. This involved modifying the StorageClass associated with these deployments to connect to CephFS, ensuring a complete transition to the new storage infrastructure across our entire Kubernetes environment.

During the migration process, we also had to make an adjustment to apply increased permissions to all pod deployments that utilize the new PVCs connected to CephFS. This modification to our deployments was essential due to the necessity of interacting with the CephFS, requiring the setting `privileged: true` within the

`securityContext` field as shown in Listing 3.6:

Listing 3.6: Security Context for Pod Deployments

```
securityContext:
  privileged: true
```

This grants containers elevated privileges, allowing them to interact with the CephFS. Additionally, we specified an appropriate `fsGroup` to ensure consistent file ownership and permissions within the shared volumes.

Chapter 4

Evaluation

4.1 Experimental Design

Our experimental design revolves around comparing the performance and efficiency of two different storage solutions within our Kubernetes environment. The first setup, which serves as our baseline, utilizes K3s with storage handled through hostpath for our databases and storage storageclasses provisioned by Rancher and Kubernetes. In this configuration, storage is directly mounted from the host file system into containers, providing a straightforward storage solution within Kubernetes.

Additionally, as part of our baseline experimental setup, we also incorporate connectivity to an external PostgreSQL database hosted elsewhere on the same network. This database mimics a similar topology to our solution, allowing us to access the performance impact of utilizing an external database compared to storing data within our Kubernetes cluster. This external connection introduces network latency considerations that could affect overall performance and throughput.

In contrast, our experimental solution leverages the integration of K3s and Ceph via Rook, utilizing Persistent Volume Claims to store data within the Ceph file system. This approach offers greater scalability, redundancy, and resilience compared to hostpath storage, as well as providing dynamic provisioning and management of storage resources through the Ceph backend.

To conduct our tests, a python testing container was utilized within the same network as the K3s VM to allow communication between components. This containerized environment facilitated isolation and ease of use to connect to one of the services on the VM. This container would evaluate the performance of these two storage solutions, conducting a series of tests consisting of 10 ingests each from various geospatial sensors. These sensors support image and video to interpret movement and activity data, such as turns, acceleration, object recognition, time-stamping, and geo-referencing. Some of these sensors include Full Motion Video (FMV) and Ground Moving Target Indicator (GMTI):

- Full Motion Video (FMV) Data: FMV sensors capture video footage from aerial platforms such as drones or aircrafts. These sensors generate large volumes of video data. An ingest process for FMV sensor data can involve capturing, processing, and storing the video. Upon ingesting, the data undergoes processing steps before being persisted in storage. This processing can involve tasks such as object detection and classification for further analysis.

- Ground Moving Target Indicator (GMTI) Data: GMTI sensors detect and track moving objects on the ground. GMTI data typically consists of radar signals reflecting off objects, which are then processed to extract various information. Ingesting GMTI data involves processing the radar signals, extracting information, and storing the processed data.

These ingests are initiated concurrently and managed through Kubernetes and Volcano for completion. We measure the time taken in seconds for each set of ingests

to complete, allowing us to assess the efficiency and effectiveness of each storage setup under real-world workload scenarios. The tests involved both manual checks of API endpoints and automated time calculations facilitated by the tavern container, saving each test as XML files for documentation and analysis. It also an adopted practice of deleting data after each full test run to maintain a clean testing environment to ensure consistency and reliability. By running 30 tests on each option, we aim to gather extensive data to inform our decision-making process regarding the optimal storage solution for our Kubernetes environment.

4.2 Results

After conducting our series of tests to compare the baseline configuration (utilizing hostpath storage an external PostgreSQL database) and our implementation setup (integrating K3s with Ceph via Rook for storage), we have gathered results to give insights on the performance differences between these two storage solutions previously mentioned.

For the baseline configuration, the mean time taken for a series of ingests to process was approximately 908.84 seconds, with a standard deviation of 71.36 seconds. The minimum recorded time was 786.55 seconds, while the maximum was 1086.91 seconds. These metrics indicate inconsistent performance, with ingest times having a notable variation.

Alternatively, our experimental design utilizing Ceph storage showcased promising results. The mean time taken for a series of ingests decreased to approximately 786.58 seconds, with a reduced standard deviation of 41.83 seconds. Also, the minimum time decreased to 666.22 seconds, while the maximum time also decreased to 847.57 seconds.

These results suggest several advantages of the Ceph solution over our baseline configuration. The Ceph configuration showed a more consistent performance which can be concluded by the reduced standard deviation. This indicates a tighter distribution of ingest times, resulting in a more predictable and reliable processing of various geospatial senor data. This also demonstrated faster ingest times, with both the mean and minimum ingest times showing improvement compared to the baseline. This mean time had a reduction of approximately 13% which signifies a substantial improvement in processing efficiency.

Furthermore, with shorter processing times, our Kubernetes environment can handle a larger volume of data more effectively and facilitate timely decision-making. It allows us to more easily accommodate growing data volumes and adapt seamlessly into changing workload demands for better performance.

Ultimately, our Ceph integration solution for data storage displays clear advantages over the baseline configuration. With improved consistency and efficiency, it provides a more reliable response for handling geospatial sensor data.

	Baseline	Ceph
Mean:	908.84	787.58
Standard deviation:	71.36	41.83

Table 4.1: Summary comparison of ingest times between Ceph and baseline. The ingest times for Ceph have a mean of 787 and a standard deviation of 42 second, whereas the ingest times for baseline model have a mean of 909 and a standard deviation of 71 second.

4.3 Context and Limitations

It's important to explain what the results of our experiments do and do not convey. While our results demonstrated a significant improvement in speed and consistency compared to the baseline configuration, it's important to contextualize these results within the scope of our experiment and the limitations.

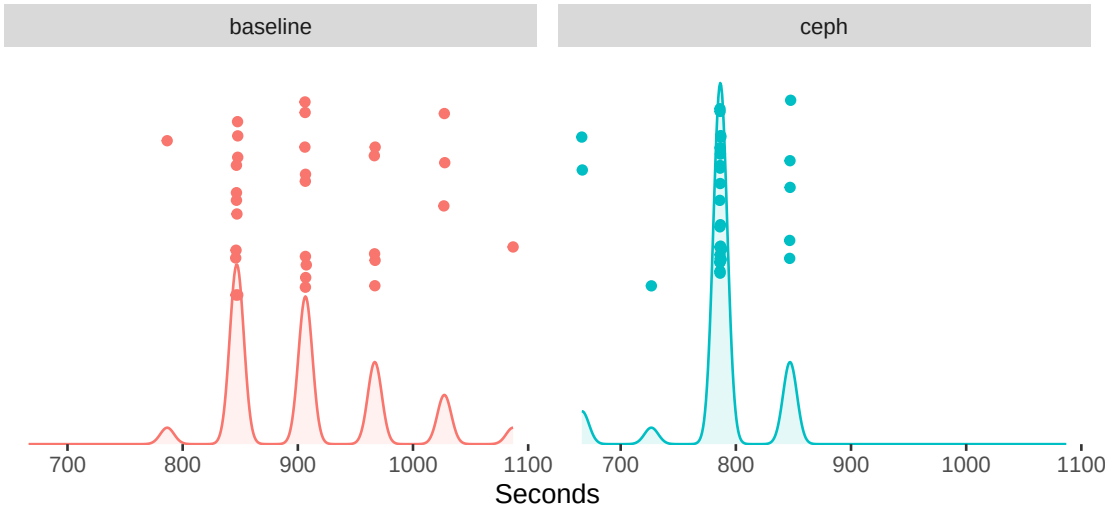

Figure 4.1: Comparison of ingest times between Ceph and baseline. The ingest times for Ceph range between 666 and 848 seconds, with a mean of 787 and a standard deviation of 42 second. The ingest times for baseline model range between 787 and 1087 seconds, with a mean of 909 and a standard deviation of 71 second.

Firstly, the observed speed improvements are significant; however, it's important to note that these results were obtained within the context of a single-node K3s VM environment rather than a full-fledged production Kubernetes cluster. While this configuration allows us to assess the performance of our storage solution in a controlled environment, it may not fully reflect certain complexities associated with a deployment scale across multiple nodes. Additionally, our K3s environment is basically the same as our K8s; they perform the same data processing tasks on the same data, just a more lightweight version.

Additionally, while our solution offers advantages in terms of speed and consistency, it's important to recognize that it may not be a one-size-fits-all solution for every use case. Organizations with unique use cases and requirements may need to change the implementation of this solution to suit their needs. Also, the choice between different storage solutions also depends on various factors such as data volume, access patterns, hardware specifications, and costs which should be explored.

Chapter 5

Conclusion

5.1 Conclusions

The problem we have addressed is the effective management of databases within traditional storage solutions within Kubernetes environments. The challenge deals with the need for scalable, resilient, and efficient storage systems to support the processing and management of a variety of geospatial sensor data. This also includes addressing issues such as slow ingest times, inconsistent performance, and the lack of fault tolerance. We have also provided a script that can easily be adapted to generate a Ceph cluster for developers to experiment with before applying this method to similar deployments.

Our solution to this problem lies within the integration of K3s with Ceph via Rook for storage management within Kubernetes. By leveraging Ceph's distributed storage ability and Rook's integration within Kubernetes cluster, we provide an efficient storage solution tailored to the demands of geospatial data processing workflows. This offers dynamic provisioning, fault tolerance, and enhanced performance, addressing the many shortcomings of traditional storage solutions.

The significance of our solution deals with its ability to significantly improve data processing efficiency and reliability within Kubernetes. By reducing ingest times, improving consistency, and ensuring fault tolerance, the proposed solution enables the

ability to efficiently process geospatial workflows, which can lead to faster insights and better decision-making. Additionally, the scalability and flexibility of this solution allows for the adaption to changing data requirements and scaling of storage infrastructure based on demand.

By implementing this solution, organizations can expect accelerated data processing, improved reliability, and scalability, ultimately leading to a more efficient infrastructure. Our work does not only address a critical need within the field of geospatial data but also provides a valuable contribution to the broader Kubernetes ecosystem, for innovative storage solutions.

5.2 Future Work

Moving forward, there are several opportunities for future development and improvements of our solution to continue addressing the challenges in data processing within Kubernetes environments.

Firstly, we aim to refine our existing solution to optimize performance. This can involve fine-tuning configuration parameters and optimizing resource allocation to improve the overall system efficiency.

Also, we intend to apply our solution to address more realistic versions of processing geospatial data. This could involve handling more complex sensors, larger datasets, or live data. By testing our solution against these scenarios, we could validate its effectiveness in real-world deployments.

Additionally, we can run our solution on a full Kubernetes production cluster to evaluate its performance and scalability under these more realistic conditions. Our K3s environment is a lightweight version of our Kubernetes production cluster that performs the same tasks and data processing. By deploying our solution in a

production environment, we can still gain more insight and assess its ability to handle various workloads and resource constraints on a larger scale.

Lastly, we can explore the configuration of Ceph in greater depth and deploy a much larger Ceph cluster to investigate its performance ability. By scaling up the Ceph infrastructure, we can simulate much higher data volumes and more demanding workloads to identify bottlenecks and optimize cluster configurations.

BIBLIOGRAPHY

[1] How can i deploy hdfs (hadoop distributed fs) to a k8s (kubernetes) cluster?, 2023. [online] Available: https://stackoverflow.com/questions/61736885/how-can-i-deploy-hdfs-hadoop-distributed-fs-to-a-k8s-kubernetes-cluster.

[2] Alimadji, D. R., Hung, M.-H., Lin, Y.-C., Suryajaya, B., and Chen, C.-C. A novel big data processing approach to feature extraction for electrical discharge machining based on container technology. In *2021 IEEE/ACIS 22nd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)* (2021), pp. 142–147.

[3] Butenuth, M., v. Gösseln, G., Tiedge, M., Heipke, C., Lipeck, U., and Sester, M. Integration of heterogeneous geospatial data in a federated database. *ISPRS Journal of Photogrammetry and Remote Sensing 62*, 5 (2007), 328–346. Theme Issue: Distributed Geoinformatics.

[4] Butler, R., and Pettey, C. Shareable, persistent, in-memory, read-only data. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)* (2015), The Steering Committee of The World Congress in Computer Science, p. 41.

[5] Delnat, W., Truyen, E., Rafique, A., Van Landuyt, D., and Joosen, W. K8-scalar: a workbench to compare autoscalers for container-orchestrated database clusters. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems* (2018), pp. 33–39.

[6] Faleiro, J. M., and Abadi, D. J. Rethinking serializable multiversion concurrency control. *arXiv preprint arXiv:1412.2324* (2014).

[7] Figueiredo, R., and Subratie, K. Demo: Edgevpn.io: Open-source virtual private network for seamless edge computing with kubernetes. In *2020 IEEE/ACM Symposium on Edge Computing (SEC)* (2020), pp. 190–192.

[8] Hainaut, J. L., Englebert, V., Henrard, J., Hick, J. M., and Roland, D. Database evolution: the db-main approach. In *Entity-Relationship Approach-ER'94 Business Modelling and Re-Engineering: 13th International Conference*

on the Entity-Relationship Approach Manchester, United Kingdom, December 13–16, 1994 Proceedings 13 (1994), Springer, pp. 112–131.

[9] Heddings, A. Should you run a database in docker?, 2020. [online] Available: https://www.cloudsavvyit.com/5414/should-you-run-a-database-in-docker/.

[10] Krzywiec, W. Database in a docker container – how to start and what's it about, 2019. [online] Available: https://wkrzywiec.medium.com/database-in-a-docker-container-how-to-start-and-whats-it-about-5e3ceea77e50.

[11] Kyte, T., Kuhn, D., Kyte, T., and Kuhn, D. Locking and issues. *Oracle Database Transactions and Locking Revealed* (2014), 7–28.

[12] Larsson, O. Running databases in a kubernetes cluster an evaluation, 2019.

[13] Li, Z. Long live the image: Container-native data persistence in production. In *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)* (2021), pp. 82–85.

[14] Li, Z., Saldías-Vallejos, N., Rodríguez, M. A., and Rainer, A. On kubernetes-aided federated database systems. In *2022 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)* (2022), pp. 1–8.

[15] Mathew, M., Shi, Q., Zhang, S., and Murchison, J. Containerizing apache hadoop infrastructure at uber, 2021. [online] Available: https://www.uber.com/blog/hadoop-container-blog/.

[16] Perera, H. C. S., De Silva, T. S. D., Wasala, W. M. D. C., Rajapakshe, R. M. P. R. L., Kodagoda, N., Samaratunge, U. S. S., and Jayanandana, H. H. N. C. Database scaling on kubernetes. In *2021 3rd International Conference on Advancements in Computing (ICAC)* (2021), pp. 258–263.

[17] "Should you run your database in Docker?", 2022, [online] Available: https://vsupalov.com/database-in-docker/.

[18] Tesliuk, A., Bobkov, S., Ilyin, V., Novikov, A., Poyda, A., and Velikhov, V. Kubernetes container orchestration as a framework for flexible and effective scientific data analysis. In *2019 Ivannikov Ispras Open Conference (ISPRAS)* (2019), pp. 67–71.

[19] B. Good, "To run or not to run a database on Kubernetes: What to consider", Jul. 2019, [online] Available: https://cloud.google.com/blog/products/databases/to-run-or-not-to-run-a-database-on-kubernetes-what-to-consider.

[20] Truyen, E., Van Landuyt, D., Lagaisse, B., and Joosen, W. Performance overhead of container orchestration frameworks for management of multi-tenant database deployments. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing* (2019), pp. 156–159.

