DATA PROTECTION FOR KUBERNETES USING COHESITY DATAPLATFORM AND HPE APOLLO SERVERS
EXECUTIVE SUMMARY

Container technology adoption is increasing at a fast pace. The global application container market size was valued at USD 1.5 billion in 2018 and is expected to reach USD 8.2 billion by 2025, according to a report by Grand View Research, Inc., registering a 26.5% compound annual growth rate during the forecast period.¹

The rapid container technology adoption growth is due to the ease that containerized applications can be ported and deployed across different environments. Containers package the applications with everything they need to run with and isolate them within the deployment environment. This enables containerized applications to easily run on different environments such as local desktops, virtual and physical servers, development, testing and production environments, and private or public clouds.

Another factor driving growth of containers is their increasing prevalence for deploying artificial intelligence (AI) and analytics applications. Containers are becoming the standard way to build and deploy machine learning (ML) models, create real-time analytics pipelines, and run batch analytics and ETL jobs. Their portability across different environments makes containers the perfect vehicle to manage the full lifecycle of AI/ML models and most analytics applications.

With the massive adoption of containers for analytics and AI/ML, data protection requirements are inevitable. These requirements include recovering containerized applications from failures and disasters, replicating environments for migrating a test and development environment to production, or replicating environments from production to staging before an upgrade, and being able to move container clusters more easily.

To protect containerized applications, there are key requirements that must be met. These include seamless operations and policies across on-premises and clouds, operational simplicity for container deployment and data management policies that can span across multiple environments, and backup/restore at the application level—not at VM/server level.

Hewlett Packard Enterprise offers a data protection solution for a Kubernetes environment deployed on HPE Apollo systems with Cohesity® DataPlatform®. The solution leverages the flexibility and high-density storage optimization for Big Data on HPE Apollo systems by deploying Cohesity DataPlatform to protect a containerized application—to meet the needs of demanding AI/ML and Deep Learning (DL) workloads—with the right compute, flexible I/O, and storage options.

Target audience: Presales consultants, solution architects, storage operators, data center managers, enterprise architects, and deployment and implementation engineers. A working knowledge of Kubernetes and Cohesity DataPlatform is recommended.

Document purpose: The purpose of this document is to describe a solution that highlights how a containerized application, such as MySQL in a Kubernetes environment running on HPE Apollo servers, can be protected by Cohesity DataPlatform.

This technical white paper describes solution testing performed by Hewlett Packard Enterprise in February 2021.

INTRODUCTION

The combination of HPE Apollo systems and Cohesity DataPlatform provides end-to-end enterprise-grade container protection, safeguarding containers wherever they live (on-premises or the cloud or hybrid) and restoring them wherever they are needed. Through the HPE Complete program, Hewlett Packard Enterprise provides one-stop shopping where customers can purchase validated turnkey backup and recovery solutions, reducing risk and improving recovery readiness while protecting their data. The Cohesity solution available through HPE Complete, provides a converged web-scale data management platform that consolidates all secondary workloads—including storage and data services—onto one unified, efficient solution. The HPE and Cohesity solution modernizes and simplifies data and application management by providing one platform for multiple workloads.

One of the key challenges for protecting Kubernetes environments is managing the dynamic deployment, because the containers are not bundled with physical servers or virtual machines. Containerized applications can run on different environments such as local desktops, virtual and physical servers, development, testing, and production environments, and private or public clouds. Enterprises not only need to quickly deploy containerized AI and analytics applications but also protect and restore these applications. Supporting containers with data backup and mobility at scale is a critical need for AI/ML environments. The HPE and Cohesity solution simplifies data protection, consolidates file and object services, provides instant access to test/dev copies, and supports in-place search and analytics. This converged platform provides a holistic solution that advances user operations, enhances reliability, and radically reduces cost and complexity.

This technical white paper provides an overview of the requirements for backing up a containerized application in a Kubernetes environment using Cohesity DataPlatform, then restoring that containerized application.

¹ grandviewresearch.com/press-release/global-application-container-market
SOLUTION COMPONENTS

Hewlett Packard Enterprise validated protecting a containerized application in a Kubernetes environment using Cohesity DataPlatform in a lab environment. This section provides details on each major component incorporated into the solution.

Additional guidance regarding HPE Solutions for Cohesity DataPlatform can be found in HPE Solutions for Cohesity DataPlatform QuickSpecs.

FIGURE 1 illustrates the lab environment’s configuration.

FIGURE 1. Environment overview

HPE Apollo 2000 servers

New infrastructure must support a diversity of processor technologies and data-intensive workloads in the architecture to enable the converged use of analytics, AI, and high performance computing (HPC) to unlock the potential of your data and accelerate innovation. HPE Apollo systems deliver the flexibility to tailor the system to the precise needs of demanding HPC workloads with the right compute, flexible I/O, and storage options.

The HPE Apollo 2000 system provides a density-optimized compute platform that doubles the density of traditional 1U servers, providing up to four 2P servers in a 2U form factor. The HPE ProLiant XL170r Gen10 2P server is an ideal node for compute-intensive and in-memory analytics in real-time, interactive, and batch workloads.

The Kubernetes environment used for this testing was deployed on an HPE Apollo 2000 Gen10 server with multiple HPE ProLiant XL170r Gen10 servers. The Kubernetes cluster had version 1.18.6 installed.

HPE ProLiant DL380 Gen10 servers

The HPE ProLiant DL family of servers delivers a reliable, fast, and secure infrastructure solution, which helps increase IT staff productivity and accelerates service delivery. In addition, the rack portfolio is performance optimized for multi-application workloads to significantly increase the speed of IT operations and enable IT to respond to business needs of any size, faster.

Cohesity DataPlatform, version 6.5.1d, was installed on three HPE ProLiant DL380 Gen10 servers.

Cohesity DataPlatform

Cohesity DataPlatform consolidates secondary data and applications including backups, files, objects, test/dev, and analytics on a single, software-defined platform. Inspired by web-scale architecture, DataPlatform is a scale-out solution based on a unique distributed file system, SpanFS®. DataPlatform modernizes and simplifies secondary data and application management by providing one platform for multiple secondary workloads.

TESTING OVERVIEW

Installation and configuration guidance

Cohesity DataPlatform documentation for version 6.5 can be accessed using this link: docs.cohesity.com/6.5/Web/UserGuide/Content/Welcome/Welcome.htm
The initial Cohesity DataPlatform setup was performed by leveraging the following resources from Cohesity Documentation (account required):

- **On-Premises Cluster Setup**, then expanded HPE ProLiant DL380 Gen10, which contained two PDFs:
  - Worksheet that contained a checklist of network requirements and worksheets required to install and configure the Cohesity cluster.
  - HPE ProLiant DL380 Gen10 Setup Guide provided the complete initial hardware and software configuration instructions.

A summary of the steps performed to protect a Kubernetes namespace using Cohesity DataPlatform were:

1. Applied GFlags to Cohesity DataPlatform to enable the backup/recovery of Kubernetes namespace
2. Configured the Cohesity S3 server to run in insecure mode (required for Velero)
3. Extracted the Bearer token ID of the Kubernetes service account (required to register the Kubernetes cluster)
4. Obtained files from Cohesity Support that were used to deploy the third-party software, Restic and Velero, for backing up Kubernetes namespaces
5. Registered the Kubernetes cluster as a source on Cohesity DataPlatform
6. Deployed Velero and Restic on the Kubernetes cluster
7. Created a Protection Group in Cohesity DataPlatform that was used to back up the Kubernetes namespace
8. Restored the Kubernetes namespace to a new namespace using the previously created Protection Group

**Preparing Cohesity DataPlatform**

A few steps had to be performed on Cohesity DataPlatform before it was possible to protect a namespace on the Kubernetes cluster. First, to enable Kubernetes cluster protection groups to be created, the required GFlags had to be applied on Cohesity DataPlatform. The online documentation, How to enable Kubernetes protection in Cohesity 6.5.1 (early access), provides detailed steps for this. After the GFlags were applied, the Cohesity S3 server needed to be run in insecure mode for Velero to protect the Kubernetes cluster. Then the Bearer token ID of the Kubernetes service account was extracted so the Kubernetes cluster could be registered with Cohesity DataPlatform. Cohesity Support is needed to run the Cohesity S3 server in insecure mode and detailed steps for extracting the Bearer token are provided in the online documentation: Kubernetes Cluster Requirements. While contacting Cohesity Support for assistance with running the Cohesity S3 server in insecure mode, the k8s_helper.py and cloud_credentials.json files were also requested. Those two files were later used to deploy the third-party software Velero and Restic on the Kubernetes cluster. The final preparation of Cohesity DataPlatform required registering the Kubernetes cluster as a source. Detailed steps are provided in the online documentation: Setup Kubernetes Cluster. FIGURE 2 and FIGURE 3 show the required fields to register the Kubernetes cluster as a source and the Kubernetes cluster successfully added as a source.