Appendix A: Ceph Deployment Script

```
import argparse
import logging
import time

from ssh_lib import SimpleSSH
from vbox_utils import create_vm, attach_iso, start_vm, stop_vm,
    create_nat_network, apply_nat_network, nat_network_ports,
    nat_ports
from executor import LocalExecutor, SSHExecutor

default_cephadm_url =
    'https://download.ceph.com/rpm-18.2.0/el8/noarch/cephadm'
default_iso_path = 'PATH-TO-ISO'
default_repo_url = 'URL-TO-REPO'
default_cert_url = 'URL-TO-CERT'

default_nodes = 3
default_memory = 8192
default_cpus = 4
```

```
17 default_vm_name = f'Ceph'
18 default_ssh_port = 22170
19
20 default_username = 'root'
21 default_password = 'password'
22 default_network_name = 'cephNatNetwork'
23
24 def parse_args(argv=None):
25     parser = argparse.ArgumentParser(description='Automate Ceph
       ↪ Deployment')
26     parser.add_argument('--hostname', default='localhost', help='The
       ↪ hostname to use while logging in via SSH')
27     parser.add_argument('--ostype', default='RedHat_64', help='The OS
       ↪ type of the VM')
28     parser.add_argument('--nodes', default=default_nodes, type=int,
       ↪ help='The number of worker nodes')
29     parser.add_argument('--cpus', default=default_cpus, type=int,
       ↪ help='The number of CPUs for the VMs')
30     parser.add_argument('--memory', default=default_memory, type=int,
       ↪ help='The memory for the VMs')
31     parser.add_argument('--network_name',
       ↪ default=default_network_name, help='NAT Network name for the
       ↪ VMs')
32     parser.add_argument('--user', default=default_username, help='The
       ↪ user to use while logging in via SSH')
```

```
33      parser.add_argument('--password', default=default_password,
        ↪  help='The password to use when logging in via SSH.')
34      parser.add_argument('--cephadm_url', default=default_cephadm_url,
        ↪  help='The URL where cephadm installation is')
35      parser.add_argument('--network_cidr', default='145.42.3.0/24',
        ↪  help='The IP addresses to set the vms on')
36      parser.add_argument('--vm_name', default=default_vm_name,
        ↪  help='The name of the VM')
37      parser.add_argument('--ssh_port', default=default_ssh_port,
        ↪  help='The port to forward for SSH connections')
38      parser.add_argument('--iso_path', default=default_iso_path,
        ↪  help='The path to the VM iso file')
39      parser.add_argument('--cert_url', default=default_cert_url,
        ↪  help='The url to the certificates')
40      parser.add_argument('--repo_url', default=default_repo_url,
        ↪  help='The url to the repo')
41      args = parser.parse_args(argv)
42      return args
43
44  def deploy_ceph(executor, cephadm_url, vm_name, ostype, nodes,
    ↪  memory, cpus, iso_path, hostname, ssh_port, user, password,
    ↪  network_name, cert_url, repo_url):
45      """Deploys a ceph cluster within virtualbox
46
47      Args:
```

```
48          executor (Executor object): Executor that will run commands
            ↪ on the host machine
49          cephadm_url (string): The url for the cephadm installation
50          vm_name (string): The name of the VMs
51          ostype (string): The OS type of the VM
52          nodes (int): The number of worker nodes
53          memory (int): The memory, in KB, for the VMs
54          cpus (int): The cpus for the VMs
55          iso_path (string): The full path to the iso file
56          hostname (string): The hostname to use while logging in
            ↪ with SSH
57          ssh_port (int): The port to use for SSH connections
58          user (string): The user to use when logging in with SSH
59          password (string): The password to use when logging in with
            ↪ SSH
60          network_name (string): The name of the nat network
61          cert_url (string): The full url to the certificates
62          repo_url (string): The full url to the repo
63      """
64      try:
65          vm_info = create_vms(executor, vm_name, ostype, nodes,
            ↪ memory, cpus, ssh_port, iso_path, user, password)
66          vm_ip_info = setup_vms(executor, vm_info, network_name,
            ↪ hostname, user, password, cert_url, repo_url)
67
```

```
68          install_packages(vm_ip_info, hostname, user, password)
69          for vm_name, vm_data in vm_ip_info.items():
70              if 'Admin' in vm_name:
71                  admin_vm_exe = SSHExecutor(client=SimpleSSH(hostname,
                    ↪  vm_data['ssh'], user, password))
72                  install_cephadm(admin_vm_exe, cephadm_url)
73                  ceph_key = bootstrap_cluster(admin_vm_exe,
                    ↪  vm_data['ip'], user, password)
74                  prepare_hosts(vm_ip_info, ceph_key, hostname, user,
                    ↪  password)
75                  enable_ceph_cli(admin_vm_exe)
76                  add_hosts(admin_vm_exe, vm_ip_info)
77                  break
78      finally:
79          logging.debug('Script Stopped')
80
81  def create_vms(executor, vm_name, ostype, nodes, memory, cpus,
    ↪  ssh_port, iso_path, user, password):
82      """Creates the VM nodes for the ceph cluster
83
84      Args:
85          executor (Executor object): Executor that will run commands
            ↪  on the host machine
86          vm_name (string): The name of the VMs
87          ostype (string): The OS type of the VM
```

```
88          nodes (int): The number of worker nodes
89          memory (int): The memory, in KB, for the VMs
90          cpus (int): The cpus for the VMs
91          ssh_port (int): The port to use for SSH connections
92          iso_path (string): The full path to the iso file
93          user (string): The user to use when logging in with SSH
94          password (string): The password to use when logging in with
            ↪ SSH
95
96      Returns:
97          dict[string, string]: A dictionary that contains the VM
            ↪ info of [vm_name, ssh_port]
98      """
99      logging.debug('Creating VMs')
100     vm_info = {}
101     # Create Ceph-Admin Node
102     admin_vm_name = f'{vm_name}-Admin'
103     create_vm(executor, admin_vm_name, ostype, memory, cpus)
104     attach_iso(executor, admin_vm_name, iso_path, user, password)
105     vm_info[admin_vm_name] = ssh_port
106     stop_vm(executor, admin_vm_name)
107     nat_ports(executor, admin_vm_name, 'ssh', ssh_port, 22)
108
109     # Create nodes
110     for i in range(1, nodes + 1):
```

```
111             node_vm_name = f'{vm_name}-Node{i}'
112             create_vm(executor, node_vm_name, ostype, memory, cpus)
113             attach_iso(executor, node_vm_name, iso_path, user, password)
114             current_port = ssh_port + int(i)
115             vm_info[node_vm_name] = current_port
116             stop_vm(executor, node_vm_name)
117             nat_ports(executor, node_vm_name, 'ssh', current_port, 22)
118 
119     return vm_info
120 
121 def setup_vms(executor, vm_info, network_name, hostname, user,
    ↪  password, cert_url, repo_url):
122     """Setup the VM nodes to contain all the dependencies and
        ↪  information needed to create a ceph cluster
123 
124     Args:
125         executor (Executor object): Executor that will run commands
            ↪  on the host machine
126         vm_info (dict[string, string]): A dictionary that contains
            ↪  the VM info of [vm_name, ssh_port]
127         network_name (string): The name of the nat network
128         hostname (string): The hostname to use while logging in
            ↪  with SSH
129         user (string): The user to use when logging in with SSH
```

```
130         password (string): The password to use when logging in with
            ↪ SSH
131         cert_url (string): The full url to the certificates
132         repo_url (string): The full url to the repo
133 
134     Returns:
135         dict[string: [string, string]]: A dictionary containing the
            ↪ VM info of [vm_name [ssh_port, ip_addr]]
136     """
137     logging.debug('Setting up VMs')
138     vm_ip_info = {}
139     create_nat_network(executor, network_name)
140     for vm_name, vm_ssh in vm_info.items():
141         apply_nat_network(executor, network_name, vm_name)
142         start_vm(executor, vm_name)
143 
144     time.sleep(280)
145     for vm_name, vm_ssh in vm_info.items():
146         try:
147             vm_exe = SSHExecutor(client=SimpleSSH(hostname, vm_ssh,
                ↪ user, password))
148             vm_ip = obtain_ip(vm_exe)
149             nat_network_ports(executor, network_name, vm_name,
                ↪ vm_ssh, "127.0.0.1", "22", vm_ip)
150 
```

```
151             vm_exe.run('rm /etc/hostname')
152             vm_exe.run(f'echo {vm_name} > /etc/hostname')
153             vm_exe.run(f'hostname {vm_name}')
154 
155             set_repo(vm_exe, repo_url)
156             set_certs(vm_exe, cert_url)
157 
158             vm_ip_info[vm_name] = {'ssh': vm_ssh, 'ip': vm_ip}
159         finally:
160             logging.debug('VMs Info: ', vm_ip_info)
161 
162     return vm_ip_info
163 
164 def obtain_ip(executor):
165     """Configures the nat network on NIC2 to be on and auto
        ↪ connect. Then obtains the machines IP address on the
        ↪ network.
166 
167     Args:
168         executor (Executor object): Executor that will run commands
            ↪ on the VM
169 
170     Returns:
171         string: The IP address of the machine
172     """
```

```
173     logging.debug('Turning on Nat Network')
174     executor.run('nmcli con up enp0s8') # Turn on nat network (NIC2)
175     executor.run('nmcli con mod enp0s8 connection.autoconnect yes') #
        ↪  auto connect nat network (NIC2)
176
177     vm_output = executor.run('ip addr show enp0s8 | grep -Po \'inet
        ↪  \K[\d.]+\'')
178
179     return vm_output[0]
180
181 def set_repo(executor, repo_url):
182     """Adds the repo into the VM
183
184     Args:
185         executor (Executor object): Executor that will run commands
            ↪  on the VM
186         repo_url (string): The full url to the repo
187     """
188     logging.debug('Adding Repo')
189     repo_parts = repo_url.split('/')
190     repo_name = repo_parts[-1]
191     executor.run(f'curl -o /etc/yum.repos.d/{repo_name} {repo_url}')
192     executor.run('yum clean all')
193     executor.run('yum list')
194
```

```
195 def set_certs(executor, cert_url):
196     """Adds the certificates into the VM
197 
198     Args:
199         executor (Executor object): Executor that will run commands
        ↪  on the VM
200         cert_url (string): The full url to the certificates
201     """
202     logging.debug('Adding Certificates')
203     cert_parts = cert_url.split('/')
204     cert_name = cert_parts[-1]
205     executor.run(f'curl -o
    ↪  /etc/pki/ca-trust/source/anchors/{cert_name} {cert_url}')
206     executor.run('update-ca-trust extract')
207 
208 def install_packages(vm_ip_info, hostname, user, password):
209     """Installs the nessesary packages needed for ceph on the VM
210 
211     Args:
212         vm_ip_info (dict[string: [string, string]]): A dictionary
        ↪  containing the VM info of [vm_name [ssh_port, ip_addr]]
213         hostname (string): The hostname to use while logging in
        ↪  with SSH
214         user (string): The user to use when logging in with SSH
```

```
215             password (string): The password to use when logging in with
                ↪ SSH
216         """
217         logging.debug('Installing required packages')
218         for vm_name, vm_data in vm_ip_info.items():
219             try:
220                 vm_exe = SSHExecutor(client=SimpleSSH(hostname,
                    ↪ vm_data['ssh'], user, password))
221                 vm_exe.run('yum install -y python38 systemd docker chrony
                    ↪ lvm2')
222                 time.sleep(60) # Allow time for packages to be installed
223                 configure_chrony(vm_exe)
224             finally:
225                 logging.debug('Packages installed')
226
227 def configure_chrony(executor):
228     """Configures chrony to connect to the correct NTP servers
229
230     Args:
231         executor (Executor object): Executor that will run commands
                ↪ on the VM
232     """
233     logging.debug('Configuring Chrony')
234     executor.run('sed \"s/^server*/#server/g\" /etc/chrony.conf >
        ↪ /tmp/chrony.conf')
```

```
235     executor.run('echo \"server (NTP-SERVER1) iburst\" >>
        ↪ /tmp/chrony.conf')
236     executor.run('echo \"server (NTP-SERVER2) iburst\" >>
        ↪ /tmp/chrony.conf')
237     executor.run('sed \"s/^#allow*/allow/g\" /tmp/chrony.conf >
        ↪ /etc/chrony.conf')
238     executor.run('rm -f /tmp/chrony.conf')
239     executor.run('chronyc -a \'burst 4/4\'')
240     executor.run('systemctl restart chronyd')
241
242 def install_cephadm(executor, cephadm_url):
243     """Installs cephadm onto the admin node
244
245     Args:
246         executor (Executor object): Executor that will run commands
            ↪ on the VM
247         cephadm_url (string): The url for the cephadm installation
248     """
249     logging.debug('Installing Cephadm on Admin Node')
250     executor.run(f'curl --silent --remote-name --location
        ↪ {cephadm_url}')
251     executor.run('chmod +x cephadm')
252     executor.run('./cephadm add-repo --release reef')
253     executor.run('./cephadm install')
254
```

```
255
256 def bootstrap_cluster(executor, mon_ip, user, password):
257     """Creates a ceph cluster with just the admin node connected
258
259     Args:
260         executor (Executor object): Executor that will run commands
        ↪  on the VM
261         mon_ip (string): The IP address of the admin node
262         user (string): The user to use when logging into the ceph
        ↪  dashboard
263         password (string): The password to use when logging into
        ↪  the ceph dashboard
264
265     Returns:
266         _type_: _description_
267     """
268     logging.debug('Bootstrapping Ceph Cluster')
269     executor.run(f'cephadm bootstrap --mon-ip {mon_ip}
    ↪  --initial-dashboard-user {user} --initial-dashboard-password
    ↪  {password}')
270     ceph_key = executor.run('cat /etc/ceph/ceph.pub')
271
272     return ceph_key[0]
273
274 def enable_ceph_cli(executor):
```

```
275     """Enables ceph command line interface
276
277     Args:
278         executor (Executor object): Executor that will run commands
            ↪ on the VM
279     """
280     logging.debug('Enabling Ceph CLI')
281     executor.run('cephadm install ceph-common')
282
283 def prepare_hosts(vm_ip_info, ceph_key, hostname, user, password):
284     """Adds the necessary authorization key information to the
        ↪ worker nodes
285
286     Args:
287         vm_ip_info (dict[string: [string, string]]): A dictionary
            ↪ containing the VM info of [vm_name [ssh_port, ip_addr]]
288         ceph_key (string): The ceph authorization key
289         hostname (string): The hostname to use while logging in
            ↪ with SSH
290         user (string): The user to use when logging in with SSH
291         password (string): The password to use when logging in with
            ↪ SSH
292     """
293     logging.debug('Preparing Hosts')
294     for vm_name, vm_data in vm_ip_info.items():
```

```
295             if 'Node' in vm_name:
296                 vm_exe = SSHExecutor(client=SimpleSSH(hostname,
                    ↪  vm_data['ssh'], user, password))
297                 vm_exe.run('mkdir .ssh')
298                 vm_exe.run(f'echo {ceph_key} > ~/.ssh/authorized_keys')
299
300 def add_hosts(executor, vm_ip_info):
301     """Adds a host to the cluster
302
303     Args:
304         executor (Executor object): Executor that will run commands
            ↪  on the VM
305         vm_ip_info (dict[string: [string, string]]): A dictionary
            ↪  containing the VM info of [vm_name [ssh_port, ip_addr]]
306     """
307     logging.debug('Adding Hosts')
308     for vm_name, vm_data in vm_ip_info.items():
309         if 'Node' in vm_name:
310             vm_ip = vm_data['ip']
311             executor.run(f'ceph orch host add {vm_name} {vm_ip}
                ↪  --labels=mon,mgr,osd,mds')
312
313
314 if __name__ == '__main__':
315     args = parse_args()
```

```
316
317     logging.debug('Arguments:')
318     for arg in vars(args):
319         logging.debug('  %s: %s', arg, getattr(args, arg))
320
321     exe = LocalExecutor()
322     deploy_ceph(exe, args.cephadm_url, args.vm_name, args.ostype,
        ↪  args.nodes, args.memory, args.cpus, args.iso_path,
        ↪  args.hostname, args.ssh_port, args.user, args.password,
        ↪  args.network_name, args.cert_url, args.repo_url)
```

Code 1: ceph_deploy.py

```
1  import re
2  import logging
3  import time
4  import os
5  import subprocess
6
7  # Parse vboxmanage list runningvms output.
8  vm_list_re = re.compile('"(.+)" {(.+)}')
9
10 def is_vm_running(executor, vm_name):
11     """Checks if a specified VM is running
12
13     Args:
```

```
14          executor (Executor object): Executor that will run the
            ↪  vboxmanage commands
15          vm_name (string): The name of the VM to check
16
17      Returns:
18          bool: True if the VM is running, otherwise false
19      """
20      logging.debug('is_vm_running: %s', vm_name)
21      vm_list = executor.run('vboxmanage list runningvms')
22      for line in vm_list:
23          match = vm_list_re.match(line)
24          if match is not None and match.group(1) == vm_name:
25              return True
26      return False
27  def get_vm_directory(vm_name):
28      """Returns the directory of the VM
29
30      Args:
31          vm_name (string): The VM's name
32      """
33      vm_info = subprocess.Popen(['vboxmanage', 'showvminfo', vm_name],
34                                 universal_newlines=True,
35                                 stdout=subprocess.PIPE,
                                   ↪  stderr=subprocess.PIPE)
36
```

```
37      stdout, stderr = vm_info.communicate()
38
39      for line in stdout.splitlines():
40          if 'Config file:' in line:
41              full_path = line.split(':')[1].strip()
42              vm_dir = os.path.dirname(full_path)
43              return vm_dir
44
45      raise ValueError(f'Directory for {vm_name} not found in
        ↪  vboxmanage output')
46
47  def create_vm(executor, vm_name, ostype, memory, cpus):
48      """Creates a VM
49
50      Args:
51          executor (Executor object): Executor that will run the
            ↪  vboxmanage commands
52          vm_name (string): The name of the VM to create
53          ostype (string): The type of OS for the VM
54          memory (int): The memory, in KB, for the VM
55          cpus (int): The number of CPUs for the VM
56      """
57      executor.run(f'vboxmanage createvm --name {vm_name} --ostype
        ↪  {ostype} --register')
```

```
58      executor.run(f'vboxmanage modifyvm {vm_name} --memory {memory}
        ↪  --cpus {cpus}')
59
60  def attach_iso(executor, vm_name, iso_path, user, password):
61      """Attach an ISO image to the VM
62
63      Args:
64          executor (Executor object): Executor that will run the
            ↪  vboxmanage commands
65          vm_name (string): The name of the VM you want to attach an
            ↪  iso to
66          iso_path (bool): The full path to the iso file
67          user (string): The user to use while logging in
68          password (string): The password to use while logging in
69      """
70      if is_vm_running(executor, vm_name):
71          executor.run(f'vboxmanage controlvm {vm_name} poweroff')
72      vm_path = get_vm_directory(vm_name)
73      executor.run(f'vboxmanage createhd --filename
        ↪  \"{vm_path}/{vm_name}/{vm_name}.vdi\" --size 20000')
74      executor.run(f'vboxmanage storagectl {vm_name} --name \"SATA
        ↪  Controller\" --add sata')
75      executor.run(f'vboxmanage storageattach {vm_name} --storagectl
        ↪  \"SATA Controller\" --port 0 --device 0 --type hdd --medium
        ↪  \"{vm_path}/{vm_name}/{vm_name}.vdi\"')
```

```
76      executor.run(f'vboxmanage storagectl {vm_name} --name \"IDE
        ↪  Controller\" --add ide')
77      executor.run(f'vboxmanage storageattach {vm_name} --storagectl
        ↪  \"IDE Controller\" --port 0 --device 0 --type dvddrive
        ↪  --medium {iso_path}')
78      executor.run(f'vboxmanage modifyvm {vm_name} --boot1 dvd --boot2
        ↪  disk --boot3 none --boot4 none')
79      executor.run(f'vboxmanage unattended install {vm_name} --iso
        ↪  {iso_path} --user {user} --password {password}')
80
81      start_vm(executor, vm_name)
82
83  def start_vm(executor, vm_name, timeout=300):
84      """Starts a VM if it is not already running.  Returns once the
        ↪  VM is running
85
86      Args:
87          executor (Executor object): ci_scripts.executor.Executor
            ↪  that will run the vboxmanage commands
88          vm_name (string): Name of the VM to start
89          timeout (int): Number of seconds to wait for the VM to
            ↪  start before
90              throwing an exception
91      """
92      logging.debug('start_vm: %s', vm_name)
```

```
93      if is_vm_running(executor, vm_name):
94          return
95
96      logging.debug('vboxmanage startvm %s', vm_name)
97      executor.run(f'vboxmanage startvm {vm_name} --type headless')
98      time.sleep(10)
99      start = time.time()
100     while not is_vm_running(executor, vm_name):
101         if time.time() - start > timeout:
102             raise (f"{vm_name} is not running")
103         time.sleep(10)
104
105     logging.debug('VM STARTED: %s', vm_name)
106
107 def stop_vm(executor, vm_name, abort=False):
108     """Stops a VM if its running
109
110     Args:
111         executor (Executor object): ci_scripts.executor.Executor
            ↪  that will run the vboxmanage commands
112         vm_name (string): Name of the VM to stop
113         abort (bool, optional): If true, power off the VM.  If
            ↪  false, shutdown the VM
114     """
115     logging.debug('stop_vm: %s', vm_name)
```

```
116     if not is_vm_running(executor, vm_name):
117         return
118
119     if abort:
120         logging.debug('poweroff %s', vm_name)
121         executor.run(f'vboxmanage controlvm {vm_name} poweroff')
122     else:
123         logging.debug('acpipowerbutton %s', vm_name)
124         executor.run(f'vboxmanage controlvm {vm_name}
            ↪ acpipowerbutton')
125         executor.run(f'vboxmanage controlvm {vm_name} poweroff')
126
127     while is_vm_running(executor, vm_name):
128         time.sleep(5)
129
130 def reconfigure_vm(executor, vm_name, cpus=None, memory=None):
131     """Modifies an existing VM.  The VM must be powered off.
132
133     Args:
134         executor (Executor): The executor that will run the
            ↪ vboxmanage commands
135         vm_name (string): The VM's name
136         cpus (int, optional): the new number of CPU cores
137         memory (int, optional): the new RAM of the VM, in MB
138     """
```

```
139     if cpus is None and memory is None:
140         raise ValueError("Can't modify VM with no values to modify")
141     modify_cmd = ['vboxmanage', 'modifyvm', vm_name]
142     if cpus is not None:
143         modify_cmd.extend(['--cpus', str(cpus)])
144     if memory is not None:
145         modify_cmd.extend(['--memory', str(memory)])
146     logging.debug('reconfiguring %s', vm_name)
147     executor.run(modify_cmd)
148 
149 def create_nat_network(executor, network_name,
    ↪ network_cidr='145.42.3.0/24', dhcp_enabled=True):
150     """_summary_
151 
152     Args:
153         executor (Executor): The executor that will run the
            ↪ vboxmanage commands
154         network_name (string): The name of the nat network
155         network_cidr (string, optional): The IP addresses for the
            ↪ nat network to use. Defaults to '145.42.3.0/24'.
156         dhcp_enabled (bool, optional): Utilize dhcp. Defaults to
            ↪ True.
157     """
158     if dhcp_enabled:
```

```
159         executor.run(f'vboxmanage natnetwork add --netname
            ↪ {network_name} --network {network_cidr}  --dhcp on
            ↪ --enable')
160     else:
161         executor.run(f'vboxmanage natnetwork add --netname
            ↪ {network_name} --network {network_cidr} --disable')
162
163 def apply_nat_network(executor, network_name, vm_name):
164     """Modify a VM to utilize a nat network
165
166     Args:
167         executor (Executor): The executor that will run the
            ↪ vboxmanage commands
168         network_name (string): The name of the nat network
169         vm_name (string): The VM's name
170     """
171     if is_vm_running(executor, vm_name):
172         stop_vm(executor, vm_name)
173     cmd = f'vboxmanage modifyvm {vm_name} --nic2 natnetwork
        ↪ --nat-network2 {network_name}'
174     logging.debug('import cmd: %s', cmd)
175     executor.run(cmd)
176
177 def nat_ports(executor, vm_name, port_name, host_port, guest_port):
178     """Configure port forwarding on a nat nic
```

```
179
180         Args:
181             executor (Executor): The executor that will run the
                ↪  vboxmanage commands
182             vm_name (string): The VM's name
183             port_name (string): The name of the port you want to forward
184             host_port (string): The port for the host machine
185             guest_port (string): The port for the guest machine
186         """
187         executor.run(f'vboxmanage modifyvm {vm_name} --nic1 nat --natpf1
            ↪  {port_name},tcp,,{host_port},,{guest_port}')
188
189     def nat_network_ports(executor, nat_network, forward_name, host_port,
        ↪  host_ip, guest_port, guest_ip):
190         """Configure port forwarding on a nat network
191
192         Args:
193             executor (Executor): The executor that will run the
                ↪  vboxmanage commands
194             nat_network (string): The name of the nat network
195             forward_name (string): The name of the port you want to
                ↪  forward
196             host_port (string): The port for the host machine
197             host_ip (string): The IP for the host machine
198             guest_port (string): The port for the guest machine
```

```
199             guest_ip (string): The IP for the guest machine
200         """
201         executor.run(f'vboxmanage natnetwork modify --netname
    ↪ {nat_network} --port-forward-4 \"{forward_name}:tcp:[{host_ip
    ↪ }]:{host_port}:[{guest_ip}]:{guest_port}\"')
```

Code 2: vbox_utils.py

Appendix B: Rook Configuration

```
"""
Copyright 2020 The Rook Authors. All rights reserved.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
↪  implied.
See the License for the specific language governing permissions and
limitations under the License.
"""

import errno
import sys
```

```
19 import json
20 import argparse
21 import re
22 import subprocess
23 import hmac
24 from hashlib import sha1 as sha
25 from os import linesep as LINESEP
26 from os import path
27 from email.utils import formatdate
28 import requests
29 from requests.auth import AuthBase
30
31 py3k = False
32 if sys.version_info.major >= 3:
33     py3k = True
34     import urllib.parse
35     from ipaddress import ip_address, IPv4Address
36
37 ModuleNotFoundError = ImportError
38
39 try:
40     import rados
41 except ModuleNotFoundError as noModErr:
42     print(f"Error: {noModErr}\nExiting the script...")
43     sys.exit(1)
```

```
44
45 try:
46     import rbd
47 except ModuleNotFoundError as noModErr:
48     print(f"Error: {noModErr}\nExiting the script...")
49     sys.exit(1)
50
51 try:
52     # for 2.7.x
53     from StringIO import StringIO
54 except ModuleNotFoundError:
55     # for 3.x
56     from io import StringIO
57
58 try:
59     # for 2.7.x
60     from urlparse import urlparse
61     from urllib import urlencode as urlencode
62 except ModuleNotFoundError:
63     # for 3.x
64     from urllib.parse import urlparse
65     from urllib.parse import urlencode as urlencode
66
67 try:
68     from base64 import encodestring
```

```
except:
    from base64 import encodebytes as encodestring


class ExecutionFailureException(Exception):
    pass


##################################################
################## DummyRados ####################
##################################################
# this is mainly for testing and could be used where 'rados' is not
    ↪ available


class DummyRados(object):
    def __init__(self):
        self.return_val = 0
        self.err_message = ""
        self.state = "connected"
        self.cmd_output_map = {}
        self.cmd_names = {}
        self._init_cmd_output_map()
        self.dummy_host_ip_map = {}

```

```
93      def _init_cmd_output_map(self):
94          json_file_name = "test-data/ceph-status-out"
95          script_dir = path.abspath(path.dirname(__file__))
96          ceph_status_str = ""
97          with open(
98              path.join(script_dir, json_file_name), mode="r",
                ↪  encoding="UTF-8"
99          ) as json_file:
100             ceph_status_str = json_file.read()
101         self.cmd_names["fs ls"] = """{"format": "json", "prefix": "fs
            ↪  ls"}"""
102         self.cmd_names[
103             "quorum_status"
104         ] = """{"format": "json", "prefix": "quorum_status"}"""
105         self.cmd_names[
106             "mgr services"
107         ] = """{"format": "json", "prefix": "mgr services"}"""
108         # all the commands and their output
109         self.cmd_output_map[
110             self.cmd_names["fs ls"]
111         ] = """[{"name":"myfs","metadata_pool":"myfs-metadata","metad
            ↪  ata_pool_id":2,"data_pool_ids":[3],"data_pools":["myfs-re
            ↪  plicated"]}]"""
112         self.cmd_output_map[
113             self.cmd_names["quorum_status"]
```

```
114             ] = """{"election_epoch":3,"quorum":[0],"quorum_names":["a"],
                ↪  "quorum_leader_name":"a","quorum_age":14385,"features":{"
                ↪  quorum_con":"4540138292836696063","quorum_mon":["kraken",
                ↪  "luminous","mimic","osdmap-prune","nautilus","octopus"]},
                ↪  "monmap":{"epoch":1,"fsid":"af4e1673-0b72-402d-990a-22d29
                ↪  19d0f1c","modified":"2020-05-07T03:36:39.918035Z","create
                ↪  d":"2020-05-07T03:36:39.918035Z","min_mon_release":15,"mi
                ↪  n_mon_release_name":"octopus","features":{"persistent":["
                ↪  kraken","luminous","mimic","osdmap-prune","nautilus","oct
                ↪  opus"],"optional":[]},"mons":[{"rank":0,"name":"a","publi
                ↪  c_addrs":{"addrvec":[{"type":"v2","addr":"10.110.205.174:
                ↪  3300","nonce":0},{"type":"v1","addr":"10.110.205.174:6789
                ↪  ","nonce":0}]},"addr":"10.110.205.174:6789/0","public_add
                ↪  r":"10.110.205.174:6789/0","priority":0,"weight":0}]}}"""
115             self.cmd_output_map[
116                 self.cmd_names["mgr services"]
117             ] = """{"dashboard":"https://ceph-dashboard:8443/","prometheu
                ↪  s":"http://ceph-dashboard-db:9283/"}"""
118             self.cmd_output_map[
```

```
119             """{"caps": ["mon", "allow r, allow command
                ↪  quorum_status", "osd", "profile rbd-read-only, allow
                ↪  rwx pool=default.rgw.meta, allow r pool=.rgw.root,
                ↪  allow rw pool=default.rgw.control, allow x
                ↪  pool=default.rgw.buckets.index"], "entity":
                ↪  "client.healthchecker", "format": "json", "prefix":
                ↪  "auth get-or-create"}"""
120         ] = """[{"entity":"client.healthchecker","key":"AQDFkbNeft5bF
            ↪  RAATndLNUSEKruozxiZi3lrdA==","caps":{"mon":"allow r,
            ↪  allow command quorum_status","osd":"profile
            ↪  rbd-read-only, allow rwx pool=default.rgw.meta, allow r
            ↪  pool=.rgw.root, allow rw pool=default.rgw.control, allow
            ↪  x pool=default.rgw.buckets.index"}}]"""
121         self.cmd_output_map[
122             """{"caps": ["mon", "profile rbd, allow command 'osd
                ↪  blocklist'", "osd", "profile rbd"], "entity":
                ↪  "client.csi-rbd-node", "format": "json", "prefix":
                ↪  "auth get-or-create"}"""
123         ] = """[{"entity":"client.csi-rbd-node","key":"AQBOgrNeHbK1Ax
            ↪  AAubYBeV8S1U/GPzq5SVeq6g==","caps":{"mon":"profile rbd,
            ↪  allow command 'osd blocklist'","osd":"profile rbd"}}]"""
124         self.cmd_output_map[
```

```
125             """{"caps": ["mon", "profile rbd, allow command 'osd
                ↪  blocklist'", "mgr", "allow rw", "osd", "profile
                ↪  rbd"], "entity": "client.csi-rbd-provisioner",
                ↪  "format": "json", "prefix": "auth get-or-create"}"""
126         ] = """[{"entity":"client.csi-rbd-provisioner","key":"AQBNgrN
            ↪  e1geyKxAA8ekViRdE+hss5OweYBkwNg==","caps":{"mgr":"allow
            ↪  rw","mon":"profile rbd, allow command 'osd
            ↪  blocklist'","osd":"profile rbd"}}]"""
127         self.cmd_output_map[
128             """{"caps": ["mon", "allow r, allow command 'osd
                ↪  blocklist'", "mgr", "allow rw", "osd", "allow rw tag
                ↪  cephfs *=*", "mds", "allow rw"], "entity":
                ↪  "client.csi-cephfs-node", "format": "json",
                ↪  "prefix": "auth get-or-create"}"""
129         ] = """[{"entity":"client.csi-cephfs-node","key":"AQBOgrNeENu
            ↪  nKxAAPCmgE7R6G8DcXnaJ1F32qg==","caps":{"mds":"allow
            ↪  rw","mgr":"allow rw","mon":"allow r, allow command 'osd
            ↪  blocklist'","osd":"allow rw tag cephfs *=*"}}]"""
130         self.cmd_output_map[
131             """{"caps": ["mon", "allow r, allow command 'osd
                ↪  blocklist'", "mgr", "allow rw", "osd", "allow rw tag
                ↪  cephfs metadata=*"], "entity":
                ↪  "client.csi-cephfs-provisioner", "format": "json",
                ↪  "prefix": "auth get-or-create"}"""
```

```
132         ] = """[{"entity":"client.csi-cephfs-provisioner","key":"AQBO
            ↪ grNeAFgcGBAAvGqKOAD0D3xxmVY0R912dg==","caps":{"mgr":"allo
            ↪ w rw","mon":"allow r, allow command 'osd
            ↪ blocklist'","osd":"allow rw tag cephfs metadata=*"}}]"""
133         self.cmd_output_map[
134             """{"caps": ["mon", "allow r, allow command 'osd
                ↪ blocklist'", "mgr", "allow rw", "osd", "allow rw tag
                ↪ cephfs metadata=*"], "entity":
                ↪ "client.csi-cephfs-provisioner-openshift-storage",
                ↪ "format": "json", "prefix": "auth get-or-create"}"""
135         ] = """[{"entity":"client.csi-cephfs-provisioner-openshift-st
            ↪ orage","key":"BQBOgrNeAFgcGBAAvGqKOAD0D3xxmVY0R912dg==","
            ↪ caps":{"mgr":"allow rw","mon":"allow r, allow command
            ↪ 'osd blocklist'","osd":"allow rw tag cephfs
            ↪ metadata=*"}}]"""
136         self.cmd_output_map[
137             """{"caps": ["mon", "allow r, allow command 'osd
                ↪ blocklist'", "mgr", "allow rw", "osd", "allow rw tag
                ↪ cephfs metadata=myfs"], "entity": "client.csi-cephfs
                ↪ -provisioner-openshift-storage-myfs", "format":
                ↪ "json", "prefix": "auth get-or-create"}"""
```

```
138         ] = """[{"entity":"client.csi-cephfs-provisioner-openshift-st
            ↪ orage-myfs","key":"CQBOgrNeAFgcGBAAvGqKOAD0D3xxmVY0R912dg
            ↪ ==","caps":{"mgr":"allow rw","mon":"allow r, allow
            ↪ command 'osd blocklist'","osd":"allow rw tag cephfs
            ↪ metadata=myfs"}}]"""
139         self.cmd_output_map[
140             """{"caps": ["mon", "allow r, allow command
                ↪ quorum_status, allow command version", "mgr", "allow
                ↪ command config", "osd", "profile rbd-read-only,
                ↪ allow rwx pool=default.rgw.meta, allow r
                ↪ pool=.rgw.root, allow rw pool=default.rgw.control,
                ↪ allow rx pool=default.rgw.log, allow x
                ↪ pool=default.rgw.buckets.index"], "entity":
                ↪ "client.healthchecker", "format": "json", "prefix":
                ↪ "auth get-or-create"}"""
141         ] = """[{"entity":"client.healthchecker","key":"AQDFkbNeft5bF
            ↪ RAATndLNUSEKruozxiZi3lrdA==","caps":{"mon": "allow r,
            ↪ allow command quorum_status, allow command version",
            ↪ "mgr": "allow command config", "osd": "profile
            ↪ rbd-read-only, allow rwx pool=default.rgw.meta, allow r
            ↪ pool=.rgw.root, allow rw pool=default.rgw.control, allow
            ↪ rx pool=default.rgw.log, allow x
            ↪ pool=default.rgw.buckets.index"}}]"""
142         self.cmd_output_map[
```

```
143             """{"caps": ["mon", "allow r, allow command
                ↪  quorum_status, allow command version", "mgr", "allow
                ↪  command config", "osd", "profile rbd-read-only,
                ↪  allow rwx pool=default.rgw.meta, allow r
                ↪  pool=.rgw.root, allow rw pool=default.rgw.control,
                ↪  allow rx pool=default.rgw.log, allow x
                ↪  pool=default.rgw.buckets.index"], "entity":
                ↪  "client.healthchecker", "format": "json", "prefix":
                ↪  "auth caps"}"""
144         ] = """[{"entity":"client.healthchecker","key":"AQDFkbNeft5bF
                ↪  RAATndLNUSRKruozxiZi3lrdA==","caps":{"mon": "allow r,
                ↪  allow command quorum_status, allow command version",
                ↪  "mgr": "allow command config", "osd": "profile
                ↪  rbd-read-only, allow rwx pool=default.rgw.meta, allow r
                ↪  pool=.rgw.root, allow rw pool=default.rgw.control, allow
                ↪  rx pool=default.rgw.log, allow x
                ↪  pool=default.rgw.buckets.index"}}]"""
145         self.cmd_output_map[
146             """{"format": "json", "prefix": "mgr services"}"""
147         ] = """{"dashboard":
                ↪  "http://rook-ceph-mgr-a-57cf9f84bc-f4jnl:7000/",
                ↪  "prometheus":
                ↪  "http://rook-ceph-mgr-a-57cf9f84bc-f4jnl:9283/"}"""
148         self.cmd_output_map[
```

```
149             """{"entity": "client.healthchecker", "format": "json",
                ↪  "prefix": "auth get"}"""
150         ] = """{"dashboard":
                ↪  "http://rook-ceph-mgr-a-57cf9f84bc-f4jnl:7000/",
                ↪  "prometheus":
                ↪  "http://rook-ceph-mgr-a-57cf9f84bc-f4jnl:9283/"}"""
151         self.cmd_output_map[
152             """{"entity": "client.healthchecker", "format": "json",
                ↪  "prefix": "auth get"}"""
153         ] = """[{"entity":"client.healthchecker","key":"AQDFkbNeft5bF
                ↪  RAATndLNUSEKruozxiZi3lrdA==","caps":{"mon": "allow r,
                ↪  allow command quorum_status, allow command version",
                ↪  "mgr": "allow command config", "osd": "profile
                ↪  rbd-read-only, allow rwx pool=default.rgw.meta, allow r
                ↪  pool=.rgw.root, allow rw pool=default.rgw.control, allow
                ↪  rx pool=default.rgw.log, allow x
                ↪  pool=default.rgw.buckets.index"}}]"""
154         self.cmd_output_map[
155             """{"entity": "client.csi-cephfs-node", "format":
                ↪  "json", "prefix": "auth get"}"""
156         ] = """[]"""
157         self.cmd_output_map[
158             """{"entity": "client.csi-rbd-node", "format": "json",
                ↪  "prefix": "auth get"}"""
159         ] = """[]"""
```

```
160             self.cmd_output_map[
161                 """{"entity": "client.csi-rbd-provisioner", "format":
                    ↪  "json", "prefix": "auth get"}"""
162             ] = """[]"""
163             self.cmd_output_map[
164                 """{"entity": "client.csi-cephfs-provisioner", "format":
                    ↪  "json", "prefix": "auth get"}"""
165             ] = """[]"""
166             self.cmd_output_map[
167                 """{"entity":
                    ↪  "client.csi-cephfs-provisioner-openshift-storage",
                    ↪  "format": "json", "prefix": "auth get"}"""
168             ] = """[]"""
169             self.cmd_output_map[
170                 """{"entity": "client.csi-cephfs-provisioner-openshift-s
                    ↪  torage-myfs", "format": "json", "prefix": "auth
                    ↪  get"}"""
171             ] = """[]"""
172             self.cmd_output_map[
173                 """{"entity": "client.csi-cephfs-provisioner", "format":
                    ↪  "json", "prefix": "auth get"}"""
174             ] = """[{"entity":"client.csi-cephfs-provisioner","key":"AQDF
                ↪  kbNeft5bFRAATndLNUSEKruozxiZi3lrdA==","caps":{"mon":"allo
                ↪  w r", "mgr":"allow rw", "osd":"allow rw tag cephfs
                ↪  metadata=*"}}]"""
```

```
175             self.cmd_output_map[
176                 """{"caps": ["mon", "allow r, allow command 'osd
                    ↪  blocklist'", "mgr", "allow rw", "osd", "allow rw tag
                    ↪  cephfs metadata=*"], "entity":
                    ↪  "client.csi-cephfs-provisioner", "format": "json",
                    ↪  "prefix": "auth caps"}"""
177             ] = """[{"entity":"client.csi-cephfs-provisioner","key":"AQDF
                ↪  kbNeft5bFRAATndLNUSEKruozxiZi3lrdA==","caps":{"mon":"allo
                ↪  w r,  allow command 'osd blocklist'", "mgr":"allow rw",
                ↪  "osd":"allow rw tag cephfs metadata=*"}}]"""
178             self.cmd_output_map['{"format": "json", "prefix": "status"}']
                ↪  = ceph_status_str
179
180         def shutdown(self):
181             pass
182
183         def get_fsid(self):
184             return "af4e1673-0b72-402d-990a-22d2919d0f1c"
185
186         def conf_read_file(self):
187             pass
188
189         def connect(self):
190             pass
191
```

```
192     def pool_exists(self, pool_name):
193         return True
194 
195     def mon_command(self, cmd, out):
196         json_cmd = json.loads(cmd)
197         json_cmd_str = json.dumps(json_cmd, sort_keys=True)
198         cmd_output = self.cmd_output_map[json_cmd_str]
199         return self.return_val, cmd_output,
            ↪  str(self.err_message.encode("utf-8"))
200 
201     def _convert_hostname_to_ip(self, host_name):
202         ip_reg_x = re.compile(r"\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}")
203         # if provided host is directly an IP address, return the
            ↪  same
204         if ip_reg_x.match(host_name):
205             return host_name
206         import random
207 
208         host_ip = self.dummy_host_ip_map.get(host_name, "")
209         if not host_ip:
210             host_ip = f"172.9.{random.randint(0,
                ↪  254)}.{random.randint(0, 254)}"
211             self.dummy_host_ip_map[host_name] = host_ip
212         del random
213         return host_ip
```

```
214
215     @classmethod
216     def Rados(conffile=None):
217         return DummyRados()
218
219
220 class S3Auth(AuthBase):
221
222     """Attaches AWS Authentication to the given Request object."""
223
224     service_base_url = "s3.amazonaws.com"
225
226     def __init__(self, access_key, secret_key, service_url=None):
227         if service_url:
228             self.service_base_url = service_url
229         self.access_key = str(access_key)
230         self.secret_key = str(secret_key)
231
232     def __call__(self, r):
233         # Create date header if it is not created yet.
234         if "date" not in r.headers and "x-amz-date" not in r.headers:
235             r.headers["date"] = formatdate(timeval=None,
                ↪  localtime=False, usegmt=True)
236         signature = self.get_signature(r)
237         if py3k:
```

```
238             signature = signature.decode("utf-8")
239         r.headers["Authorization"] = f"AWS
            ↪  {self.access_key}:{signature}"
240         return r
241
242     def get_signature(self, r):
243         canonical_string = self.get_canonical_string(r.url,
            ↪  r.headers, r.method)
244         if py3k:
245             key = self.secret_key.encode("utf-8")
246             msg = canonical_string.encode("utf-8")
247         else:
248             key = self.secret_key
249             msg = canonical_string
250         h = hmac.new(key, msg, digestmod=sha)
251         return encodestring(h.digest()).strip()
252
253     def get_canonical_string(self, url, headers, method):
254         parsedurl = urlparse(url)
255         objectkey = parsedurl.path[1:]
256
257         bucket = parsedurl.netloc[: -len(self.service_base_url)]
258         if len(bucket) > 1:
259             # remove last dot
260             bucket = bucket[:-1]
```

```
261
262         interesting_headers = {"content-md5": "", "content-type": "",
            ↪  "date": ""}
263         for key in headers:
264             lk = key.lower()
265             try:
266                 lk = lk.decode("utf-8")
267             except:
268                 pass
269             if headers[key] and (
270                 lk in interesting_headers.keys() or
                    ↪  lk.startswith("x-amz-")
271             ):
272                 interesting_headers[lk] = headers[key].strip()
273
274         # If x-amz-date is used it supersedes the date header.
275         if not py3k:
276             if "x-amz-date" in interesting_headers:
277                 interesting_headers["date"] = ""
278         else:
279             if "x-amz-date" in interesting_headers:
280                 interesting_headers["date"] = ""
281
282         buf = f"{method}\n"
283         for key in sorted(interesting_headers.keys()):
```

```
284                 val = interesting_headers[key]
285                 if key.startswith("x-amz-"):
286                     buf += f"{key}:{val}\n"
287                 else:
288                     buf += f"{val}\n"
289 
290             # append the bucket if it exists
291             if bucket != "":
292                 buf += f"/{bucket}"
293 
294             # add the objectkey. even if it doesn't exist, add the slash
295             buf += f"/{objectkey}"
296 
297             return buf
298 
299 
300 class RadosJSON:
301     EXTERNAL_USER_NAME = "client.healthchecker"
302     EXTERNAL_RGW_ADMIN_OPS_USER_NAME = "rgw-admin-ops-user"
303     EMPTY_OUTPUT_LIST = "Empty output list"
304     DEFAULT_RGW_POOL_PREFIX = "default"
305     DEFAULT_MONITORING_ENDPOINT_PORT = "9283"
306 
307     @classmethod
308     def gen_arg_parser(cls, args_to_parse=None):
```

```
309         argP = argparse.ArgumentParser()
310
311         common_group = argP.add_argument_group("common")
312         common_group.add_argument("--verbose", "-v",
            ↪  action="store_true", default=False)
313         common_group.add_argument(
314             "--ceph-conf", "-c", help="Provide a ceph conf file.",
                ↪  type=str
315         )
316         common_group.add_argument(
317             "--keyring", "-k", help="Path to ceph keyring file.",
                ↪  type=str
318         )
319         common_group.add_argument(
320             "--run-as-user",
321             "-u",
322             default="",
323             type=str,
324             help="Provides a user name to check the cluster's health
                ↪  status, must be prefixed by 'client.'",
325         )
326         common_group.add_argument(
327             "--k8s-cluster-name", default="", help="Kubernetes
                ↪  cluster name"
328         )
```

```
329         common_group.add_argument(
330             "--namespace",
331             default="",
332             help="Namespace where CephCluster is running",
333         )
334         common_group.add_argument(
335             "--rgw-pool-prefix", default="", help="RGW Pool prefix"
336         )
337         common_group.add_argument(
338             "--restricted-auth-permission",
339             default=False,
340             help="Restrict cephCSIKeyrings auth permissions to
                 ↪  specific pools, cluster."
341             + "Mandatory flags that need to be set are
                 ↪  --rbd-data-pool-name, and --k8s-cluster-name."
342             + "--cephfs-filesystem-name flag can also be passed in
                 ↪  case of cephfs user restriction, so it can restrict
                 ↪  user to particular cephfs filesystem"
343             + "sample run: `python3
                 ↪  /etc/ceph/create-external-cluster-resources.py
                 ↪  --cephfs-filesystem-name myfs --rbd-data-pool-name
                 ↪  replicapool --k8s-cluster-name rookstorage
                 ↪  --restricted-auth-permission true`"
```

```
344             + "Note: Restricting the csi-users per pool, and per
                ↪  cluster will require creating new csi-users and new
                ↪  secrets for that csi-users."
345             + "So apply these secrets only to new `Consumer cluster`
                ↪  deployment while using the same `Source cluster`.",
346         )
347         common_group.add_argument(
348             "--v2-port-enable",
349             action="store_true",
350             default=False,
351             help="Enable v2 mon port(3300) for mons",
352         )
353
354         output_group = argP.add_argument_group("output")
355         output_group.add_argument(
356             "--format",
357             "-t",
358             choices=["json", "bash"],
359             default="json",
360             help="Provides the output format (json | bash)",
361         )
362         output_group.add_argument(
363             "--output",
364             "-o",
365             default="",
```

```
366             help="Output will be stored into the provided file",
367         )
368         output_group.add_argument(
369             "--cephfs-filesystem-name",
370             default="",
371             help="Provides the name of the Ceph filesystem",
372         )
373         output_group.add_argument(
374             "--cephfs-metadata-pool-name",
375             default="",
376             help="Provides the name of the cephfs metadata pool",
377         )
378         output_group.add_argument(
379             "--cephfs-data-pool-name",
380             default="",
381             help="Provides the name of the cephfs data pool",
382         )
383         output_group.add_argument(
384             "--rbd-data-pool-name",
385             default="",
386             required=False,
387             help="Provides the name of the RBD datapool",
388         )
389         output_group.add_argument(
390             "--alias-rbd-data-pool-name",
```

```
391             default="",
392             required=False,
393             help="Provides an alias for the  RBD data pool name,
                ↪  necessary if a special character is present in the
                ↪  pool name such as a period or underscore",
394         )
395         output_group.add_argument(
396             "--rgw-endpoint",
397             default="",
398             required=False,
399             help="RADOS Gateway endpoint (in `<IPv4>:<PORT>` or
                ↪  `<[IPv6]>:<PORT>` or `<FQDN>:<PORT>` format)",
400         )
401         output_group.add_argument(
402             "--rgw-tls-cert-path",
403             default="",
404             required=False,
405             help="RADOS Gateway endpoint TLS certificate",
406         )
407         output_group.add_argument(
408             "--rgw-skip-tls",
409             required=False,
410             default=False,
411             help="Ignore TLS certification validation when a
                ↪  self-signed certificate is provided (NOT RECOMMENDED",
```

```
412             )
413             output_group.add_argument(
414                 "--monitoring-endpoint",
415                 default="",
416                 required=False,
417                 help="Ceph Manager prometheus exporter endpoints (comma
                    ↪  separated list of (format `<IPv4>` or `<[IPv6]>` or
                    ↪  `<FQDN>`) entries of active and standby mgrs)",
418             )
419             output_group.add_argument(
420                 "--monitoring-endpoint-port",
421                 default="",
422                 required=False,
423                 help="Ceph Manager prometheus exporter port",
424             )
425             output_group.add_argument(
426                 "--skip-monitoring-endpoint",
427                 default=False,
428                 action="store_true",
429                 help="Do not check for a monitoring endpoint for the Ceph
                    ↪  cluster",
430             )
431             output_group.add_argument(
432                 "--rbd-metadata-ec-pool-name",
433                 default="",
```

```
434             required=False,
435             help="Provides the name of erasure coded RBD metadata
                ↪  pool",
436         )
437         output_group.add_argument(
438             "--dry-run",
439             default=False,
440             action="store_true",
441             help="Dry run prints the executed commands without
                ↪  running them",
442         )
443         output_group.add_argument(
444             "--rados-namespace",
445             default="",
446             required=False,
447             help="divides a pool into separate logical namespaces",
448         )
449         output_group.add_argument(
450             "--subvolume-group",
451             default="",
452             required=False,
453             help="provides the name of the subvolume group",
454         )
455         output_group.add_argument(
456             "--rgw-realm-name",
```

```
457             default="",
458             required=False,
459             help="provides the name of the rgw-realm",
460         )
461         output_group.add_argument(
462             "--rgw-zone-name",
463             default="",
464             required=False,
465             help="provides the name of the rgw-zone",
466         )
467         output_group.add_argument(
468             "--rgw-zonegroup-name",
469             default="",
470             required=False,
471             help="provides the name of the rgw-zonegroup",
472         )
473 
474         upgrade_group = argP.add_argument_group("upgrade")
475         upgrade_group.add_argument(
476             "--upgrade",
477             action="store_true",
478             default=False,
```

```
479             help="Upgrades the cephCSIKeyrings(For example:
                ↪  client.csi-cephfs-provisioner) and