**NOTE**

Run the command `kubectl describe service kubernetes` on the master node in the Kubernetes cluster to obtain the REST endpoint. The REST endpoint will be entered as the hostname or IP address when registering the Kubernetes cluster. This value will also be used when executing the k8s_helper.py script later.

---

**FIGURE 2** Registering the Kubernetes cluster as a source
Preparing the Kubernetes cluster

After the Kubernetes cluster was registered as a source on Cohesity DataPlatform, the k8s_helper.py and cloud_credentials.json files that were obtained from Cohesity Support (in a TAR package) were used to deploy Velero and Restic for backing up Kubernetes namespaces. The online document, Setup Kubernetes Cluster, was leveraged when deploying the third-party software. A summary of the steps used to deploy Velero and Restic were:

1. Get the Cohesity DataPlatform’s S3 “Secret Access Key” and “Secret Access Key ID.”
2. Extract the contents of the TAR package received from Cohesity Support.
3. Insert the S3 credentials in the cloud_credentials.json file.
4. Get the encoded base64 string for the secret file in the v1.4 folder.
5. Edit the cloud_credentials.json file to replace the encoded value for “cloud.”
6. Run the k8s_helper.py script (requires the previously extracted Bearer token ID for the Kubernetes cluster).

The S3 credentials for Cohesity DataPlatform were captured by selecting Settings → Access Management then choosing the admin account. FIGURE 4 shows where the Cohesity S3 access keys are in the GUI.
FIGURE 4 Where to view and copy the Cohesity S3 Access Keys

Extracting the contents of the TAR package received from Cohesity Support created a directory named deployment that contained `k8s_helper.py`, `cloud_credentials.json`, and additional directories for the version of Velero to be used. For this testing, files in the `v1.4` folder were used as seen in FIGURE 5.

FIGURE 5 Files from Cohesity Support that were used to deploy Velero and Restic

FIGURE 6 shows viewing the S3 secret on the Kubernetes cluster then generating the base64 encoded string for the file.

FIGURE 6 Generating the base64 encoded string for the S3 secret

FIGURE 7 shows the output for the `cloud_credentials.json` file with the base64 encoded string entered for the “cloud” credentials.

FIGURE 7 Output for the updated `cloud_credentials.json` file
With the required files in place and edited accordingly on the Kubernetes cluster, the `k8s_helper.py` script was executed to deploy Velero and Restic. Syntax for the command was:

```bash
[root@master node for Kubernetes cluster]#/k8s_helper.py --cluster_endpoint <same value previously used to register the Kubernetes cluster> --version v1.4 --token <Bearer token ID>
```

After the `k8s_helper.py` script successfully completed, the status of the Velero and Restic deployments were verified as seen in **FIGURE 8**.

**FIGURE 8** How to verify the status of the Velero and Restic deployments

**NOTE**
If running the `k8s_helper.py` script results in an unauthorized error, add the admin privilege to the user used to deploy Velero (in this case, "default" user from the "default" namespace) by executing the following command on the master node of the Kubernetes cluster:

```
kubectl create clusterrolebinding cohesity-admin --clusterrole=cluster-admin --serviceaccount=default:default
```

### Setup hostpath CSI driver on the Kubernetes cluster

There was ample storage on the HPE Apollo 2000 Gen10 server that was used to deploy the Kubernetes cluster in order to meet the testing requirements. Due to the storage being local and how the Kubernetes cluster was deployed, the 'hostpath' for the containerized application was empty (no value). Hostpath is a storage type where volumes refer to directories on the node where a pod is scheduled for running. Hostpath CSI driver is a sample (non-production) CSI driver that creates a local directory as a volume on a single node.