                ↪  client.healthchecker ceph users with new permissions
                ↪  needed for the new cluster version and older
                ↪  permission will still be applied."
480           + "Sample run: `python3
                ↪  /etc/ceph/create-external-cluster-resources.py
                ↪  --upgrade`, this will upgrade all the default csi
                ↪  users(non-restricted)"
481           + "For restricted users(For example: client.csi-cephfs-pr
                ↪  ovisioner-openshift-storage-myfs), users created
                ↪  using --restricted-auth-permission flag need to pass
                ↪  mandatory flags"
482           + "mandatory flags: '--rbd-data-pool-name,
                ↪  --k8s-cluster-name and --run-as-user' flags while
                ↪  upgrading"
483           + "in case of cephfs users if you have passed
                ↪  --cephfs-filesystem-name flag while creating user
                ↪  then while upgrading it will be mandatory too"
484           + "Sample run: `python3
                ↪  /etc/ceph/create-external-cluster-resources.py
                ↪  --upgrade --rbd-data-pool-name replicapool
                ↪  --k8s-cluster-name rookstorage  --run-as-user
                ↪  client.csi-rbd-node-rookstorage-replicapool`"
```

```
485             + "PS: An existing non-restricted user cannot be
                ↪  converted to a restricted user by upgrading."
486             + "Upgrade flag should only be used to append new
                ↪  permissions to users, it shouldn't be used for
                ↪  changing user already applied permission, for example
                ↪  you shouldn't change in which pool user has access",
487         )
488
489         if args_to_parse:
490             assert (
491                 type(args_to_parse) == list
492             ), "Argument to 'gen_arg_parser' should be a list"
493         else:
494             args_to_parse = sys.argv[1:]
495         return argP.parse_args(args_to_parse)
496
497     def validate_rbd_metadata_ec_pool_name(self):
498         if self._arg_parser.rbd_metadata_ec_pool_name:
499             rbd_metadata_ec_pool_name =
                ↪  self._arg_parser.rbd_metadata_ec_pool_name
500             rbd_pool_name = self._arg_parser.rbd_data_pool_name
501
502             if rbd_pool_name == "":
503                 raise ExecutionFailureException(
504                     "Flag '--rbd-data-pool-name' should not be empty"
```

```
505                     )
506
507                 if rbd_metadata_ec_pool_name == "":
508                     raise ExecutionFailureException(
509                         "Flag '--rbd-metadata-ec-pool-name' should not be
                            ↪  empty"
510                     )
511
512                 cmd_json = {"prefix": "osd dump", "format": "json"}
513                 ret_val, json_out, err_msg =
                    ↪  self._common_cmd_json_gen(cmd_json)
514                 if ret_val != 0 or len(json_out) == 0:
515                     raise ExecutionFailureException(
516                         f"{cmd_json['prefix']} command failed.\n"
517                         f"Error: {err_msg if ret_val != 0 else
                            ↪  self.EMPTY_OUTPUT_LIST}"
518                     )
519                 metadata_pool_exist, pool_exist = False, False
520
521                 for key in json_out["pools"]:
522                     # if erasure_code_profile is empty and pool name
                        ↪  exists then it replica pool
523                     if (
524                         key["erasure_code_profile"] == ""
525                         and key["pool_name"] == rbd_metadata_ec_pool_name
```

```
526                 ):
527                     metadata_pool_exist = True
528                 # if erasure_code_profile is not empty and pool name
                    ↪  exists then it is ec pool
529                 if key["erasure_code_profile"] and key["pool_name"]
                    ↪  == rbd_pool_name:
530                     pool_exist = True
531
532             if not metadata_pool_exist:
533                 raise ExecutionFailureException(
534                     "Provided rbd_ec_metadata_pool name,"
535                     f" {rbd_metadata_ec_pool_name}, does not exist"
536                 )
537             if not pool_exist:
538                 raise ExecutionFailureException(
539                     f"Provided rbd_data_pool name, {rbd_pool_name},
                        ↪  does not exist"
540                 )
541             return rbd_metadata_ec_pool_name
542
543     def dry_run(self, msg):
544         if self._arg_parser.dry_run:
545             print("Execute: " + "'" + msg + "'")
546
547     def validate_rgw_endpoint_tls_cert(self):
```

```
548             if self._arg_parser.rgw_tls_cert_path:
549                 with open(self._arg_parser.rgw_tls_cert_path,
                    ↪  encoding="utf8") as f:
550                     contents = f.read()
551                     return contents.rstrip()
552
553         def _check_conflicting_options(self):
554             if not self._arg_parser.upgrade and not
                    ↪  self._arg_parser.rbd_data_pool_name:
555                 raise ExecutionFailureException(
556                     "Either '--upgrade' or '--rbd-data-pool-name
                            ↪  <pool_name>' should be specified"
557                 )
558
559         def _invalid_endpoint(self, endpoint_str):
560             # separating port, by getting last split of `:` delimiter
561             try:
562                 endpoint_str_ip, port = endpoint_str.rsplit(":", 1)
563             except ValueError:
564                 raise ExecutionFailureException(f"Not a proper endpoint:
                        ↪  {endpoint_str}")
565
566             try:
567                 if endpoint_str_ip[0] == "[":
```

```
568                 endpoint_str_ip = endpoint_str_ip[1 :
                    ↪  len(endpoint_str_ip) - 1]
569             ip_type = (
570                 "IPv4" if type(ip_address(endpoint_str_ip)) is
                    ↪  IPv4Address else "IPv6"
571             )
572         except ValueError:
573             ip_type = "FQDN"
574         if not port.isdigit():
575             raise ExecutionFailureException(f"Port not valid: {port}")
576         intPort = int(port)
577         if intPort < 1 or intPort > 2**16 - 1:
578             raise ExecutionFailureException(f"Out of range port
                ↪  number: {port}")
579 
580         return ip_type
581 
582     def endpoint_dial(self, endpoint_str, ip_type, timeout=3,
        ↪  cert=None):
583         # if the 'cluster' instance is a dummy one,
584         # don't try to reach out to the endpoint
585         if isinstance(self.cluster, DummyRados):
586             return "", "", ""
587         if ip_type == "IPv6":
588             try:
```

```
589                 endpoint_str_ip, endpoint_str_port =
                    ↪  endpoint_str.rsplit(":", 1)
590             except ValueError:
591                 raise ExecutionFailureException(
592                     f"Not a proper endpoint: {endpoint_str}"
593                 )
594             if endpoint_str_ip[0] != "[":
595                 endpoint_str_ip = "[" + endpoint_str_ip + "]"
596             endpoint_str = ":".join([endpoint_str_ip,
                ↪  endpoint_str_port])
597
598         protocols = ["http", "https"]
599         response_error = None
600         for prefix in protocols:
601             try:
602                 ep = f"{prefix}://{endpoint_str}"
603                 verify = None
604                 # If verify is set to a path to a directory,
605                 # the directory must have been processed using the
                    ↪  c_rehash utility supplied with OpenSSL.
606                 if prefix == "https" and
                    ↪  self._arg_parser.rgw_skip_tls:
607                     verify = False
608                     r = requests.head(ep, timeout=timeout,
                        ↪  verify=False)
```

```
609                 elif prefix == "https" and cert:
610                     verify = cert
611                     r = requests.head(ep, timeout=timeout,
                        ↪  verify=cert)
612                 else:
613                     r = requests.head(ep, timeout=timeout)
614                 if r.status_code == 200:
615                     return prefix, verify, ""
616             except Exception as err:
617                 response_error = err
618                 continue
619         sys.stderr.write(
620             f"unable to connect to endpoint: {endpoint_str}, failed
                ↪  error: {response_error}"
621         )
622         return (
623             "",
624             "",
625             ("-1"),
626         )
627
628     def __init__(self, arg_list=None):
629         self.out_map = {}
630         self._excluded_keys = set()
631         self._arg_parser = self.gen_arg_parser(args_to_parse=arg_list)
```

```
632             self._check_conflicting_options()
633             self.run_as_user = self._arg_parser.run_as_user
634             self.output_file = self._arg_parser.output
635             self.ceph_conf = self._arg_parser.ceph_conf
636             self.ceph_keyring = self._arg_parser.keyring
637             # if user not provided, give a default user
638             if not self.run_as_user and not self._arg_parser.upgrade:
639                 self.run_as_user = self.EXTERNAL_USER_NAME
640             if not self._arg_parser.rgw_pool_prefix and not
                ↪  self._arg_parser.upgrade:
641                 self._arg_parser.rgw_pool_prefix =
                    ↪  self.DEFAULT_RGW_POOL_PREFIX
642             if self.ceph_conf:
643                 kwargs = {}
644                 if self.ceph_keyring:
645                     kwargs["conf"] = {"keyring": self.ceph_keyring}
646                 self.cluster = rados.Rados(conffile=self.ceph_conf,
                    ↪  **kwargs)
647             else:
648                 self.cluster = rados.Rados()
649                 self.cluster.conf_read_file()
650             self.cluster.connect()
651
652         def shutdown(self):
653             if self.cluster.state == "connected":
```

```
654             self.cluster.shutdown()
655
656     def get_fsid(self):
657         if self._arg_parser.dry_run:
658             return self.dry_run("ceph fsid")
659         return str(self.cluster.get_fsid())
660
661     def _common_cmd_json_gen(self, cmd_json):
662         cmd = json.dumps(cmd_json, sort_keys=True)
663         ret_val, cmd_out, err_msg = self.cluster.mon_command(cmd, b"")
664         if self._arg_parser.verbose:
665             print(f"Command Input: {cmd}")
666             print(
667                 f"Return Val: {ret_val}\nCommand Output: {cmd_out}\n"
668                 f"Error Message: {err_msg}\n----------\n"
669             )
670         json_out = {}
671         # if there is no error (i.e; ret_val is ZERO) and 'cmd_out'
            ↪  is not empty
672         # then convert 'cmd_out' to a json output
673         if ret_val == 0 and cmd_out:
674             json_out = json.loads(cmd_out)
675         return ret_val, json_out, err_msg
676
677     def get_ceph_external_mon_data(self):
```

```
678             cmd_json = {"prefix": "quorum_status", "format": "json"}
679             if self._arg_parser.dry_run:
680                 return self.dry_run("ceph " + cmd_json["prefix"])
681             ret_val, json_out, err_msg =
                ↪  self._common_cmd_json_gen(cmd_json)
682             # if there is an unsuccessful attempt,
683             if ret_val != 0 or len(json_out) == 0:
684                 raise ExecutionFailureException(
685                     "'quorum_status' command failed.\n"
686                     f"Error: {err_msg if ret_val != 0 else
                        ↪  self.EMPTY_OUTPUT_LIST}"
687                 )
688             q_leader_name = json_out["quorum_leader_name"]
689             q_leader_details = {}
690             q_leader_matching_list = [
691                 l for l in json_out["monmap"]["mons"] if l["name"] ==
                    ↪  q_leader_name
692             ]
693             if len(q_leader_matching_list) == 0:
694                 raise ExecutionFailureException("No matching 'mon'
                    ↪  details found")
695             q_leader_details = q_leader_matching_list[0]
696             # get the address vector of the quorum-leader
697             q_leader_addrvec = q_leader_details.get("public_addrs",
                ↪  {}).get("addrvec", [])
```

```
698             # if the quorum-leader has only one address in the
                ↪  address-vector
699             # and it is of type 'v2' (ie; with <IP>:3300),
700             # raise an exception to make user aware that
701             # they have to enable 'v1' (ie; with <IP>:6789) type as well
702             if len(q_leader_addrvec) == 1 and q_leader_addrvec[0]["type"]
                ↪  == "v2":
703                 raise ExecutionFailureException(
704                     "Only 'v2' address type is enabled, user should also
                        ↪  enable 'v1' type as well"
705                 )
706             ip_addr = str(q_leader_details["public_addr"].split("/")[0])
707
708             if self._arg_parser.v2_port_enable:
709                 if len(q_leader_addrvec) > 1:
710                     if q_leader_addrvec[0]["type"] == "v2":
711                         ip_addr = q_leader_addrvec[0]["addr"]
712                     elif q_leader_addrvec[1]["type"] == "v2":
713                         ip_addr = q_leader_addrvec[1]["addr"]
714                 else:
715                     sys.stderr.write(
716                         "'v2' address type not present, and
                            ↪  'v2-port-enable' flag is provided"
717                     )
718
```

```
719             return f"{str(q_leader_name)}={ip_addr}"
720
721         def _convert_hostname_to_ip(self, host_name, port, ip_type):
722             # if 'cluster' instance is a dummy type,
723             # call the dummy instance's "convert" method
724             if not host_name:
725                 raise ExecutionFailureException("Empty hostname provided")
726             if isinstance(self.cluster, DummyRados):
727                 return self.cluster._convert_hostname_to_ip(host_name)
728
729             if ip_type == "FQDN":
730                 # check which ip FQDN should be converted to, IPv4 or
                    ↪ IPv6
731                 # check the host ip, the endpoint ip type would be
                    ↪ similar to host ip
732                 cmd_json = {"prefix": "orch host ls", "format": "json"}
733                 ret_val, json_out, err_msg =
                    ↪ self._common_cmd_json_gen(cmd_json)
734                 # if there is an unsuccessful attempt,
735                 if ret_val != 0 or len(json_out) == 0:
736                     raise ExecutionFailureException(
737                         "'orch host ls' command failed.\n"
738                         f"Error: {err_msg if ret_val != 0 else
                            ↪ self.EMPTY_OUTPUT_LIST}"
739                     )
```

```
740             host_addr = json_out[0]["addr"]
741             # add :80 sample port in ip_type, as _invalid_endpoint
                ↪  also verify port
742             host_ip_type = self._invalid_endpoint(host_addr + ":80")
743             import socket
744
745             # example output [(<AddressFamily.AF_INET: 2>,
                ↪  <SocketKind.SOCK_STREAM: 1>, 6, '',
                ↪  ('93.184.216.34', 80)), ...]
746             # we need to get 93.184.216.34 so it would be
                ↪  ip[0][4][0]
747             if host_ip_type == "IPv6":
748                 ip = socket.getaddrinfo(
749                     host_name, port, family=socket.AF_INET6,
                        ↪  proto=socket.IPPROTO_TCP
750                 )
751             elif host_ip_type == "IPv4":
752                 ip = socket.getaddrinfo(
753                     host_name, port, family=socket.AF_INET,
                        ↪  proto=socket.IPPROTO_TCP
754                 )
755             del socket
756             return ip[0][4][0]
757         return host_name
758
```

```
759     def get_active_and_standby_mgrs(self):
760         if self._arg_parser.dry_run:
761             return "", self.dry_run("ceph status")
762         monitoring_endpoint_port =
            ↪  self._arg_parser.monitoring_endpoint_port
763         monitoring_endpoint_ip_list =
            ↪  self._arg_parser.monitoring_endpoint
764         standby_mgrs = []
765         if not monitoring_endpoint_ip_list:
766             cmd_json = {"prefix": "status", "format": "json"}
767             ret_val, json_out, err_msg =
                ↪  self._common_cmd_json_gen(cmd_json)
768             # if there is an unsuccessful attempt,
769             if ret_val != 0 or len(json_out) == 0:
770                 raise ExecutionFailureException(
771                     "'mgr services' command failed.\n"
772                     f"Error: {err_msg if ret_val != 0 else
                        ↪  self.EMPTY_OUTPUT_LIST}"
773                 )
774             monitoring_endpoint = (
775                 json_out.get("mgrmap", {}).get("services",
                    ↪  {}).get("prometheus", "")
776             )
777             if not monitoring_endpoint:
778                 raise ExecutionFailureException(
```

```
779                     "can't find monitoring_endpoint, prometheus
                        ↪  module might not be enabled, "
780                     "enable the module by running 'ceph mgr module
                        ↪  enable prometheus'"
781                 )
782             # now check the stand-by mgr-s
783             standby_arr = json_out.get("mgrmap", {}).get("standbys",
                ↪  [])
784             for each_standby in standby_arr:
785                 if "name" in each_standby.keys():
786                     standby_mgrs.append(each_standby["name"])
787             try:
788                 parsed_endpoint = urlparse(monitoring_endpoint)
789             except ValueError:
790                 raise ExecutionFailureException(
791                     f"invalid endpoint: {monitoring_endpoint}"
792                 )
793             monitoring_endpoint_ip_list = parsed_endpoint.hostname
794             if not monitoring_endpoint_port:
795                 monitoring_endpoint_port = str(parsed_endpoint.port)
796
797         # if monitoring endpoint port is not set, put a default mon
            ↪  port
798         if not monitoring_endpoint_port:
```

```
799             monitoring_endpoint_port =
                ↪  self.DEFAULT_MONITORING_ENDPOINT_PORT
800
801         # user could give comma and space separated inputs (like
            ↪  --monitoring-endpoint="<ip1>, <ip2>")
802         monitoring_endpoint_ip_list =
            ↪  monitoring_endpoint_ip_list.replace(",", " ")
803         monitoring_endpoint_ip_list_split =
            ↪  monitoring_endpoint_ip_list.split()
804         # if monitoring-endpoint could not be found, raise an error
805         if len(monitoring_endpoint_ip_list_split) == 0:
806             raise ExecutionFailureException("No 'monitoring-endpoint'
                ↪  found")
807         # first ip is treated as the main monitoring-endpoint
808         monitoring_endpoint_ip = monitoring_endpoint_ip_list_split[0]
809         # rest of the ip-s are added to the 'standby_mgrs' list
810         standby_mgrs.extend(monitoring_endpoint_ip_list_split[1:])
811         failed_ip = monitoring_endpoint_ip
812
813         monitoring_endpoint = ":".join(
814             [monitoring_endpoint_ip, monitoring_endpoint_port]
815         )
816         ip_type = self._invalid_endpoint(monitoring_endpoint)
817         try:
818             monitoring_endpoint_ip = self._convert_hostname_to_ip(
```

```
819                 monitoring_endpoint_ip, monitoring_endpoint_port,
                    ↪  ip_type
820             )
821             # collect all the 'stand-by' mgr ips
822             mgr_ips = []
823             for each_standby_mgr in standby_mgrs:
824                 failed_ip = each_standby_mgr
825                 mgr_ips.append(
826                     self._convert_hostname_to_ip(
827                         each_standby_mgr, monitoring_endpoint_port,
                        ↪  ip_type
828                     )
829                 )
830         except:
831             raise ExecutionFailureException(
832                 f"Conversion of host: {failed_ip} to IP failed. "
833                 "Please enter the IP addresses of all the ceph-mgrs
                    ↪  with the '--monitoring-endpoint' flag"
834             )
835 
836         _, _, err = self.endpoint_dial(monitoring_endpoint, ip_type)
837         if err == "-1":
838             raise ExecutionFailureException(err)
839         # add the validated active mgr IP into the first index
840         mgr_ips.insert(0, monitoring_endpoint_ip)
```

```
841         all_mgr_ips_str = ",".join(mgr_ips)
842         return all_mgr_ips_str, monitoring_endpoint_port
843
844     def check_user_exist(self, user):
845         cmd_json = {"prefix": "auth get", "entity": f"{user}",
            ↪  "format": "json"}
846         ret_val, json_out, _ = self._common_cmd_json_gen(cmd_json)
847         if ret_val != 0 or len(json_out) == 0:
848             return ""
849         return str(json_out[0]["key"])
850
851     def get_cephfs_provisioner_caps_and_entity(self):
852         entity = "client.csi-cephfs-provisioner"
853         caps = {
854             "mon": "allow r, allow command 'osd blocklist'",
855             "mgr": "allow rw",
856             "osd": "allow rw tag cephfs metadata=*",
857         }
858         if self._arg_parser.restricted_auth_permission:
859             k8s_cluster_name = self._arg_parser.k8s_cluster_name
860             if k8s_cluster_name == "":
861                 raise ExecutionFailureException(
862                     "k8s_cluster_name not found, please set the
                        ↪  '--k8s-cluster-name' flag"
863                 )
```

```
864             cephfs_filesystem =
                ↪  self._arg_parser.cephfs_filesystem_name
865             if cephfs_filesystem == "":
866                 entity = f"{entity}-{k8s_cluster_name}"
867             else:
868                 entity =
                    ↪  f"{entity}-{k8s_cluster_name}-{cephfs_filesystem}"
869                 caps["osd"] = f"allow rw tag cephfs
                    ↪  metadata={cephfs_filesystem}"
870
871         return caps, entity
872
873     def get_cephfs_node_caps_and_entity(self):
874         entity = "client.csi-cephfs-node"
875         caps = {
876             "mon": "allow r, allow command 'osd blocklist'",
877             "mgr": "allow rw",
878             "osd": "allow rw tag cephfs *=*",
879             "mds": "allow rw",
880         }
881         if self._arg_parser.restricted_auth_permission:
882             k8s_cluster_name = self._arg_parser.k8s_cluster_name
883             if k8s_cluster_name == "":
884                 raise ExecutionFailureException(
```

```
885                     "k8s_cluster_name not found, please set the
                        ↪  '--k8s-cluster-name' flag"
886                 )
887             cephfs_filesystem =
                ↪  self._arg_parser.cephfs_filesystem_name
888             if cephfs_filesystem == "":
889                 entity = f"{entity}-{k8s_cluster_name}"
890             else:
891                 entity =
                    ↪  f"{entity}-{k8s_cluster_name}-{cephfs_filesystem}"
892                 caps["osd"] = f"allow rw tag cephfs
                    ↪  *={cephfs_filesystem}"
893
894         return caps, entity
895
896     def get_entity(self, entity, rbd_pool_name, alias_rbd_pool_name,
        ↪  k8s_cluster_name):
897         if (
898             rbd_pool_name.count(".") != 0
899             or rbd_pool_name.count("_") != 0
900             or alias_rbd_pool_name != ""
901             # checking alias_rbd_pool_name is not empty as there
                ↪  maybe a special character used other than . or _
902         ):
903             if alias_rbd_pool_name == "":
```

```
904                 raise ExecutionFailureException(
905                     "please set the '--alias-rbd-data-pool-name' flag
                        ↪  as the rbd data pool name contains '.' or '_'"
906                 )
907             if (
908                 alias_rbd_pool_name.count(".") != 0
909                 or alias_rbd_pool_name.count("_") != 0
910             ):
911                 raise ExecutionFailureException(
912                     "'--alias-rbd-data-pool-name' flag value should
                        ↪  not contain '.' or '_'"
913                 )
914             entity =
                ↪  f"{entity}-{k8s_cluster_name}-{alias_rbd_pool_name}"
915         else:
916             entity = f"{entity}-{k8s_cluster_name}-{rbd_pool_name}"
917 
918         return entity
919 
920     def get_rbd_provisioner_caps_and_entity(self):
921         entity = "client.csi-rbd-provisioner"
922         caps = {
923             "mon": "profile rbd, allow command 'osd blocklist'",
924             "mgr": "allow rw",
925             "osd": "profile rbd",
```

```
926             }
927             if self._arg_parser.restricted_auth_permission:
928                 rbd_pool_name = self._arg_parser.rbd_data_pool_name
929                 alias_rbd_pool_name =
                    ↪  self._arg_parser.alias_rbd_data_pool_name
930                 k8s_cluster_name = self._arg_parser.k8s_cluster_name
931                 if rbd_pool_name == "":
932                     raise ExecutionFailureException(
933                         "mandatory flag not found, please set the
                            ↪  '--rbd-data-pool-name' flag"
934                     )
935                 if k8s_cluster_name == "":
936                     raise ExecutionFailureException(
937                         "mandatory flag not found, please set the
                            ↪  '--k8s-cluster-name' flag"
938                     )
939                 entity = self.get_entity(
940                     entity, rbd_pool_name, alias_rbd_pool_name,
                        ↪  k8s_cluster_name
941                 )
942                 caps["osd"] = f"profile rbd pool={rbd_pool_name}"
943
944             return caps, entity
945
946         def get_rbd_node_caps_and_entity(self):
```

```
947         entity = "client.csi-rbd-node"
948         caps = {
949             "mon": "profile rbd, allow command 'osd blocklist'",
950             "osd": "profile rbd",
951         }
952         if self._arg_parser.restricted_auth_permission:
953             rbd_pool_name = self._arg_parser.rbd_data_pool_name
954             alias_rbd_pool_name =
                ↪  self._arg_parser.alias_rbd_data_pool_name
955             k8s_cluster_name = self._arg_parser.k8s_cluster_name
956             if rbd_pool_name == "":
957                 raise ExecutionFailureException(
958                     "mandatory flag not found, please set the
                        ↪  '--rbd-data-pool-name' flag"
959                 )
960             if k8s_cluster_name == "":
961                 raise ExecutionFailureException(
962                     "mandatory flag not found, please set the
                        ↪  '--k8s-cluster-name' flag"
963                 )
964             entity = self.get_entity(
965                 entity, rbd_pool_name, alias_rbd_pool_name,
                    ↪  k8s_cluster_name
966             )
967             caps["osd"] = f"profile rbd pool={rbd_pool_name}"
```

```
968
969             return caps, entity
970
971         def get_healthchecker_caps_and_entity(self):
972             entity = "client.healthchecker"
973             caps = {
974                 "mon": "allow r, allow command quorum_status, allow
                    ↪  command version",
975                 "mgr": "allow command config",
976                 "osd": f"profile rbd-read-only, allow rwx
                    ↪  pool={self._arg_parser.rgw_pool_prefix}.rgw.meta,
                    ↪  allow r pool=.rgw.root, allow rw
                    ↪  pool={self._arg_parser.rgw_pool_prefix}.rgw.control,
                    ↪  allow rx
                    ↪  pool={self._arg_parser.rgw_pool_prefix}.rgw.log,
                    ↪  allow x pool={self._arg_parser.rgw_pool_prefix}.rgw.b
                    ↪  uckets.index",
977             }
978
979             return caps, entity
980
981         def get_caps_and_entity(self, user_name):
982             if "client.csi-cephfs-provisioner" in user_name:
983                 if "client.csi-cephfs-provisioner" != user_name:
984                     self._arg_parser.restricted_auth_permission = True
```

```
985             return self.get_cephfs_provisioner_caps_and_entity()
986         if "client.csi-cephfs-node" in user_name:
987             if "client.csi-cephfs-node" != user_name:
988                 self._arg_parser.restricted_auth_permission = True
989             return self.get_cephfs_node_caps_and_entity()
990         if "client.csi-rbd-provisioner" in user_name:
991             if "client.csi-rbd-provisioner" != user_name:
992                 self._arg_parser.restricted_auth_permission = True
993             return self.get_rbd_provisioner_caps_and_entity()
994         if "client.csi-rbd-node" in user_name:
995             if "client.csi-rbd-node" != user_name:
996                 self._arg_parser.restricted_auth_permission = True
997             return self.get_rbd_node_caps_and_entity()
998         if "client.healthchecker" in user_name:
999             if "client.healthchecker" != user_name:
1000                self._arg_parser.restricted_auth_permission = True
1001            return self.get_healthchecker_caps_and_entity()
1002
1003        raise ExecutionFailureException(
1004            f"no user found with user_name: {user_name}, "
1005            "get_caps_and_entity command failed.\n"
1006        )
1007
1008    def create_cephCSIKeyring_user(self, user):
1009        """
```

```
1010            command: ceph auth get-or-create
                ↪  client.csi-cephfs-provisioner mon 'allow r' mgr 'allow
                ↪  rw' osd 'allow rw tag cephfs metadata=*'
1011            """
1012            caps, entity = self.get_caps_and_entity(user)
1013            cmd_json = {
1014                "prefix": "auth get-or-create",
1015                "entity": entity,
1016                "caps": [cap for cap_list in list(caps.items()) for cap
                    ↪  in cap_list],
1017                "format": "json",
1018            }
1019
1020            if self._arg_parser.dry_run:
1021                return (
1022                    self.dry_run(
1023                        "ceph "
1024                        + cmd_json["prefix"]
1025                        + " "
1026                        + cmd_json["entity"]
1027                        + " "
1028                        + " ".join(cmd_json["caps"])
1029                    ),
1030                    "",
1031                )
```

```
1032            # check if user already exist
1033            user_key = self.check_user_exist(entity)
1034            if user_key != "":
1035                return user_key, f"{entity.split('.', 1)[1]}"
1036                # entity.split('.',1)[1] to rename
                    ↪  entity(client.csi-rbd-node) as csi-rbd-node
1037
1038            ret_val, json_out, err_msg =
                ↪  self._common_cmd_json_gen(cmd_json)
1039            # if there is an unsuccessful attempt,
1040            if ret_val != 0 or len(json_out) == 0:
1041                raise ExecutionFailureException(
1042                    f"'auth get-or-create {user}' command failed.\n"
1043                    f"Error: {err_msg if ret_val != 0 else
                        ↪  self.EMPTY_OUTPUT_LIST}"
1044                )
1045            return str(json_out[0]["key"]), f"{entity.split('.', 1)[1]}"
1046            # entity.split('.',1)[1] to rename
                ↪  entity(client.csi-rbd-node) as csi-rbd-node
1047
1048        def get_cephfs_data_pool_details(self):
1049            cmd_json = {"prefix": "fs ls", "format": "json"}
1050            if self._arg_parser.dry_run:
1051                return self.dry_run("ceph " + cmd_json["prefix"])
```

```
1052            ret_val, json_out, err_msg =
                ↪  self._common_cmd_json_gen(cmd_json)
1053            # if there is an unsuccessful attempt, report an error
1054            if ret_val != 0:
1055                # if fs and data_pool arguments are not set, silently
                    ↪  return
1056                if (
1057                    self._arg_parser.cephfs_filesystem_name == ""
1058                    and self._arg_parser.cephfs_data_pool_name == ""
1059                ):
1060                    return
1061                # if user has provided any of the
1062                # '--cephfs-filesystem-name' or
                    ↪  '--cephfs-data-pool-name' arguments,
1063                # raise an exception as we are unable to verify the args
1064                raise ExecutionFailureException(
1065                    f"'fs ls' ceph call failed with error: {err_msg}"
1066                )
1067
1068            matching_json_out = {}
1069            # if '--cephfs-filesystem-name' argument is provided,
1070            # check whether the provided filesystem-name exists or not
1071            if self._arg_parser.cephfs_filesystem_name:
1072                # get the matching list
1073                matching_json_out_list = [
```

```
1074                matched
1075                for matched in json_out
1076                if str(matched["name"]) ==
                    ↪  self._arg_parser.cephfs_filesystem_name
1077            ]
1078            # unable to find a matching fs-name, raise an error
1079            if len(matching_json_out_list) == 0:
1080                raise ExecutionFailureException(
1081                    f"Filesystem provided,
                        ↪  '{self._arg_parser.cephfs_filesystem_name}', "
1082                    f"is not found in the fs-list: {[str(x['name'])
                        ↪  for x in json_out]}"
1083                )
1084            matching_json_out = matching_json_out_list[0]
1085        # if cephfs filesystem name is not provided,
1086        # try to get a default fs name by doing the following
1087        else:
1088            # a. check if there is only one filesystem is present
1089            if len(json_out) == 1:
1090                matching_json_out = json_out[0]
1091            # b. or else, check if data_pool name is provided
1092            elif self._arg_parser.cephfs_data_pool_name:
1093                # and if present, check whether there exists a fs
                    ↪  which has the data_pool
1094                for eachJ in json_out:
```

```
1095                    if self._arg_parser.cephfs_data_pool_name in
                        ↪  eachJ["data_pools"]:
1096                        matching_json_out = eachJ
1097                        break
1098                # if there is no matching fs exists, that means
                    ↪  provided data_pool name is invalid
1099                if not matching_json_out:
1100                    raise ExecutionFailureException(
1101                        f"Provided data_pool name,
                            ↪  {self._arg_parser.cephfs_data_pool_name},"
1102                        " does not exists"
1103                    )
1104            # c. if nothing is set and couldn't find a default,
1105            else:
1106                # just return silently
1107                return
1108
1109        if matching_json_out:
1110            self._arg_parser.cephfs_filesystem_name =
                ↪  str(matching_json_out["name"])
1111            self._arg_parser.cephfs_metadata_pool_name = str(
1112                matching_json_out["metadata_pool"]
1113            )
1114
1115        if isinstance(matching_json_out["data_pools"], list):
```

```
1116                # if the user has already provided data-pool-name,
1117                # through --cephfs-data-pool-name
1118                if self._arg_parser.cephfs_data_pool_name:
1119                    # if the provided name is not matching with the one
                        ↪  in the list
1120                    if (
1121                        self._arg_parser.cephfs_data_pool_name
1122                        not in matching_json_out["data_pools"]
1123                    ):
1124                        raise ExecutionFailureException(
1125                            f"Provided data-pool-name: '{self._arg_parser
                                ↪  .cephfs_data_pool_name}',
                                ↪  "
1126                            "doesn't match from the data-pools list: "
1127                            f"{[str(x) for x in
                                ↪  matching_json_out['data_pools']]}"
1128                        )
1129                # if data_pool name is not provided,
1130                # then try to find a default data pool name
1131                else:
1132                    # if no data_pools exist, silently return
1133                    if len(matching_json_out["data_pools"]) == 0:
1134                        return
1135                    self._arg_parser.cephfs_data_pool_name = str(
1136                        matching_json_out["data_pools"][0]
```

```
1137                 )
1138             # if there are more than one 'data_pools' exist,
1139             # then warn the user that we are using the selected name
1140             if len(matching_json_out["data_pools"]) > 1:
1141                 print(
1142                     "WARNING: Multiple data pools detected: "
1143                     f"{[str(x) for x in
                        ↪  matching_json_out['data_pools']]}\n"
1144                     f"Using the data-pool:
                        ↪  '{self._arg_parser.cephfs_data_pool_name}'\n"
1145                 )
1146
1147     def create_checkerKey(self, user):
1148         caps, entity = self.get_caps_and_entity(user)
1149         cmd_json = {
1150             "prefix": "auth get-or-create",
1151             "entity": entity,
1152             "caps": [cap for cap_list in list(caps.items()) for cap
                    ↪  in cap_list],
1153             "format": "json",
1154         }
1155
1156         if self._arg_parser.dry_run:
1157             return self.dry_run(
1158                 "ceph "
```

```
1159                    + cmd_json["prefix"]
1160                    + " "
1161                    + cmd_json["entity"]
1162                    + " "
1163                    + " ".join(cmd_json["caps"])
1164                )
1165            # check if user already exist
1166            user_key = self.check_user_exist(entity)
1167            if user_key != "":
1168                return user_key
1169
1170            ret_val, json_out, err_msg =
                ↪  self._common_cmd_json_gen(cmd_json)
1171            # if there is an unsuccessful attempt,
1172            if ret_val != 0 or len(json_out) == 0:
1173                raise ExecutionFailureException(
1174                    f"'auth get-or-create {self.run_as_user}' command
                        ↪  failed\n"
1175                    f"Error: {err_msg if ret_val != 0 else
                        ↪  self.EMPTY_OUTPUT_LIST}"
1176                )
1177            return str(json_out[0]["key"])
1178
1179        def get_ceph_dashboard_link(self):
1180            cmd_json = {"prefix": "mgr services", "format": "json"}
```

```
1181            if self._arg_parser.dry_run:
1182                return self.dry_run("ceph " + cmd_json["prefix"])
1183            ret_val, json_out, _ = self._common_cmd_json_gen(cmd_json)
1184            # if there is an unsuccessful attempt,
1185            if ret_val != 0 or len(json_out) == 0:
1186                return None
1187            if "dashboard" not in json_out:
1188                return None
1189            return json_out["dashboard"]
1190
1191        def create_rgw_admin_ops_user(self):
1192            cmd = [
1193                "radosgw-admin",
1194                "user",
1195                "create",
1196                "--uid",
1197                self.EXTERNAL_RGW_ADMIN_OPS_USER_NAME,
1198                "--display-name",
1199                "Rook RGW Admin Ops user",
1200                "--caps",
1201                "buckets=*;users=*;usage=read;metadata=read;zone=read",
1202                "--rgw-realm",
1203                self._arg_parser.rgw_realm_name,
1204                "--rgw-zonegroup",
1205                self._arg_parser.rgw_zonegroup_name,
```

```
1206                 "--rgw-zone",
1207                 self._arg_parser.rgw_zone_name,
1208             ]
1209             if self._arg_parser.dry_run:
1210                 return self.dry_run("ceph " + " ".join(cmd))
1211             try:
1212                 output = subprocess.check_output(cmd,
                     ↪  stderr=subprocess.PIPE)
1213             except subprocess.CalledProcessError as execErr:
1214                 # if the user already exists, we just query it
1215                 if execErr.returncode == errno.EEXIST:
1216                     cmd = [
1217                         "radosgw-admin",
1218                         "user",
1219                         "info",
1220                         "--uid",
1221                         self.EXTERNAL_RGW_ADMIN_OPS_USER_NAME,
1222                         "--rgw-realm",
1223                         self._arg_parser.rgw_realm_name,
1224                         "--rgw-zonegroup",
1225                         self._arg_parser.rgw_zonegroup_name,
1226                         "--rgw-zone",
1227                         self._arg_parser.rgw_zone_name,
1228                     ]
1229                     try:
```

```
1230                     output = subprocess.check_output(cmd,
                         ↪  stderr=subprocess.PIPE)
1231                 except subprocess.CalledProcessError as execErr:
1232                     err_msg = (
1233                         f"failed to execute command {cmd}. Output:
                             ↪  {execErr.output}. "
1234                         f"Code: {execErr.returncode}. Error:
                             ↪  {execErr.stderr}"
1235                     )
1236                     sys.stderr.write(err_msg)
1237                     return None, None, False, "-1"
1238             else:
1239                 err_msg = (
1240                     f"failed to execute command {cmd}. Output:
                         ↪  {execErr.output}. "
1241                     f"Code: {execErr.returncode}. Error:
                         ↪  {execErr.stderr}"
1242                 )
1243                 sys.stderr.write(err_msg)
1244                 return None, None, False, "-1"
1245
1246         # if it is python2, don't check for ceph version for adding
             ↪  `info=read` cap(rgw_validation)
1247         if sys.version_info.major < 3:
1248             jsonoutput = json.loads(output)
```

```
1249                return (
1250                    jsonoutput["keys"][0]["access_key"],
1251                    jsonoutput["keys"][0]["secret_key"],
1252                    False,
1253                    "",
1254                )
1255
1256            # separately add info=read caps for rgw-endpoint ip
                ↪ validation
1257            info_cap_supported = True
1258            cmd = [
1259                "radosgw-admin",
1260                "caps",
1261                "add",
1262                "--uid",
1263                self.EXTERNAL_RGW_ADMIN_OPS_USER_NAME,
1264                "--caps",
1265                "info=read",
1266                "--rgw-realm",
1267                self._arg_parser.rgw_realm_name,
1268                "--rgw-zonegroup",
1269                self._arg_parser.rgw_zonegroup_name,
1270                "--rgw-zone",
1271                self._arg_parser.rgw_zone_name,
1272            ]
```

```
1273            try:
1274                output = subprocess.check_output(cmd,
                    ↪  stderr=subprocess.PIPE)
1275            except subprocess.CalledProcessError as execErr:
1276                # if the ceph version not supported for adding
                    ↪  `info=read` cap(rgw_validation)
1277                if (
1278                    "could not add caps: unable to add caps: info=read\n"
1279                    in execErr.stderr.decode("utf-8")
1280                    and execErr.returncode == 244
1281                ):
1282                    info_cap_supported = False
1283                else:
1284                    err_msg = (
1285                        f"failed to execute command {cmd}. Output:
                            ↪  {execErr.output}. "
1286                        f"Code: {execErr.returncode}. Error:
                            ↪  {execErr.stderr}"
1287                    )
1288                    sys.stderr.write(err_msg)
1289                    return None, None, False, "-1"
1290
1291            jsonoutput = json.loads(output)
1292            return (
1293                jsonoutput["keys"][0]["access_key"],
```

```
1294                jsonoutput["keys"][0]["secret_key"],
1295                info_cap_supported,
1296                "",
1297            )
1298
1299        def validate_rbd_pool(self):
1300            if not self.cluster.pool_exists(self._arg_parser.rbd_data_poo
                ↪ l_name):
1301                raise ExecutionFailureException(
1302                    f"The provided pool,
                        ↪ '{self._arg_parser.rbd_data_pool_name}', does not
                        ↪ exist"
1303                )
1304
1305        def init_rbd_pool(self):
1306            if isinstance(self.cluster, DummyRados):
1307                return
1308            rbd_pool_name = self._arg_parser.rbd_data_pool_name
1309            ioctx = self.cluster.open_ioctx(rbd_pool_name)
1310            rbd_inst = rbd.RBD()
1311            rbd_inst.pool_init(ioctx, True)
1312
1313        def validate_rados_namespace(self):
1314            rbd_pool_name = self._arg_parser.rbd_data_pool_name
1315            rados_namespace = self._arg_parser.rados_namespace
```

```
1316            if rados_namespace == "":
1317                return
1318            rbd_inst = rbd.RBD()
1319            ioctx = self.cluster.open_ioctx(rbd_pool_name)
1320            if rbd_inst.namespace_exists(ioctx, rados_namespace) is False:
1321                raise ExecutionFailureException(
1322                    f"The provided rados Namespace, '{rados_namespace}', "
1323                    f"is not found in the pool '{rbd_pool_name}'"
1324                )
1325
1326        def get_or_create_subvolume_group(self, subvolume_group,
        ↪  cephfs_filesystem_name):
1327            cmd = [
1328                "ceph",
1329                "fs",
1330                "subvolumegroup",
1331                "getpath",
1332                cephfs_filesystem_name,
1333                subvolume_group,
1334            ]
1335            try:
1336                _ = subprocess.check_output(cmd, stderr=subprocess.PIPE)
1337            except subprocess.CalledProcessError:
1338                cmd = [
1339                    "ceph",
```

```
1340                 "fs",
1341                 "subvolumegroup",
1342                 "create",
1343                 cephfs_filesystem_name,
1344                 subvolume_group,
1345             ]
1346             try:
1347                 _ = subprocess.check_output(cmd,
                     ↪  stderr=subprocess.PIPE)
1348             except subprocess.CalledProcessError:
1349                 raise ExecutionFailureException(
1350                     f"subvolume group {subvolume_group} is not able
                         ↪  to get created"
1351                 )
1352
1353     def pin_subvolume(
1354         self, subvolume_group, cephfs_filesystem_name, pin_type,
             ↪  pin_setting
1355     ):
1356         cmd = [
1357             "ceph",
1358             "fs",
1359             "subvolumegroup",
1360             "pin",
1361             cephfs_filesystem_name,
```

```
            subvolume_group,
            pin_type,
            pin_setting,
        ]
        try:
            _ = subprocess.check_output(cmd, stderr=subprocess.PIPE)
        except subprocess.CalledProcessError:
            raise ExecutionFailureException(
                f"subvolume group {subvolume_group} is not able to
                ↪  get pinned"
            )

    def get_rgw_fsid(self, base_url, verify):
        access_key = self.out_map["RGW_ADMIN_OPS_USER_ACCESS_KEY"]
        secret_key = self.out_map["RGW_ADMIN_OPS_USER_SECRET_KEY"]
        rgw_endpoint = self._arg_parser.rgw_endpoint
        base_url = base_url + "://" + rgw_endpoint + "/admin/info?"
        params = {"format": "json"}
        request_url = base_url + urlencode(params)

        try:
            r = requests.get(
                request_url,
                auth=S3Auth(access_key, secret_key, rgw_endpoint),
                verify=verify,
```

```
1386                 )
1387             except requests.exceptions.Timeout:
1388                 sys.stderr.write(
1389                     f"invalid endpoint:, not able to call admin-ops
                         ↪  api{rgw_endpoint}"
1390                 )
1391                 return "", "-1"
1392             r1 = r.json()
1393             if r1 is None or r1.get("info") is None:
1394                 sys.stderr.write(
1395                     f"The provided rgw Endpoint,
                         ↪  '{self._arg_parser.rgw_endpoint}', is invalid."
1396                 )
1397                 return (
1398                     "",
1399                     "-1",
1400                 )
1401
1402             return r1["info"]["storage_backends"][0]["cluster_id"], ""
1403
1404         def validate_rgw_endpoint(self, info_cap_supported):
1405             # if the 'cluster' instance is a dummy one,
1406             # don't try to reach out to the endpoint
1407             if isinstance(self.cluster, DummyRados):
1408                 return
```

```
1409
1410            rgw_endpoint = self._arg_parser.rgw_endpoint
1411
1412            # validate rgw endpoint only if ip address is passed
1413            ip_type = self._invalid_endpoint(rgw_endpoint)
1414
1415            # check if the rgw endpoint is reachable
1416            cert = None
1417            if not self._arg_parser.rgw_skip_tls and
                ↪  self.validate_rgw_endpoint_tls_cert():
1418                cert = self._arg_parser.rgw_tls_cert_path
1419            base_url, verify, err = self.endpoint_dial(rgw_endpoint,
                ↪  ip_type, cert=cert)
1420            if err != "":
1421                return "-1"
1422
1423            # check if the rgw endpoint belongs to the same cluster
1424            # only check if `info` cap is supported
1425            if info_cap_supported:
1426                fsid = self.get_fsid()
1427                rgw_fsid, err = self.get_rgw_fsid(base_url, verify)
1428                if err == "-1":
1429                    return "-1"
1430                if fsid != rgw_fsid:
1431                    sys.stderr.write(
```

```
1432                    f"The provided rgw Endpoint,
                        ↪  '{self._arg_parser.rgw_endpoint}', is
                        ↪  invalid. We are validating by calling the
                        ↪  adminops api through rgw-endpoint and
                        ↪  validating the cluster_id '{rgw_fsid}' is
                        ↪  equal to the ceph cluster fsid '{fsid}'"
1433                )
1434                return "-1"
1435
1436            # check if the rgw endpoint pool exist
1437            # only validate if rgw_pool_prefix is passed else it will