**NOTE**
For production environments, Hewlett Packard Enterprise does not recommend using a sample CSI driver. HPE recommends using a supported CSI driver that is relevant to your storage provider. A detailed list of production CSI drivers that are supported with Cloud Native Computing Foundation (CNCF)-certified Kubernetes distributions is available online at [kubernetes-csi.github.io/docs/drivers.html](https://kubernetes-csi.github.io/docs/drivers.html).

The following steps were performed to set up hostpath CSI driver on the Kubernetes cluster:

1. Install the required custom resources (CRDs) on the Kubernetes cluster.
2. Clone the csi-driver-host-path repository on one of the Kubernetes nodes and deploy the driver.
3. Validate that the hostpath driver pods are running in the default namespace on the Kubernetes cluster.
4. Edit the `csi-automation-setup.yaml` file to deploy a storage class that would use the CSI drivers.
The commands used to deploy the required CRDs on the Kubernetes cluster were:

```bash
# SNAPSHOTTER_VERSION=v2.0.1
# # Apply VolumeSnapshot CRDs
# kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/${SNAPSHOTTER_VERSION}/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
# kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/${SNAPSHOTTER_VERSION}/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yaml
# kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/${SNAPSHOTTER_VERSION}/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
# # Create snapshot controller
# git clone https://github.com/kubernetes-csi/csi-driver-host-path.git
# cd csi-driver-host-path/deploy/kubernetes-1.18/
# ./deploy.sh
```

Then the csi-driver-host-path repository was cloned, and the driver deployed using these commands:

```bash
# git clone https://github.com/kubernetes-csi/csi-driver-host-path.git
# cd csi-driver-host-path/deploy/kubernetes-1.18/
# ./deploy.sh
```

Next, run `kubectl get pods` to confirm the csi-driver-host-path pods were successfully deployed and running. The output should look like this:

```
NAME                         READY   STATUS    RESTARTS   AGE
csi-hostpath-attacher-0      1/1     Running   0          107s
csi-hostpath-provisioner-0   1/1     Running   0          105s
csi-hostpath-resizer-0       1/1     Running   0          104s
csi-hostpath-snapshotter-0   1/1     Running   0          104s
csi-hostpath-socat-0         1/1     Running   0          103s
csi-hostpathplugin-0         5/5     Running   0          105s
snapshot-controller-0        1/1     Running   0          5m6s
```

Finally, in the same directory as the `deploy.sh` script, a `csi-automation-setup.yaml` file can be found. That file was edited as follows to create a storage class that would use the csi-driver-host-path:

```
# cat csi-driver-host-path/deploy/kubernetes-1.18/csi-automation-setup.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-hostpath-sc
provisioner: hostpath.csi.k8s.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
allowVolumeExpansion: true
```

**Protecting a Kubernetes namespace**

With the Cohesity DataPlatform and Kubernetes cluster successfully configured and the Velero and Restic software deployed, a Protection Group was created in Cohesity DataPlatform to back up a Kubernetes namespace. The online documentation, Protect a Kubernetes Cluster, provides step-by-step instructions for creating a Protection Group. **FIGURE 9** **FIGURE 10** and **FIGURE 11** illustrate selections made when creating the Protection Group used for this testing.
FIGURE 9 Click “Protect” to begin creating the Protection Group

FIGURE 10 Selecting a namespace for the Protection Group
FIGURE 11  All selections for the new Protection Group, click “Protect” to start the backup

The new Protection Group was listed on the Protection Groups page, and by selecting it, the results of the backup task could be confirmed as seen in FIGURE 12.

FIGURE 12  Confirming the backup task successfully completed
Recovering a Kubernetes namespace

With the containerized application successfully protected, the application could be restored to the following:

- The original location (assuming the application is not functioning as expected).
- To a new location on the same Kubernetes cluster or to another Kubernetes cluster (for example, to deploy the application in a new environment, or to create a different instance). This capability can be used to perform application migration for the purposes of cross-cluster migration, cluster lifecycle, or just storage lifecycle within an existing cluster. As undesired cluster sprawl happens, application mobility is key for Kubernetes teams.

For this testing, the containerized application was restored to the same Kubernetes cluster. See the online documentation, Recover Kubernetes Cluster, for detailed steps. FIGURE 13 shows how to start the restore process.

FIGURE 13 Selecting to recover data for a Kubernetes cluster

After selecting to recover data for a Kubernetes cluster, the previously protected namespace was chosen to be restored as shown in FIGURE 14.