                ↪  take default value and we don't create these default
                ↪  pools
1438            if self._arg_parser.rgw_pool_prefix != "default":
1439                rgw_pools_to_validate = [
1440                    f"{self._arg_parser.rgw_pool_prefix}.rgw.meta",
1441                    ".rgw.root",
1442                    f"{self._arg_parser.rgw_pool_prefix}.rgw.control",
1443                    f"{self._arg_parser.rgw_pool_prefix}.rgw.log",
1444                ]
1445                for _rgw_pool_to_validate in rgw_pools_to_validate:
1446                    if not
                        ↪  self.cluster.pool_exists(_rgw_pool_to_validate):
1447                        sys.stderr.write(
```

```
1448                         f"The provided pool,
                             ↪  '{_rgw_pool_to_validate}', does not exist"
1449                     )
1450                     return "-1"
1451
1452         return ""
1453
1454     def validate_rgw_multisite(self, rgw_multisite_config_name,
     ↪  rgw_multisite_config):
1455         if rgw_multisite_config != "":
1456             cmd = [
1457                 "radosgw-admin",
1458                 rgw_multisite_config,
1459                 "get",
1460                 "--rgw-" + rgw_multisite_config,
1461                 rgw_multisite_config_name,
1462             ]
1463             try:
1464                 _ = subprocess.check_output(cmd,
                     ↪  stderr=subprocess.PIPE)
1465             except subprocess.CalledProcessError as execErr:
1466                 err_msg = (
1467                     f"failed to execute command {cmd}. Output:
                         ↪  {execErr.output}. "
```

```
1468                        f"Code: {execErr.returncode}. Error:
                            ↪  {execErr.stderr}"
1469                    )
1470                    sys.stderr.write(err_msg)
1471                    return "-1"
1472            return ""
1473
1474        def _gen_output_map(self):
1475            if self.out_map:
1476                return
1477            self._arg_parser.k8s_cluster_name = (
1478                self._arg_parser.k8s_cluster_name.lower()
1479            )  # always convert cluster name to lowercase characters
1480            self.validate_rbd_pool()
1481            self.init_rbd_pool()
1482            self.validate_rados_namespace()
1483            self._excluded_keys.add("K8S_CLUSTER_NAME")
1484            self.get_cephfs_data_pool_details()
1485            self.out_map["NAMESPACE"] = self._arg_parser.namespace
1486            self.out_map["K8S_CLUSTER_NAME"] =
            ↪  self._arg_parser.k8s_cluster_name
1487            self.out_map["ROOK_EXTERNAL_FSID"] = self.get_fsid()
1488            self.out_map["ROOK_EXTERNAL_USERNAME"] = self.run_as_user
1489            self.out_map["ROOK_EXTERNAL_CEPH_MON_DATA"] =
            ↪  self.get_ceph_external_mon_data()
```

```
1490            self.out_map["ROOK_EXTERNAL_USER_SECRET"] =
                ↪  self.create_checkerKey(
1491                "client.healthchecker"
1492            )
1493            self.out_map["ROOK_EXTERNAL_DASHBOARD_LINK"] =
                ↪  self.get_ceph_dashboard_link()
1494            (
1495                self.out_map["CSI_RBD_NODE_SECRET"],
1496                self.out_map["CSI_RBD_NODE_SECRET_NAME"],
1497            ) = self.create_cephCSIKeyring_user("client.csi-rbd-node")
1498            (
1499                self.out_map["CSI_RBD_PROVISIONER_SECRET"],
1500                self.out_map["CSI_RBD_PROVISIONER_SECRET_NAME"],
1501            ) = self.create_cephCSIKeyring_user("client.csi-rbd-provision
                ↪  er")
1502            self.out_map["CEPHFS_POOL_NAME"] =
                ↪  self._arg_parser.cephfs_data_pool_name
1503            self.out_map[
1504                "CEPHFS_METADATA_POOL_NAME"
1505            ] = self._arg_parser.cephfs_metadata_pool_name
1506            self.out_map["CEPHFS_FS_NAME"] =
                ↪  self._arg_parser.cephfs_filesystem_name
1507            self.out_map[
1508                "RESTRICTED_AUTH_PERMISSION"
1509            ] = self._arg_parser.restricted_auth_permission
```

```
1510            self.out_map["RADOS_NAMESPACE"] =
                ↪  self._arg_parser.rados_namespace
1511            self.out_map["SUBVOLUME_GROUP"] =
                ↪  self._arg_parser.subvolume_group
1512            self.out_map["CSI_CEPHFS_NODE_SECRET"] = ""
1513            self.out_map["CSI_CEPHFS_PROVISIONER_SECRET"] = ""
1514            # create CephFS node and provisioner keyring only when MDS
                ↪  exists
1515            if self.out_map["CEPHFS_FS_NAME"] and
                ↪  self.out_map["CEPHFS_POOL_NAME"]:
1516                (
1517                    self.out_map["CSI_CEPHFS_NODE_SECRET"],
1518                    self.out_map["CSI_CEPHFS_NODE_SECRET_NAME"],
1519                ) = self.create_cephCSIKeyring_user("client.csi-cephfs-no
                    ↪  de")
1520                (
1521                    self.out_map["CSI_CEPHFS_PROVISIONER_SECRET"],
1522                    self.out_map["CSI_CEPHFS_PROVISIONER_SECRET_NAME"],
1523                ) = self.create_cephCSIKeyring_user("client.csi-cephfs-pr
                    ↪  ovisioner")
1524                # create the default "csi" subvolumegroup
1525                self.get_or_create_subvolume_group(
1526                    "csi", self._arg_parser.cephfs_filesystem_name
1527                )
1528                # pin the default "csi" subvolumegroup
```

```
1529                self.pin_subvolume(
1530                    "csi", self._arg_parser.cephfs_filesystem_name,
                        ↪  "distributed", "1"
1531                )
1532                if self.out_map["SUBVOLUME_GROUP"]:
1533                    self.get_or_create_subvolume_group(
1534                        self._arg_parser.subvolume_group,
1535                        self._arg_parser.cephfs_filesystem_name,
1536                    )
1537                    self.pin_subvolume(
1538                        self._arg_parser.subvolume_group,
1539                        self._arg_parser.cephfs_filesystem_name,
1540                        "distributed",
1541                        "1",
1542                    )
1543            self.out_map["RGW_TLS_CERT"] = ""
1544            self.out_map["MONITORING_ENDPOINT"] = ""
1545            self.out_map["MONITORING_ENDPOINT_PORT"] = ""
1546            if not self._arg_parser.skip_monitoring_endpoint:
1547                (
1548                    self.out_map["MONITORING_ENDPOINT"],
1549                    self.out_map["MONITORING_ENDPOINT_PORT"],
1550                ) = self.get_active_and_standby_mgrs()
1551            self.out_map["RBD_POOL_NAME"] =
            ↪  self._arg_parser.rbd_data_pool_name
```

```
1552            self.out_map[
1553                "RBD_METADATA_EC_POOL_NAME"
1554            ] = self.validate_rbd_metadata_ec_pool_name()
1555            self.out_map["RGW_POOL_PREFIX"] =
                ↪  self._arg_parser.rgw_pool_prefix
1556            self.out_map["RGW_ENDPOINT"] = ""
1557            if self._arg_parser.rgw_endpoint:
1558                if self._arg_parser.dry_run:
1559                    self.create_rgw_admin_ops_user()
1560                else:
1561                    if (
1562                        self._arg_parser.rgw_realm_name != ""
1563                        and self._arg_parser.rgw_zonegroup_name != ""
1564                        and self._arg_parser.rgw_zone_name != ""
1565                    ):
1566                        err = self.validate_rgw_multisite(
1567                            self._arg_parser.rgw_realm_name, "realm"
1568                        )
1569                        err = self.validate_rgw_multisite(
1570                            self._arg_parser.rgw_zonegroup_name,
                             ↪  "zonegroup"
1571                        )
1572                        err = self.validate_rgw_multisite(
1573                            self._arg_parser.rgw_zone_name, "zone"
1574                        )
```

```
1575
1576                    if (
1577                        self._arg_parser.rgw_realm_name == ""
1578                        and self._arg_parser.rgw_zonegroup_name == ""
1579                        and self._arg_parser.rgw_zone_name == ""
1580                    ) or (
1581                        self._arg_parser.rgw_realm_name != ""
1582                        and self._arg_parser.rgw_zonegroup_name != ""
1583                        and self._arg_parser.rgw_zone_name != ""
1584                    ):
1585                        (
1586                            self.out_map["RGW_ADMIN_OPS_USER_ACCESS_KEY"],
1587                            self.out_map["RGW_ADMIN_OPS_USER_SECRET_KEY"],
1588                            info_cap_supported,
1589                            err,
1590                        ) = self.create_rgw_admin_ops_user()
1591                        err =
                        ↪  self.validate_rgw_endpoint(info_cap_supported)
1592                        if self._arg_parser.rgw_tls_cert_path:
1593                            self.out_map[
1594                                "RGW_TLS_CERT"
1595                            ] = self.validate_rgw_endpoint_tls_cert()
1596                        # if there is no error, set the RGW_ENDPOINT
1597                        if err != "-1":
```

```
1598                         self.out_map["RGW_ENDPOINT"] =
                             ↪  self._arg_parser.rgw_endpoint
1599                 else:
1600                     err = "Please provide all the RGW multisite
                         ↪  parameters or none of them"
1601                     sys.stderr.write(err)
1602
1603     def gen_shell_out(self):
1604         self._gen_output_map()
1605         shOutIO = StringIO()
1606         for k, v in self.out_map.items():
1607             if v and k not in self._excluded_keys:
1608                 shOutIO.write(f"export {k}={v}{LINESEP}")
1609         shOut = shOutIO.getvalue()
1610         shOutIO.close()
1611         return shOut
1612
1613     def gen_json_out(self):
1614         self._gen_output_map()
1615         if self._arg_parser.dry_run:
1616             return ""
1617         json_out = [
1618             {
1619                 "name": "rook-ceph-mon-endpoints",
1620                 "kind": "ConfigMap",
```

```
1621                 "data": {
1622                     "data":
                         ↪  self.out_map["ROOK_EXTERNAL_CEPH_MON_DATA"],
1623                     "maxMonId": "0",
1624                     "mapping": "{}",
1625                 },
1626             },
1627             {
1628                 "name": "rook-ceph-mon",
1629                 "kind": "Secret",
1630                 "data": {
1631                     "admin-secret": "admin-secret",
1632                     "fsid": self.out_map["ROOK_EXTERNAL_FSID"],
1633                     "mon-secret": "mon-secret",
1634                 },
1635             },
1636             {
1637                 "name": "rook-ceph-operator-creds",
1638                 "kind": "Secret",
1639                 "data": {
1640                     "userID": self.out_map["ROOK_EXTERNAL_USERNAME"],
1641                     "userKey":
                         ↪  self.out_map["ROOK_EXTERNAL_USER_SECRET"],
1642                 },
1643             },
```

```
        ]

        # if 'MONITORING_ENDPOINT' exists, then only add 'monitoring-endpoint' to Cluster
        if (
            self.out_map["MONITORING_ENDPOINT"]
            and self.out_map["MONITORING_ENDPOINT_PORT"]
        ):
            json_out.append(
                {
                    "name": "monitoring-endpoint",
                    "kind": "CephCluster",
                    "data": {
                        "MonitoringEndpoint": self.out_map["MONITORING_ENDPOINT"],
                        "MonitoringPort": self.out_map["MONITORING_ENDPOINT_PORT"],
                    },
                }
            )

        # if 'CSI_RBD_NODE_SECRET' exists, then only add 'rook-csi-rbd-provisioner' Secret
        if (
            self.out_map["CSI_RBD_NODE_SECRET"]
```

```
1665                 and self.out_map["CSI_RBD_NODE_SECRET_NAME"]
1666             ):
1667                 json_out.append(
1668                     {
1669                         "name": f"rook-{self.out_map['CSI_RBD_NODE_SECRET
                             ↪  _NAME']}",
1670                         "kind": "Secret",
1671                         "data": {
1672                             "userID":
                                 ↪  self.out_map["CSI_RBD_NODE_SECRET_NAME"],
1673                             "userKey":
                                 ↪  self.out_map["CSI_RBD_NODE_SECRET"],
1674                         },
1675                     }
1676                 )
1677             # if 'CSI_RBD_PROVISIONER_SECRET' exists, then only add
                 ↪  'rook-csi-rbd-provisioner' Secret
1678             if (
1679                 self.out_map["CSI_RBD_PROVISIONER_SECRET"]
1680                 and self.out_map["CSI_RBD_PROVISIONER_SECRET_NAME"]
1681             ):
1682                 json_out.append(
1683                     {
1684                         "name": f"rook-{self.out_map['CSI_RBD_PROVISIONER
                             ↪  _SECRET_NAME']}",
```

```
                    "kind": "Secret",
                    "data": {
                        "userID": self.out_map["CSI_RBD_PROVISIONER_S
                        ↪  ECRET_NAME"],
                        "userKey": self.out_map["CSI_RBD_PROVISIONER_
                        ↪  SECRET"],
                    },
                }
            )
        # if 'CSI_CEPHFS_PROVISIONER_SECRET' exists, then only add
        ↪  'rook-csi-cephfs-provisioner' Secret
        if (
            self.out_map["CSI_CEPHFS_PROVISIONER_SECRET"]
            and self.out_map["CSI_CEPHFS_PROVISIONER_SECRET_NAME"]
        ):
            json_out.append(
                {
                    "name": f"rook-{self.out_map['CSI_CEPHFS_PROVISIO
                    ↪  NER_SECRET_NAME']}",
                    "kind": "Secret",
                    "data": {
                        "adminID": self.out_map["CSI_CEPHFS_PROVISION
                        ↪  ER_SECRET_NAME"],
                        "adminKey": self.out_map["CSI_CEPHFS_PROVISIO
                        ↪  NER_SECRET"],
```

```
                    },
                }
            )
        # if 'CSI_CEPHFS_NODE_SECRET' exists, then only add
        ↪  'rook-csi-cephfs-node' Secret
        if (
            self.out_map["CSI_CEPHFS_NODE_SECRET"]
            and self.out_map["CSI_CEPHFS_NODE_SECRET_NAME"]
        ):
            json_out.append(
                {
                    "name": f"rook-{self.out_map['CSI_CEPHFS_NODE_SEC
                    ↪  RET_NAME']}",
                    "kind": "Secret",
                    "data": {
                        "adminID": self.out_map["CSI_CEPHFS_NODE_SECR
                        ↪  ET_NAME"],
                        "adminKey":
                        ↪  self.out_map["CSI_CEPHFS_NODE_SECRET"],
                    },
                }
            )
        # if 'ROOK_EXTERNAL_DASHBOARD_LINK' exists, then only add
        ↪  'rook-ceph-dashboard-link' Secret
        if self.out_map["ROOK_EXTERNAL_DASHBOARD_LINK"]:
```

```
1724                json_out.append(
1725                    {
1726                        "name": "rook-ceph-dashboard-link",
1727                        "kind": "Secret",
1728                        "data": {
1729                            "userID": "ceph-dashboard-link",
1730                            "userKey": self.out_map["ROOK_EXTERNAL_DASHBO
                                ↪  ARD_LINK"],
1731                        },
1732                    }
1733                )
1734            if self.out_map["RBD_METADATA_EC_POOL_NAME"]:
1735                json_out.append(
1736                    {
1737                        "name": "ceph-rbd",
1738                        "kind": "StorageClass",
1739                        "data": {
1740                            "dataPool": self.out_map["RBD_POOL_NAME"],
1741                            "pool":
                                ↪  self.out_map["RBD_METADATA_EC_POOL_NAME"],
1742                            "csi.storage.k8s.io/provisioner-secret-name":
                                ↪  f"rook-{self.out_map['CSI_RBD_PROVISIONER
                                ↪  _SECRET_NAME']}",
```

```
1743                         "csi.storage.k8s.io/controller-expand-secret-
                             ↪ name":
                             ↪ f"rook-{self.out_map['CSI_RBD_PROVISIONER
                             ↪ _SECRET_NAME']}",
1744                         "csi.storage.k8s.io/node-stage-secret-name":
                             ↪ f"rook-{self.out_map['CSI_RBD_NODE_SECRET
                             ↪ _NAME']}",
1745                     },
1746                 }
1747             )
1748         else:
1749             json_out.append(
1750                 {
1751                     "name": "ceph-rbd",
1752                     "kind": "StorageClass",
1753                     "data": {
1754                         "pool": self.out_map["RBD_POOL_NAME"],
1755                         "csi.storage.k8s.io/provisioner-secret-name":
                             ↪ f"rook-{self.out_map['CSI_RBD_PROVISIONER
                             ↪ _SECRET_NAME']}",
1756                         "csi.storage.k8s.io/controller-expand-secret-
                             ↪ name":
                             ↪ f"rook-{self.out_map['CSI_RBD_PROVISIONER
                             ↪ _SECRET_NAME']}",
```

```
1757                         "csi.storage.k8s.io/node-stage-secret-name":
                             ↪ f"rook-{self.out_map['CSI_RBD_NODE_SECRET
                             ↪ _NAME']}",
1758                     },
1759                 }
1760             )
1761         # if 'CEPHFS_FS_NAME' exists, then only add 'cephfs'
             ↪ StorageClass
1762         if self.out_map["CEPHFS_FS_NAME"]:
1763             json_out.append(
1764                 {
1765                     "name": "cephfs",
1766                     "kind": "StorageClass",
1767                     "data": {
1768                         "fsName": self.out_map["CEPHFS_FS_NAME"],
1769                         "pool": self.out_map["CEPHFS_POOL_NAME"],
1770                         "csi.storage.k8s.io/provisioner-secret-name":
                             ↪ f"rook-{self.out_map['CSI_CEPHFS_PROVISIO
                             ↪ NER_SECRET_NAME']}",
1771                         "csi.storage.k8s.io/controller-expand-secret-
                             ↪ name":
                             ↪ f"rook-{self.out_map['CSI_CEPHFS_PROVISIO
                             ↪ NER_SECRET_NAME']}",
```

```
                        "csi.storage.k8s.io/node-stage-secret-name":
                        ↪ f"rook-{self.out_map['CSI_CEPHFS_NODE_SEC
                        ↪ RET_NAME']}",
                    },
                }
            )
        # if 'RGW_ENDPOINT' exists, then only add 'ceph-rgw'
        ↪ StorageClass
        if self.out_map["RGW_ENDPOINT"]:
            json_out.append(
                {
                    "name": "ceph-rgw",
                    "kind": "StorageClass",
                    "data": {
                        "endpoint": self.out_map["RGW_ENDPOINT"],
                        "poolPrefix": self.out_map["RGW_POOL_PREFIX"],
                    },
                }
            )
            json_out.append(
                {
                    "name": "rgw-admin-ops-user",
                    "kind": "Secret",
                    "data": {
```

```
1793                         "accessKey": self.out_map["RGW_ADMIN_OPS_USER
                             ↪  _ACCESS_KEY"],
1794                         "secretKey": self.out_map["RGW_ADMIN_OPS_USER
                             ↪  _SECRET_KEY"],
1795                     },
1796                 }
1797             )
1798         # if 'RGW_TLS_CERT' exists, then only add the
             ↪  "ceph-rgw-tls-cert" secret
1799         if self.out_map["RGW_TLS_CERT"]:
1800             json_out.append(
1801                 {
1802                     "name": "ceph-rgw-tls-cert",
1803                     "kind": "Secret",
1804                     "data": {
1805                         "cert": self.out_map["RGW_TLS_CERT"],
1806                     },
1807                 }
1808             )
1809 
1810         return json.dumps(json_out) + LINESEP
1811 
1812     def upgrade_users_permissions(self):
1813         users = [
1814             "client.csi-cephfs-node",
```

```
1815                "client.csi-cephfs-provisioner",
1816                "client.csi-rbd-node",
1817                "client.csi-rbd-provisioner",
1818                "client.healthchecker",
1819            ]
1820            if self.run_as_user != "" and self.run_as_user not in users:
1821                users.append(self.run_as_user)
1822            for user in users:
1823                self.upgrade_user_permissions(user)
1824
1825        def get_rgw_pool_name_during_upgrade(self, user, caps):
1826            if user == "client.healthchecker":
1827                # when admin has not provided rgw pool name during
                    ↪  upgrade,
1828                # get the rgw pool name from client.healthchecker user
                    ↪  which was used during connection
1829                if not self._arg_parser.rgw_pool_prefix:
1830                    # To get value 'default' which is rgw pool name from
                        ↪  'allow rwx pool=default.rgw.meta'
1831                    pattern = r"pool=(.*?)\.rgw\.meta"
1832                    match = re.search(pattern, caps)
1833                    if match:
1834                        self._arg_parser.rgw_pool_prefix = match.group(1)
1835                    else:
1836                        raise ExecutionFailureException(
```

```
                        "failed to get rgw pool name for upgrade"
                    )

    def upgrade_user_permissions(self, user):
        # check whether the given user exists or not
        cmd_json = {"prefix": "auth get", "entity": f"{user}",
        ↪  "format": "json"}
        ret_val, json_out, err_msg =
        ↪  self._common_cmd_json_gen(cmd_json)
        if ret_val != 0 or len(json_out) == 0:
            print(f"user {user} not found for upgrading.")
            return
        existing_caps = json_out[0]["caps"]
        self.get_rgw_pool_name_during_upgrade(user,
        ↪  str(existing_caps))
        new_cap, _ = self.get_caps_and_entity(user)
        cap_keys = ["mon", "mgr", "osd", "mds"]
        caps = []
        for eachCap in cap_keys:
            cur_cap_values = existing_caps.get(eachCap, "")
            new_cap_values = new_cap.get(eachCap, "")
            cur_cap_perm_list = [
                x.strip() for x in cur_cap_values.split(",") if
                ↪  x.strip()
            ]
```

```
1858                new_cap_perm_list = [
1859                    x.strip() for x in new_cap_values.split(",") if
                        ↪  x.strip()
1860                ]
1861                # append new_cap_list to cur_cap_list to maintain the
                    ↪  order of caps
1862                cur_cap_perm_list.extend(new_cap_perm_list)
1863                # eliminate duplicates without using 'set'
1864                # set re-orders items in the list and we have to keep
                    ↪  the order
1865                new_cap_list = []
1866                [new_cap_list.append(x) for x in cur_cap_perm_list if x
                    ↪  not in new_cap_list]
1867                existing_caps[eachCap] = ", ".join(new_cap_list)
1868                if existing_caps[eachCap]:
1869                    caps.append(eachCap)
1870                    caps.append(existing_caps[eachCap])
1871            cmd_json = {
1872                "prefix": "auth caps",
1873                "entity": user,
1874                "caps": caps,
1875                "format": "json",
1876            }
1877            ret_val, json_out, err_msg =
                ↪  self._common_cmd_json_gen(cmd_json)
```

```
1878            if ret_val != 0:
1879                raise ExecutionFailureException(
1880                    f"'auth caps {user}' command failed.\n Error:
                        ↪  {err_msg}"
1881                )
1882            print(f"Updated user {user} successfully.")
1883
1884        def main(self):
1885            generated_output = ""
1886            if self._arg_parser.upgrade:
1887                self.upgrade_users_permissions()
1888            elif self._arg_parser.format == "json":
1889                generated_output = self.gen_json_out()
1890            elif self._arg_parser.format == "bash":
1891                generated_output = self.gen_shell_out()
1892            else:
1893                raise ExecutionFailureException(
1894                    f"Unsupported format: {self._arg_parser.format}"
1895                )
1896            print(generated_output)
1897            if self.output_file and generated_output:
1898                fOut = open(self.output_file, mode="w", encoding="UTF-8")
1899                fOut.write(generated_output)
1900                fOut.close()
1901
```

```
1902
1903 ################################################
1904 ##################### MAIN #####################
1905 ################################################
1906 if __name__ == "__main__":
1907     rjObj = RadosJSON()
1908     try:
1909         rjObj.main()
1910     except ExecutionFailureException as err:
1911         print(f"Execution Failed: {err}")
1912         raise err
1913     except KeyError as kErr:
1914         print(f"KeyError: {kErr}")
1915     except OSError as osErr:
1916         print(f"Error while trying to output the data: {osErr}")
1917     finally:
1918         rjObj.shutdown()
```

Code 3: create-external-cluster-resources.py

```
1 #!/bin/bash
2 set -e
3
4 ##############
5 # VARIABLES #
6 ##############
```

```
7  NAMESPACE=${NAMESPACE:="rook-ceph-external"}
8  MON_SECRET_NAME=rook-ceph-mon
9  RGW_ADMIN_OPS_USER_SECRET_NAME=rgw-admin-ops-user
10 MON_SECRET_CLUSTER_NAME_KEYNAME=cluster-name
11 MON_SECRET_FSID_KEYNAME=fsid
12 MON_SECRET_ADMIN_KEYRING_KEYNAME=admin-secret
13 MON_SECRET_MON_KEYRING_KEYNAME=mon-secret
14 MON_SECRET_CEPH_USERNAME_KEYNAME=ceph-username
15 MON_SECRET_CEPH_SECRET_KEYNAME=ceph-secret
16 MON_ENDPOINT_CONFIGMAP_NAME=rook-ceph-mon-endpoints
17 ROOK_EXTERNAL_CLUSTER_NAME=$NAMESPACE
18 ROOK_RBD_FEATURES=${ROOK_RBD_FEATURES:-"layering"}
19 ROOK_EXTERNAL_MAX_MON_ID=2
20 ROOK_EXTERNAL_MAPPING={}
21 RBD_STORAGE_CLASS_NAME=ceph-rbd
22 CEPHFS_STORAGE_CLASS_NAME=cephfs
23 ROOK_EXTERNAL_MONITOR_SECRET=mon-secret
24 OPERATOR_NAMESPACE=rook-ceph                                    #
   ↪ default set to rook-ceph
25 RBD_PROVISIONER=$OPERATOR_NAMESPACE".rbd.csi.ceph.com"         #
   ↪ driver:namespace:operator
26 CEPHFS_PROVISIONER=$OPERATOR_NAMESPACE".cephfs.csi.ceph.com" #
   ↪ driver:namespace:operator
27 CLUSTER_ID_RBD=$NAMESPACE
28 CLUSTER_ID_CEPHFS=$NAMESPACE
```

```
29 : "${ROOK_EXTERNAL_ADMIN_SECRET:=admin-secret}"
30
31 #############
32 # FUNCTIONS #
33 #############
34
35 function checkEnvVars() {
36   if [ -z "$NAMESPACE" ]; then
37     echo "Please populate the environment variable NAMESPACE"
38     exit 1
39   fi
40   if [ -z "$ROOK_RBD_FEATURES" ]  [[ ! "$ROOK_RBD_FEATURES" =~
     ↪  .*"layering".* ]]; then
41     echo "Please populate the environment variable ROOK_RBD_FEATURES"
42     echo "For a kernel earlier than 5.4 use a value of 'layering';
       ↪  for 5.4 or later"
43     echo "use
       ↪  'layering,fast-diff,object-map,deep-flatten,exclusive-lock'"
44     exit 1
45   fi
46   if [ -z "$ROOK_EXTERNAL_FSID" ]; then
47     echo "Please populate the environment variable ROOK_EXTERNAL_FSID"
48     exit 1
49   fi
50   if [ -z "$ROOK_EXTERNAL_CEPH_MON_DATA" ]; then
```

```
51      echo "Please populate the environment variable
        ↪  ROOK_EXTERNAL_CEPH_MON_DATA"
52      exit 1
53    fi
54    if [[ "$ROOK_EXTERNAL_ADMIN_SECRET" == "admin-secret" ]]; then
55      if [ -z "$ROOK_EXTERNAL_USER_SECRET" ]; then
56        echo "Please populate the environment variable
          ↪  ROOK_EXTERNAL_USER_SECRET"
57        exit 1
58      fi
59      if [ -z "$ROOK_EXTERNAL_USERNAME" ]; then
60        echo "Please populate the environment variable
          ↪  ROOK_EXTERNAL_USERNAME"
61        exit 1
62      fi
63    fi
64    if [[ "$ROOK_EXTERNAL_ADMIN_SECRET" != "admin-secret" ]] && [ -n
      ↪  "$ROOK_EXTERNAL_USER_SECRET" ]; then
65      echo "Providing both ROOK_EXTERNAL_ADMIN_SECRET and
        ↪  ROOK_EXTERNAL_USER_SECRET is not supported, choose one only."
66      exit 1
67    fi
68  }
69
70  function createClusterNamespace() {
```

```
71    if ! kubectl get namespace "$NAMESPACE" &>/dev/null; then
72      kubectl \
73        create \
74        namespace \
75        "$NAMESPACE"
76    else
77      echo "cluster namespace $NAMESPACE already exists"
78    fi
79  }
80
81  function importClusterID() {
82    if [ -n "$RADOS_NAMESPACE" ]; then
83      CLUSTER_ID_RBD=$(kubectl -n "$NAMESPACE" get
        ↪  cephblockpoolradosnamespace.ceph.rook.io/"$RADOS_NAMESPACE"
        ↪  -o jsonpath='{.status.info.clusterID}')
84    fi
85    if [ -n "$SUBVOLUME_GROUP" ]; then
86      CLUSTER_ID_CEPHFS=$(kubectl -n "$NAMESPACE" get
        ↪  cephfilesystemsubvolumegroup.ceph.rook.io/"$SUBVOLUME_GROUP"
        ↪  -o jsonpath='{.status.info.clusterID}')
87    fi
88  }
89
90  function importSecret() {
```

```
91  if ! kubectl -n "$NAMESPACE" get secret "$MON_SECRET_NAME"
    ↪  &>/dev/null; then
92    kubectl -n "$NAMESPACE" \
93      create \
94      secret \
95      generic \
96      --type="kubernetes.io/rook" \
97      "$MON_SECRET_NAME" \
98      --from-literal="$MON_SECRET_CLUSTER_NAME_KEYNAME"="$ROOK_EXTERN
        ↪  AL_CLUSTER_NAME"
        ↪  \
99      --from-literal="$MON_SECRET_FSID_KEYNAME"="$ROOK_EXTERNAL_FSID"
        ↪  \
100     --from-literal="$MON_SECRET_ADMIN_KEYRING_KEYNAME"="$ROOK_EXTER
        ↪  NAL_ADMIN_SECRET"
        ↪  \
101     --from-literal="$MON_SECRET_MON_KEYRING_KEYNAME"="$ROOK_EXTERNA
        ↪  L_MONITOR_SECRET"
        ↪  \
102     --from-literal="$MON_SECRET_CEPH_USERNAME_KEYNAME"="$ROOK_EXTER
        ↪  NAL_USERNAME"
        ↪  \
103     --from-literal="$MON_SECRET_CEPH_SECRET_KEYNAME"="$ROOK_EXTERNA
        ↪  L_USER_SECRET"
104 else
```

```
105       echo "secret $MON_SECRET_NAME already exists"
106     fi
107   }
108
109   function importConfigMap() {
110     if ! kubectl -n "$NAMESPACE" get configmap
        ↪  "$MON_ENDPOINT_CONFIGMAP_NAME" &>/dev/null; then
111       kubectl -n "$NAMESPACE" \
112         create \
113         configmap \
114         "$MON_ENDPOINT_CONFIGMAP_NAME" \
115         --from-literal=data="$ROOK_EXTERNAL_CEPH_MON_DATA" \
116         --from-literal=mapping="$ROOK_EXTERNAL_MAPPING" \
117         --from-literal=maxMonId="$ROOK_EXTERNAL_MAX_MON_ID"
118     else
119       echo "configmap $MON_ENDPOINT_CONFIGMAP_NAME already exists"
120     fi
121   }
122
123   function importCsiRBDNodeSecret() {
124     if ! kubectl -n "$NAMESPACE" get secret
        ↪  "rook-$CSI_RBD_NODE_SECRET_NAME" &>/dev/null; then
125       kubectl -n "$NAMESPACE" \
126         create \
127         secret \
```

```
128         generic \
129         --type="kubernetes.io/rook" \
130         "rook-""$CSI_RBD_NODE_SECRET_NAME" \
131         --from-literal=userID="$CSI_RBD_NODE_SECRET_NAME" \
132         --from-literal=userKey="$CSI_RBD_NODE_SECRET"
133     else
134       echo "secret rook-$CSI_RBD_NODE_SECRET_NAME already exists"
135     fi
136 }
137
138 function importCsiRBDProvisionerSecret() {
139     if ! kubectl -n "$NAMESPACE" get secret
      ↪  "rook-$CSI_RBD_PROVISIONER_SECRET_NAME" &>/dev/null; then
140       kubectl -n "$NAMESPACE" \
141         create \
142         secret \
143         generic \
144         --type="kubernetes.io/rook" \
145         "rook-""$CSI_RBD_PROVISIONER_SECRET_NAME" \
146         --from-literal=userID="$CSI_RBD_PROVISIONER_SECRET_NAME" \
147         --from-literal=userKey="$CSI_RBD_PROVISIONER_SECRET"
148     else
149       echo "secret $CSI_RBD_PROVISIONER_SECRET_NAME already exists"
150     fi
151 }
```

```
152
153 function importCsiCephFSNodeSecret() {
154   if ! kubectl -n "$NAMESPACE" get secret
      ↪  "rook-$CSI_CEPHFS_NODE_SECRET_NAME" &>/dev/null; then
155     kubectl -n "$NAMESPACE" \
156       create \
157       secret \
158       generic \
159       --type="kubernetes.io/rook" \
160       "rook-""$CSI_CEPHFS_NODE_SECRET_NAME" \
161       --from-literal=adminID="$CSI_CEPHFS_NODE_SECRET_NAME" \
162       --from-literal=adminKey="$CSI_CEPHFS_NODE_SECRET"
163   else
164     echo "secret $CSI_CEPHFS_NODE_SECRET_NAME already exists"
165   fi
166 }
167
168 function importCsiCephFSProvisionerSecret() {
169   if ! kubectl -n "$NAMESPACE" get secret
      ↪  "rook-$CSI_CEPHFS_PROVISIONER_SECRET_NAME" &>/dev/null; then
170     kubectl -n "$NAMESPACE" \
171       create \
172       secret \
173       generic \
174       --type="kubernetes.io/rook" \
```

```
175         "rook-""$CSI_CEPHFS_PROVISIONER_SECRET_NAME" \
176         --from-literal=adminID="$CSI_CEPHFS_PROVISIONER_SECRET_NAME" \
177         --from-literal=adminKey="$CSI_CEPHFS_PROVISIONER_SECRET"
178     else
179       echo "secret $CSI_CEPHFS_PROVISIONER_SECRET_NAME already exists"
180     fi
181 }
182
183 function importRGWAdminOpsUser() {
184     if ! kubectl -n "$NAMESPACE" get secret
      ↪  "$RGW_ADMIN_OPS_USER_SECRET_NAME" &>/dev/null; then
185       kubectl -n "$NAMESPACE" \
186         create \
187         secret \
188         generic \
189         --type="kubernetes.io/rook" \
190         "$RGW_ADMIN_OPS_USER_SECRET_NAME" \
191         --from-literal=accessKey="$RGW_ADMIN_OPS_USER_ACCESS_KEY" \
192         --from-literal=secretKey="$RGW_ADMIN_OPS_USER_SECRET_KEY"
193     else
194       echo "secret $RGW_ADMIN_OPS_USER_SECRET_NAME already exists"
195     fi
196 }
197
198 function createECRBDStorageClass() {
```

```
199   if ! kubectl -n "$NAMESPACE" get storageclass
      ↪ $RBD_STORAGE_CLASS_NAME &>/dev/null; then
200     cat <<eof | kubectl create -f -
201 apiVersion: storage.k8s.io/v1
202 kind: StorageClass
203 metadata:
204   name: $RBD_STORAGE_CLASS_NAME
205 provisioner: $RBD_PROVISIONER
206 parameters:
207   clusterID: $CLUSTER_ID_RBD
208   pool: $RBD_METADATA_EC_POOL_NAME
209   dataPool: $RBD_POOL_NAME
210   imageFormat: "2"
211   imageFeatures: $ROOK_RBD_FEATURES
212   csi.storage.k8s.io/provisioner-secret-name:
      ↪ "rook-$CSI_RBD_PROVISIONER_SECRET_NAME"
213   csi.storage.k8s.io/provisioner-secret-namespace: $NAMESPACE
214   csi.storage.k8s.io/controller-expand-secret-name:
      ↪ "rook-$CSI_RBD_PROVISIONER_SECRET_NAME"
215   csi.storage.k8s.io/controller-expand-secret-namespace: $NAMESPACE
216   csi.storage.k8s.io/node-stage-secret-name:
      ↪ "rook-$CSI_RBD_NODE_SECRET_NAME"
217   csi.storage.k8s.io/node-stage-secret-namespace: $NAMESPACE
218   csi.storage.k8s.io/fstype: ext4
219 allowVolumeExpansion: true
```

```
220 reclaimPolicy: Delete
221 eof
222   else
223     echo "storageclass $RBD_STORAGE_CLASS_NAME already exists"
224   fi
225 }
226
227 function createRBDStorageClass() {
228   if ! kubectl -n "$NAMESPACE" get storageclass
      ↪  $RBD_STORAGE_CLASS_NAME &>/dev/null; then
229     cat <<eof | kubectl create -f -
230 apiVersion: storage.k8s.io/v1
231 kind: StorageClass
232 metadata:
233   name: $RBD_STORAGE_CLASS_NAME
234 provisioner: $RBD_PROVISIONER
235 parameters:
236   clusterID: $CLUSTER_ID_RBD
237   pool: $RBD_POOL_NAME
238   imageFormat: "2"
239   imageFeatures: $ROOK_RBD_FEATURES
240   csi.storage.k8s.io/provisioner-secret-name:
      ↪  "rook-$CSI_RBD_PROVISIONER_SECRET_NAME"
241   csi.storage.k8s.io/provisioner-secret-namespace: $NAMESPACE
```

```
242     csi.storage.k8s.io/controller-expand-secret-name:
        ↪ "rook-$CSI_RBD_PROVISIONER_SECRET_NAME"
243     csi.storage.k8s.io/controller-expand-secret-namespace: $NAMESPACE
244     csi.storage.k8s.io/node-stage-secret-name:
        ↪ "rook-$CSI_RBD_NODE_SECRET_NAME"
245     csi.storage.k8s.io/node-stage-secret-namespace: $NAMESPACE
246     csi.storage.k8s.io/fstype: ext4
247   allowVolumeExpansion: true
248   reclaimPolicy: Delete
249   eof
250     else
251       echo "storageclass $RBD_STORAGE_CLASS_NAME already exists"
252     fi
253   }
254
255   function createCephFSStorageClass() {
256     if ! kubectl -n "$NAMESPACE" get storageclass
        ↪ $CEPHFS_STORAGE_CLASS_NAME &>/dev/null; then
257       cat <<eof | kubectl create -f -
258   apiVersion: storage.k8s.io/v1
259   kind: StorageClass
260   metadata:
261     name: $CEPHFS_STORAGE_CLASS_NAME
262   provisioner: $CEPHFS_PROVISIONER
263   parameters:
```

```
264     clusterID: $CLUSTER_ID_CEPHFS
265     fsName: $CEPHFS_FS_NAME
266     pool: $CEPHFS_POOL_NAME
267     csi.storage.k8s.io/provisioner-secret-name:
        ↪ "rook-$CSI_CEPHFS_PROVISIONER_SECRET_NAME"
268     csi.storage.k8s.io/provisioner-secret-namespace: $NAMESPACE
269     csi.storage.k8s.io/controller-expand-secret-name:
        ↪ "rook-$CSI_CEPHFS_PROVISIONER_SECRET_NAME"
270     csi.storage.k8s.io/controller-expand-secret-namespace: $NAMESPACE
271     csi.storage.k8s.io/node-stage-secret-name:
        ↪ "rook-$CSI_CEPHFS_NODE_SECRET_NAME"
272     csi.storage.k8s.io/node-stage-secret-namespace: $NAMESPACE
273   allowVolumeExpansion: true
274   reclaimPolicy: Delete
275   eof
276     else
277       echo "storageclass $CEPHFS_STORAGE_CLASS_NAME already exists"
278     fi
279   }
280
281   ########
282   # MAIN #
283   ########
284   checkEnvVars
285   createClusterNamespace
```

```
importClusterID
importSecret
importConfigMap
if [ -n "$CSI_RBD_NODE_SECRET_NAME" ] && [ -n "$CSI_RBD_NODE_SECRET"
↪  ]; then
  importCsiRBDNodeSecret
fi
if [ -n "$CSI_RBD_PROVISIONER_SECRET_NAME" ] && [ -n
↪  "$CSI_RBD_PROVISIONER_SECRET" ]; then
  importCsiRBDProvisionerSecret
fi
if [ -n "$RGW_ADMIN_OPS_USER_ACCESS_KEY" ] && [ -n
↪  "$RGW_ADMIN_OPS_USER_SECRET_KEY" ]; then
  importRGWAdminOpsUser
fi
if [ -n "$CSI_CEPHFS_NODE_SECRET_NAME" ] && [ -n
↪  "$CSI_CEPHFS_NODE_SECRET" ]; then
  importCsiCephFSNodeSecret
fi
if [ -n "$CSI_CEPHFS_PROVISIONER_SECRET_NAME" ] && [ -n
↪  "$CSI_CEPHFS_PROVISIONER_SECRET" ]; then
  importCsiCephFSProvisionerSecret
fi
if [ -n "$RBD_POOL_NAME" ]; then
  if [ -n "$RBD_METADATA_EC_POOL_NAME" ]; then
```

```
306     createECRBDStorageClass
307   else
308     createRBDStorageClass
309   fi
310 fi
311 if [ -n "$CEPHFS_FS_NAME" ] && [ -n "$CEPHFS_POOL_NAME" ]; then
312   createCephFSStorageClass
313 fi
```