FIGURE 14 Selecting the containerized application for restore
FIGURE 15 illustrates the selections made to restore the application to a new namespace.

A MySQL application was restored to a new namespace using the prefix “restored-” and the suffix “-fortest.”

For this testing, the use case was that one user had successfully deployed and populated a MySQL application on the Kubernetes cluster. Another user wanted to deploy a MySQL application and leverage the data the first user had already created. This type of restore allows each user to have their own instance of the MySQL application and data to work with.

The restore operation was started by clicking the Start Recovery button, shown in FIGURE 15.

After the restore job successfully completed, the restored application and data were confirmed. For this testing, the original MySQL application and its data were confirmed and then verified with the restored MySQL application and its data using kubectl commands executed on one of the Kubernetes master nodes.

First, all namespaces for the Kubernetes cluster were viewed followed by pods for the “mysql-app-dev1” namespace and the “restored-mysql-app-dev1-fortest” as shown in FIGURE 16. The MySQL application that was protected and the restored MySQL application should have pods with the same names but different ages.

FIGURE 15 Selecting to restore the containerized application to a new namespace

FIGURE 16 Viewing the pods for the original and restored MySQL namespaces
The original MySQL application was logged in to, databases viewed, and data for a specific database was displayed as illustrated in FIGURE 17. Note that the selected namespace was “mysql-app-dev1.”

And finally, to confirm the restore was successful, the databases and data for the same database were viewed for the restored MySQL application as seen in FIGURE 18. Note that the selected namespace was “restored-mysql-app-dev1-fortest.”
This technical white paper has successfully demonstrated how to protect a containerized application in a Kubernetes environment deployed on HPE Apollo systems using Cohesity DataPlatform. The steps provided offer guidance for the components and configuration required to add a Kubernetes cluster to Cohesity DataPlatform and then back up and restore a containerized application. The specific use case tested was backing up a containerized application, then restoring that application using a different name. The use case was chosen because it is common for one user to deploy an application in a Kubernetes environment and then customize that application with specific data to suit their needs. Subsequently, another user might want to leverage that application already created to save time rather than deploying their own application and customizing it. Restoring a backup of the original application to a new name provides both users with their own versions of the application to work with.

Due to the dynamic nature of Kubernetes environments, enterprises must be able to quickly deploy containerized AI and analytics applications typically in a variety of environments. That can include physical servers or virtual machines residing on-premises, in private or public cloud. Enterprises need a solution with the compute resources required for supporting containers, including data backup and mobility at scale.

Deploying a Kubernetes environment on HPE Apollo systems provided flexibility to tailor the system to the precise needs of demanding HPC workloads with the right compute, flexible I/O, and storage options. Cohesity DataPlatform was utilized to protect a containerized application with the possibility to move that data across on-premises and the cloud.

Hewlett Packard Enterprise and Cohesity provide end-to-end enterprise-grade data protection—safeguarding data wherever it lives while also improving business continuity.
RESOURCES AND ADDITIONAL LINKS

HPE Apollo Systems
hpe.com/apollo

HPE ProLiant DL Servers

Cohesity DataPlatform
cohesity.com/products/dataplatform/

Cohesity DataPlatform version 6.5 documentation
docs.cohesity.com/6.5/Web/UserGuide/Content/Welcome/Welcome.htm

Cohesity DataPlatform version 6.5 Kubernetes documentation
docs.cohesity.com/6.5/Web/UserGuide/Content/Doc/Kubernetes.htm?tocpath=Kubernetes%20Cluster%7C..._0

HPE Complete

HPE Complete Cohesity Solution
buy.hpe.com/us/en/storage/complete-storage-solution/complete-storage-solution/complete-partner-program/cohesity/p/1009514534

HPE Solutions for Cohesity DataPlatform guidelines
h20195.www2.hpe.com/v2/getdocument.aspx?docname=a00061622enw#

Kubernetes supported CSI drivers
kubernetes-csi.github.io/docs/drivers.html

Cloud Native Computing Foundation-certified Kubernetes distributions
landscape.cncf.io/card-mode?category=certified-kubernetes-distribution%2Ccertified-kubernetes-hosted&grouping=category

LEARN MORE AT
hpe.com/solutions