Code 4: import-external-cluster.sh

```
1  # All values below are taken from the CephCluster CRD
2  # -- Cluster configuration.
3  # @default -- See [below](#ceph-cluster-spec)
4  cephClusterSpec:
5    external:
6      enable: true
7    crashCollector:
8      disable: true
9    healthCheck:
10     daemonHealth:
11       mon:
12         disabled: false
13         interval: 45s
14 # -- A list of CephBlockPool configurations to deploy
15 # @default -- See [below](#ceph-block-pools)
```

```yaml
cephBlockPools: {}

# -- A list of CephFileSystem configurations to deploy
# @default -- See [below](#ceph-file-systems)
cephFileSystems: {}

# -- A list of CephObjectStore configurations to deploy
# @default -- See [below](#ceph-object-stores)
cephObjectStores: {}
```

Code 5: values-external.yaml

```yaml
# Default values for rook-ceph-operator
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

image:
  # -- Image
  repository: rook/ceph
  # -- Image tag
  # @default -- `master`
  tag: v1.13.7
  # -- Image pull policy
  pullPolicy: IfNotPresent

crds:
```

```
15    # -- Whether the helm chart should create and update the CRDs. If
      ↪  false, the CRDs must be
16    # managed independently with deploy/examples/crds.yaml.
17    # **WARNING** Only set during first deployment. If later disabled
      ↪  the cluster may be DESTROYED.
18    # If the CRDs are deleted in this case, see
19    # [the disaster recovery
      ↪  guide](https://rook.io/docs/rook/latest/Troubleshooting/disas
      ↪  ter-recovery/#restoring-crds-after-deletion)
20    # to restore them.
21    enabled: true
22
23  # -- Pod resource requests & limits
24  resources:
25    limits:
26      memory: 512Mi
27    requests:
28      cpu: 200m
29      memory: 128Mi
30
31  # -- Kubernetes [`nodeSelector`](https://kubernetes.io/docs/concept
    ↪  s/configuration/assign-pod-node/#nodeselector) to add to the
    ↪  Deployment.
32  nodeSelector: {}
```

```
33 # Constraint rook-ceph-operator Deployment to nodes with label
   ↪ `disktype: ssd`.
34 # For more info, see https://kubernetes.io/docs/concepts/configurat
   ↪ ion/assign-pod-node/#nodeselector
35 #  disktype: ssd
36
37 # -- List of Kubernetes [`tolerations`](https://kubernetes.io/docs/
   ↪ concepts/scheduling-eviction/taint-and-toleration/) to add to
   ↪ the Deployment.
38 tolerations: []
39
40 # -- Delay to use for the `node.kubernetes.io/unreachable` pod
   ↪ failure toleration to override
41 # the Kubernetes default of 5 minutes
42 unreachableNodeTolerationSeconds: 5
43
44 # -- Whether the operator should watch cluster CRD in its own
   ↪ namespace or not
45 currentNamespaceOnly: false
46
47 # -- Pod annotations
48 annotations: {}
49
50 # -- Global log level for the operator.
51 # Options: `ERROR`, `WARNING`, `INFO`, `DEBUG`
```

```
52 logLevel: INFO
53
54 # -- If true, create & use RBAC resources
55 rbacEnable: true
56
57 rbacAggregate:
58   # -- If true, create a ClusterRole aggregated to [user facing
   ↪ roles](https://kubernetes.io/docs/reference/access-authn-auth
   ↪ z/rbac/#user-facing-roles) for
   ↪ objectbucketclaims
59   enableOBCs: false
60
61 # -- If true, create & use PSP resources
62 pspEnable: false
63
64 # -- Set the priority class for the rook operator deployment if
   ↪ desired
65 priorityClassName:
66
67 # -- Set the container security context for the operator
68 containerSecurityContext:
69   runAsNonRoot: true
70   runAsUser: 2016
71   runAsGroup: 2016
72   capabilities:
```

```
73      drop: ["ALL"]
74  # -- If true, loop devices are allowed to be used for osds in test
    ↪  clusters
75  allowLoopDevices: false
76
77  # Settings for whether to disable the drivers or other daemons if
    ↪  they are not
78  # needed
79  csi:
80    # -- Enable Ceph CSI RBD driver
81    enableRbdDriver: true
82    # -- Enable Ceph CSI CephFS driver
83    enableCephfsDriver: true
84    # -- Enable host networking for CSI CephFS and RBD nodeplugins.
      ↪  This may be necessary
85    # in some network configurations where the SDN does not provide
      ↪  access to an external cluster or
86    # there is significant drop in read/write performance
87    enableCSIHostNetwork: true
88    # -- Enable Snapshotter in CephFS provisioner pod
89    enableCephfsSnapshotter: true
90    # -- Enable Snapshotter in NFS provisioner pod
91    enableNFSSnapshotter: true
92    # -- Enable Snapshotter in RBD provisioner pod
93    enableRBDSnapshotter: true
```

```
94    # -- Enable Host mount for `/etc/selinux` directory for Ceph CSI
      ↪ nodeplugins
95    enablePluginSelinuxHostMount: false
96    # -- Enable Ceph CSI PVC encryption support
97    enableCSIEncryption: false
98
99    # -- PriorityClassName to be set on csi driver plugin pods
100   pluginPriorityClassName: system-node-critical
101
102   # -- PriorityClassName to be set on csi driver provisioner pods
103   provisionerPriorityClassName: system-cluster-critical
104
105   # -- Policy for modifying a volume's ownership or permissions
      ↪ when the RBD PVC is being mounted.
106   # supported values are documented at
      ↪ https://kubernetes-csi.github.io/docs/support-fsgroup.html
107   rbdFSGroupPolicy: "File"
108
109   # -- Policy for modifying a volume's ownership or permissions
      ↪ when the CephFS PVC is being mounted.
110   # supported values are documented at
      ↪ https://kubernetes-csi.github.io/docs/support-fsgroup.html
111   cephFSFSGroupPolicy: "File"
112
```

```
113     # -- Policy for modifying a volume's ownership or permissions
        ↪ when the NFS PVC is being mounted.
114     # supported values are documented at
        ↪ https://kubernetes-csi.github.io/docs/support-fsgroup.html
115     nfsFSGroupPolicy: "File"
116
117     # -- OMAP generator generates the omap mapping between the PV
        ↪ name and the RBD image
118     # which helps CSI to identify the rbd images for CSI operations.
119     # `CSI_ENABLE_OMAP_GENERATOR` needs to be enabled when we are
        ↪ using rbd mirroring feature.
120     # By default OMAP generator is disabled and when enabled, it will
        ↪ be deployed as a
121     # sidecar with CSI provisioner pod, to enable set it to true.
122     enableOMAPGenerator: false
123
124     # -- Set CephFS Kernel mount options to use
        ↪ https://docs.ceph.com/en/latest/man/8/mount.ceph/#options.
125     # Set to "ms_mode=secure" when connections.encrypted is enabled
        ↪ in CephCluster CR
126     cephFSKernelMountOptions:
127
128     # -- Enable adding volume metadata on the CephFS subvolumes and
        ↪ RBD images.
```

```
129   # Not all users might be interested in getting volume/snapshot
      ↪  details as metadata on CephFS subvolume and RBD images.
130   # Hence enable metadata is false by default
131   enableMetadata: false
132
133   # -- Set replicas for csi provisioner deployment
134   provisionerReplicas: 2
135
136   # -- Cluster name identifier to set as metadata on the CephFS
      ↪  subvolume and RBD images. This will be useful
137   # in cases like for example, when two container orchestrator
      ↪  clusters (Kubernetes/OCP) are using a single ceph cluster
138   clusterName:
139
140   # -- Set logging level for cephCSI containers maintained by the
      ↪  cephCSI.
141   # Supported values from 0 to 5. 0 for general useful logs, 5 for
      ↪  trace level verbosity.
142   logLevel: 0
143
144   # -- Set logging level for Kubernetes-csi sidecar containers.
145   # Supported values from 0 to 5. 0 for general useful logs (the
      ↪  default), 5 for trace level verbosity.
146   # @default -- `0`
147   sidecarLogLevel:
```

```
148
149   # -- CSI driver name prefix for cephfs, rbd and nfs.
150   # @default -- `namespace name where rook-ceph operator is
      ↪  deployed`
151   csiDriverNamePrefix:
152
153   # -- CSI RBD plugin daemonset update strategy, supported values
      ↪  are OnDelete and RollingUpdate
154   # @default -- `RollingUpdate`
155   rbdPluginUpdateStrategy:
156
157   # -- A maxUnavailable parameter of CSI RBD plugin daemonset
      ↪  update strategy.
158   # @default -- `1`
159   rbdPluginUpdateStrategyMaxUnavailable:
160
161   # -- CSI CephFS plugin daemonset update strategy, supported
      ↪  values are OnDelete and RollingUpdate
162   # @default -- `RollingUpdate`
163   cephFSPluginUpdateStrategy:
164
165   # -- A maxUnavailable parameter of CSI cephFS plugin daemonset
      ↪  update strategy.
166   # @default -- `1`
167   cephFSPluginUpdateStrategyMaxUnavailable:
```

```
168
169   # -- CSI NFS plugin daemonset update strategy, supported values
      ↪  are OnDelete and RollingUpdate
170   # @default -- `RollingUpdate`
171   nfsPluginUpdateStrategy:
172
173   # -- Set GRPC timeout for csi containers (in seconds). It should
      ↪  be >= 120. If this value is not set or is invalid, it
      ↪  defaults to 150
174   grpcTimeoutInSeconds: 150
175
176   # -- Allow starting an unsupported ceph-csi image
177   allowUnsupportedVersion: false
178
179   # -- The volume of the CephCSI RBD plugin DaemonSet
180   csiRBDPluginVolume:
181   #  - name: lib-modules
182   #    hostPath:
183   #      path: /run/booted-system/kernel-modules/lib/modules/
184   #  - name: host-nix
185   #    hostPath:
186   #      path: /nix
187
188   # -- The volume mounts of the CephCSI RBD plugin DaemonSet
189   csiRBDPluginVolumeMount:
```

```
190     #  - name: host-nix
191     #    mountPath: /nix
192     #    readOnly: true
193
194     # -- The volume of the CephCSI CephFS plugin DaemonSet
195     csiCephFSPluginVolume:
196     #  - name: lib-modules
197     #    hostPath:
198     #      path: /run/booted-system/kernel-modules/lib/modules/
199     #  - name: host-nix
200     #    hostPath:
201     #      path: /nix
202
203     # -- The volume mounts of the CephCSI CephFS plugin DaemonSet
204     csiCephFSPluginVolumeMount:
205     #  - name: host-nix
206     #    mountPath: /nix
207     #    readOnly: true
208
209     # -- CEPH CSI RBD provisioner resource requirement list
210     # csi-omap-generator resources will be applied only if
        ↪ `enableOMAPGenerator` is set to `true`
211     # @default -- see values.yaml
212     csiRBDProvisionerResource: |
213       - name : csi-provisioner
```

```
214         resource:
215           requests:
216             memory: 128Mi
217             cpu: 100m
218           limits:
219             memory: 256Mi
220       - name : csi-resizer
221         resource:
222           requests:
223             memory: 128Mi
224             cpu: 100m
225           limits:
226             memory: 256Mi
227       - name : csi-attacher
228         resource:
229           requests:
230             memory: 128Mi
231             cpu: 100m
232           limits:
233             memory: 256Mi
234       - name : csi-snapshotter
235         resource:
236           requests:
237             memory: 128Mi
238             cpu: 100m
```

```
239           limits:
240             memory: 256Mi
241       - name : csi-rbdplugin
242         resource:
243           requests:
244             memory: 512Mi
245           limits:
246             memory: 1Gi
247       - name : csi-omap-generator
248         resource:
249           requests:
250             memory: 512Mi
251             cpu: 250m
252           limits:
253             memory: 1Gi
254       - name : liveness-prometheus
255         resource:
256           requests:
257             memory: 128Mi
258             cpu: 50m
259           limits:
260             memory: 256Mi
261
262     # -- CEPH CSI RBD plugin resource requirement list
263     # @default -- see values.yaml
```

```
264   csiRBDPluginResource: |
265     - name : driver-registrar
266       resource:
267         requests:
268           memory: 128Mi
269           cpu: 50m
270         limits:
271           memory: 256Mi
272     - name : csi-rbdplugin
273       resource:
274         requests:
275           memory: 512Mi
276           cpu: 250m
277         limits:
278           memory: 1Gi
279     - name : liveness-prometheus
280       resource:
281         requests:
282           memory: 128Mi
283           cpu: 50m
284         limits:
285           memory: 256Mi
286
287   # -- CEPH CSI CephFS provisioner resource requirement list
288   # @default -- see values.yaml
```

```
289   csiCephFSProvisionerResource: |
290     - name : csi-provisioner
291       resource:
292         requests:
293           memory: 128Mi
294           cpu: 100m
295         limits:
296           memory: 256Mi
297     - name : csi-resizer
298       resource:
299         requests:
300           memory: 128Mi
301           cpu: 100m
302         limits:
303           memory: 256Mi
304     - name : csi-attacher
305       resource:
306         requests:
307           memory: 128Mi
308           cpu: 100m
309         limits:
310           memory: 256Mi
311     - name : csi-snapshotter
312       resource:
313         requests:
```

```
314             memory: 128Mi
315             cpu: 100m
316           limits:
317             memory: 256Mi
318       - name : csi-cephfsplugin
319         resource:
320           requests:
321             memory: 512Mi
322             cpu: 250m
323           limits:
324             memory: 1Gi
325       - name : liveness-prometheus
326         resource:
327           requests:
328             memory: 128Mi
329             cpu: 50m
330           limits:
331             memory: 256Mi
332
333     # -- CEPH CSI CephFS plugin resource requirement list
334     # @default -- see values.yaml
335     csiCephFSPluginResource: |
336       - name : driver-registrar
337         resource:
338           requests:
```

```
339             memory: 128Mi
340             cpu: 50m
341           limits:
342             memory: 256Mi
343       - name : csi-cephfsplugin
344         resource:
345           requests:
346             memory: 512Mi
347             cpu: 250m
348           limits:
349             memory: 1Gi
350       - name : liveness-prometheus
351         resource:
352           requests:
353             memory: 128Mi
354             cpu: 50m
355           limits:
356             memory: 256Mi
357
358     # -- CEPH CSI NFS provisioner resource requirement list
359     # @default -- see values.yaml
360     csiNFSProvisionerResource: |
361       - name : csi-provisioner
362         resource:
363           requests:
```

```
364             memory: 128Mi
365             cpu: 100m
366           limits:
367             memory: 256Mi
368       - name : csi-nfsplugin
369         resource:
370           requests:
371             memory: 512Mi
372             cpu: 250m
373           limits:
374             memory: 1Gi
375       - name : csi-attacher
376         resource:
377           requests:
378             memory: 512Mi
379             cpu: 250m
380           limits:
381             memory: 1Gi
382
383   # -- CEPH CSI NFS plugin resource requirement list
384   # @default -- see values.yaml
385   csiNFSPluginResource: |
386       - name : driver-registrar
387         resource:
388           requests:
```

```
389           memory: 128Mi
390           cpu: 50m
391         limits:
392           memory: 256Mi
393       - name : csi-nfsplugin
394         resource:
395           requests:
396             memory: 512Mi
397             cpu: 250m
398           limits:
399             memory: 1Gi
400
401   # Set provisionerTolerations and provisionerNodeAffinity for
      ↪ provisioner pod.
402   # The CSI provisioner would be best to start on the same nodes as
      ↪ other ceph daemons.
403
404   # -- Array of tolerations in YAML format which will be added to
      ↪ CSI provisioner deployment
405   provisionerTolerations:
406   #    - key: key
407   #      operator: Exists
408   #      effect: NoSchedule
409
```

```
410   # -- The node labels for affinity of the CSI provisioner
      ↪ deployment [^1]
411   provisionerNodeAffinity: #key1=value1,value2; key2=value3
412   # Set pluginTolerations and pluginNodeAffinity for plugin
      ↪ daemonset pods.
413   # The CSI plugins need to be started on all the nodes where the
      ↪ clients need to mount the storage.
414
415   # -- Array of tolerations in YAML format which will be added to
      ↪ CephCSI plugin DaemonSet
416   pluginTolerations:
417   #    - key: key
418   #      operator: Exists
419   #      effect: NoSchedule
420
421   # -- The node labels for affinity of the CephCSI RBD plugin
      ↪ DaemonSet [^1]
422   pluginNodeAffinity: # key1=value1,value2; key2=value3
423
424   # -- Enable Ceph CSI Liveness sidecar deployment
425   enableLiveness: false
426
427   # -- CSI CephFS driver metrics port
428   # @default -- `9081`
429   cephfsLivenessMetricsPort:
```

```
430
431   # -- CSI Addons server port
432   # @default -- `9070`
433   csiAddonsPort:
434
435   # -- Enable Ceph Kernel clients on kernel < 4.17. If your kernel
      ↪  does not support quotas for CephFS
436   # you may want to disable this setting. However, this will cause
      ↪  an issue during upgrades
437   # with the FUSE client. See the [upgrade
      ↪  guide](https://rook.io/docs/rook/v1.2/ceph-upgrade.html)
438   forceCephFSKernelClient: true
439
440   # -- Ceph CSI RBD driver metrics port
441   # @default -- `8080`
442   rbdLivenessMetricsPort:
443
444   serviceMonitor:
445     # -- Enable ServiceMonitor for Ceph CSI drivers
446     enabled: false
447     # -- Service monitor scrape interval
448     interval: 10s
449     # -- ServiceMonitor additional labels
450     labels: {}
451     # -- Use a different namespace for the ServiceMonitor
```

```
452       namespace:
453
454     # -- Kubelet root directory path (if the Kubelet uses a different
        ↪  path for the `--root-dir` flag)
455     # @default -- `/var/lib/kubelet`
456     kubeletDirPath:
457
458     # -- Duration in seconds that non-leader candidates will wait to
        ↪  force acquire leadership.
459     # @default -- `137s`
460     csiLeaderElectionLeaseDuration:
461
462     # -- Deadline in seconds that the acting leader will retry
        ↪  refreshing leadership before giving up.
463     # @default -- `107s`
464     csiLeaderElectionRenewDeadline:
465
466     # -- Retry period in seconds the LeaderElector clients should
        ↪  wait between tries of actions.
467     # @default -- `26s`
468     csiLeaderElectionRetryPeriod:
469
470     cephcsi:
471       # -- Ceph CSI image
472       # @default -- `quay.io/cephcsi/cephcsi:v3.10.2`
```

```
473       image:
474
475     registrar:
476       # -- Kubernetes CSI registrar image
477       # @default -- `registry.k8s.io/sig-storage/csi-node-driver-regi
          ↪  strar:v2.10.0`
478       image:
479
480     provisioner:
481       # -- Kubernetes CSI provisioner image
482       # @default --
          ↪  `registry.k8s.io/sig-storage/csi-provisioner:v4.0.0`
483       image:
484
485     snapshotter:
486       # -- Kubernetes CSI snapshotter image
487       # @default --
          ↪  `registry.k8s.io/sig-storage/csi-snapshotter:v7.0.1`
488       image:
489
490     attacher:
491       # -- Kubernetes CSI Attacher image
492       # @default -- `registry.k8s.io/sig-storage/csi-attacher:v4.5.0`
493       image:
494
```

```
495   resizer:
496     # -- Kubernetes CSI resizer image
497     # @default -- `registry.k8s.io/sig-storage/csi-resizer:v1.10.0`
498     image:
499
500   # -- Image pull policy
501   imagePullPolicy: IfNotPresent
502
503   # -- Labels to add to the CSI CephFS Deployments and DaemonSets
      ↪ Pods
504   cephfsPodLabels: #"key1=value1,key2=value2"
505
506   # -- Labels to add to the CSI NFS Deployments and DaemonSets Pods
507   nfsPodLabels: #"key1=value1,key2=value2"
508
509   # -- Labels to add to the CSI RBD Deployments and DaemonSets Pods
510   rbdPodLabels: #"key1=value1,key2=value2"
511
512   csiAddons:
513     # -- Enable CSIAddons
514     enabled: false
515     # -- CSIAddons Sidecar image
516     image: "quay.io/csiaddons/k8s-sidecar:v0.8.0"
517
518   nfs:
```

```
519       # -- Enable the nfs csi driver
520       enabled: false
521
522     topology:
523       # -- Enable topology based provisioning
524       enabled: false
525       # NOTE: the value here serves as an example and needs to be
526       # updated with node labels that define domains of interest
527       # -- domainLabels define which node labels to use as domains
528       # for CSI nodeplugins to advertise their domains
529       domainLabels:
530       # - kubernetes.io/hostname
531       # - topology.kubernetes.io/zone
532       # - topology.rook.io/rack
533
534     readAffinity:
535       # -- Enable read affinity for RBD volumes. Recommended to
536       # set to true if running kernel 5.8 or newer.
537       # @default -- `false`
538       enabled: false
539       # -- Define which node labels to use
540       # as CRUSH location. This should correspond to the values set
541       # in the CRUSH map.
542       # @default -- labels listed
          ↪  [here](../CRDs/Cluster/ceph-cluster-crd.md#osd-topology)
```

```
543       crushLocationLabels:
544
545     # -- Whether to skip any attach operation altogether for CephFS
          ↪ PVCs. See more details
546     # [here](https://kubernetes-csi.github.io/docs/skip-attach.html#s
          ↪ kip-attach-with-csi-driver-object).
547     # If cephFSAttachRequired is set to false it skips the volume
          ↪ attachments and makes the creation
548     # of pods using the CephFS PVC fast. **WARNING** It's highly
          ↪ discouraged to use this for
549     # CephFS RWO volumes. Refer to this [issue](https://github.com/ku
          ↪ bernetes/kubernetes/issues/103305) for more
          ↪ details.
550     cephFSAttachRequired: true
551     # -- Whether to skip any attach operation altogether for RBD
          ↪ PVCs. See more details
552     # [here](https://kubernetes-csi.github.io/docs/skip-attach.html#s
          ↪ kip-attach-with-csi-driver-object).
553     # If set to false it skips the volume attachments and makes the
          ↪ creation of pods using the RBD PVC fast.
554     # **WARNING** It's highly discouraged to use this for RWO volumes
          ↪ as it can cause data corruption.
555     # csi-addons operations like Reclaimspace and PVC Keyrotation
          ↪ will also not be supported if set
```

```
556     # to false since we'll have no VolumeAttachments to determine
        ↪ which node the PVC is mounted on.
557     # Refer to this [issue](https://github.com/kubernetes/kubernetes/
        ↪ issues/103305) for more
        ↪ details.
558     rbdAttachRequired: true
559     # -- Whether to skip any attach operation altogether for NFS
        ↪ PVCs. See more details
560     # [here](https://kubernetes-csi.github.io/docs/skip-attach.html#s
        ↪ kip-attach-with-csi-driver-object).
561     # If cephFSAttachRequired is set to false it skips the volume
        ↪ attachments and makes the creation
562     # of pods using the NFS PVC fast. **WARNING** It's highly
        ↪ discouraged to use this for
563     # NFS RWO volumes. Refer to this [issue](https://github.com/kuber
        ↪ netes/kubernetes/issues/103305) for more
        ↪ details.
564     nfsAttachRequired: true
565
566   # -- Enable discovery daemon
567   enableDiscoveryDaemon: false
568   # -- Set the discovery daemon device discovery interval (default to
      ↪ 60m)
569   discoveryDaemonInterval: 60m
570
```

```
571 # -- The timeout for ceph commands in seconds
572 cephCommandsTimeoutSeconds: "15"
573
574 # -- If true, run rook operator on the host network
575 useOperatorHostNetwork:
576
577 # -- If true, scale down the rook operator.
578 # This is useful for administrative actions where the rook operator
    ↪  must be scaled down, while using gitops style tooling
579 # to deploy your helm charts.
580 scaleDownOperator: false
581
582 ## Rook Discover configuration
583 ## toleration: NoSchedule, PreferNoSchedule or NoExecute
584 ## tolerationKey: Set this to the specific key of the taint to
    ↪  tolerate
585 ## tolerations: Array of tolerations in YAML format which will be
    ↪  added to agent deployment
586 ## nodeAffinity: Set to labels of the node to match
587
588 discover:
589   # -- Toleration for the discover pods.
590   # Options: `NoSchedule`, `PreferNoSchedule` or `NoExecute`
591   toleration:
592   # -- The specific key of the taint to tolerate
```

```
593   tolerationKey:
594   # -- Array of tolerations in YAML format which will be added to
      ↪  discover deployment
595   tolerations:
596   #   - key: key
597   #     operator: Exists
598   #     effect: NoSchedule
599   # -- The node labels for affinity of `discover-agent` [^1]
600   nodeAffinity:
601   #   key1=value1,value2; key2=value3
602   #
603   #   or
604   #
605   #   requiredDuringSchedulingIgnoredDuringExecution:
606   #     nodeSelectorTerms:
607   #       - matchExpressions:
608   #           - key: storage-node
609   #             operator: Exists
610   # -- Labels to add to the discover pods
611   podLabels: # "key1=value1,key2=value2"
612   # -- Add resources to discover daemon pods
613   resources:
614   #   - limits:
615   #       memory: 512Mi
616   #   - requests:
```

```
617   #       cpu: 100m
618   #       memory: 128Mi
619
620 # -- Runs Ceph Pods as privileged to be able to write to
    ↪ `hostPaths` in OpenShift with SELinux restrictions.
621 hostpathRequiresPrivileged: false
622
623 # -- Disable automatic orchestration when new devices are
    ↪ discovered.
624 disableDeviceHotplug: false
625
626 # -- Blacklist certain disks according to the regex provided.
627 discoverDaemonUdev:
628
629 # -- imagePullSecrets option allow to pull docker images from
    ↪ private docker registry. Option will be passed to all service
    ↪ accounts.
630 imagePullSecrets:
631 # - name: my-registry-secret
632
633 # -- Whether the OBC provisioner should watch on the operator
    ↪ namespace or not, if not the namespace of the cluster will be
    ↪ used
634 enableOBCWatchOperatorNamespace: true
635
```

```
636 monitoring:
637   # -- Enable monitoring. Requires Prometheus to be pre-installed.
638   # Enabling will also create RBAC rules to allow Operator to
      ↪  create ServiceMonitors
639   enabled: false
```

Code 6: values.yaml

```
1 clusterNamespace=rook-ceph
2 operatorNamespace=rook-ceph
3 cd <helm-directory>
4 helm repo add rook-release https://charts.rook.io/release
5 helm install --create-namespace --namespace $clusterNamespace
   ↪  rook-ceph rook-release/rook-ceph -f values.yaml
6 helm install --create-namespace --namespace $clusterNamespace
   ↪  rook-ceph-cluster \
7 --set operatorNamespace=$operatorNamespace
   ↪  rook-release/rook-ceph-cluster -f values-external.yaml
```

Code 7: rook_helm.sh