


From The  Architects

# Designing a Kubernetes Disaster Recovery strategy

Patricio Cerda

2023 v1.0

## Contents

Executive Summary .....	4
Why do we need a Kubernetes Disaster Recovery Strategy? .....	4
Microservices and Kubernetes .....	4
Stateful vs Stateless applications .....	6
Do we actually need to protect Kubernetes workloads? .....	6
Choosing the right solution .....	7
Kubernetes Disaster Recovery Strategy .....	9
Kubernetes Native Backup Solution .....	9
CI/CD for automation and rapid rollbacks.....	9
Where to store my backups? .....	10
Approaches for Disaster Recovery strategy with Kasten .....	10
Option 1: Disaster Recovery using Kasten Export/Import features.....	11
Stage 1: Protecting our applications .....	12
Configuring Location Profiles .....	12
Creating Kasten K10 Export policies .....	14
Getting the Import Data.....	15
Stage 2: Importing the applications in the target cluster.....	17
Configuring Location Profiles .....	17
Creating Kasten K10 Import policies .....	17
Stage 3: Restoring Applications after a Disaster .....	19
“Restore after Import” option NOT selected .....	19

“Restore after Import” option selected .....	24
Stage 4: Make the applications accessible .....	25
Option 2: Disaster Recovery using Kasten DR feature .....	27
Stage 1: Enable Kasten Disaster Recovery feature .....	28
Stage 2: Protect our applications .....	30
Stage 3: Restore Kasten configuration after a Disaster .....	31
Stage 4: Recover the cluster-scoped resources. ....	33
Stage 5: Restore Applications.....	34
Stage 6: Make the applications accessible .....	37
How to choose the Disaster Recovery Strategy .....	39
Using Kasten Export/Import features .....	39
Pros .....	39
Cons .....	39
Using Kasten Disaster Recovery feature .....	39
Pros .....	39
Cons .....	40
How to Automate the Disaster Recovery Strategy .....	40
Deploying the Kubernetes clusters .....	40
Deploying Kasten K10.....	40
Exporting and Importing Applications with Kasten.....	41
Using the Kasten Disaster Recovery feature .....	41
Restoring applications .....	41
About the Author.....	42
About Kasten by Veeam .....	42

## Executive Summary

Applications have been quickly evolving from monolithic and virtualized approaches to microservice-based architectures, where Kubernetes has emerged to become the de-facto container orchestration platform.

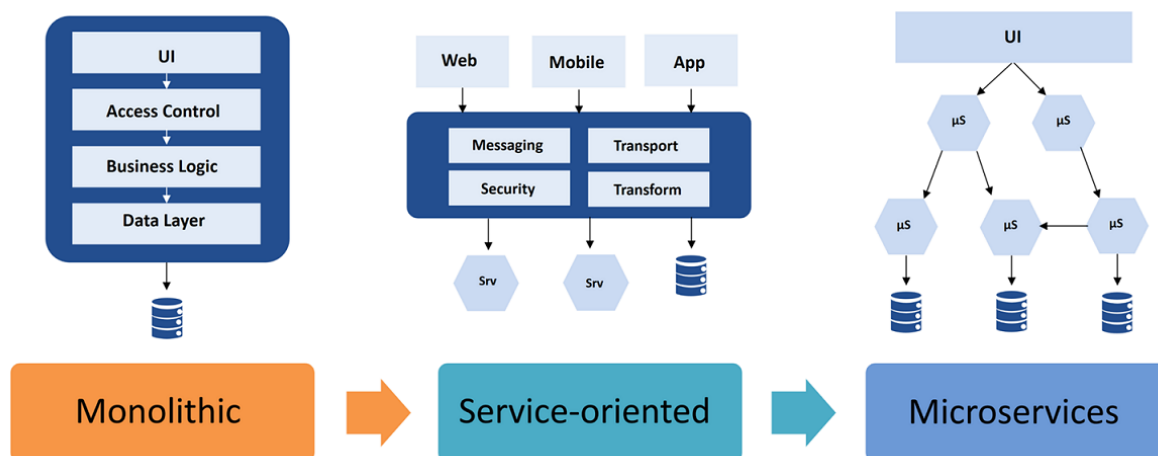
Regardless the application's architecture, data is always going to be the most important asset in any company, so protecting it is paramount. Therefore, as with any other approach, when using microservices with Kubernetes it is key to have a proper Data Protection strategy, including a proper Disaster Recovery Plan.

This whitepaper provides some recommendations and best practices to create a Disaster Recovery Strategy for a Kubernetes infrastructure, and all the applications running on it, by using Kasten by Veeam. These recommendations include the use of automation tools, which make it easier to design of the Kubernetes Disaster Recovery strategy in a DevOps context.

## Why do we need a Kubernetes Disaster Recovery Strategy?

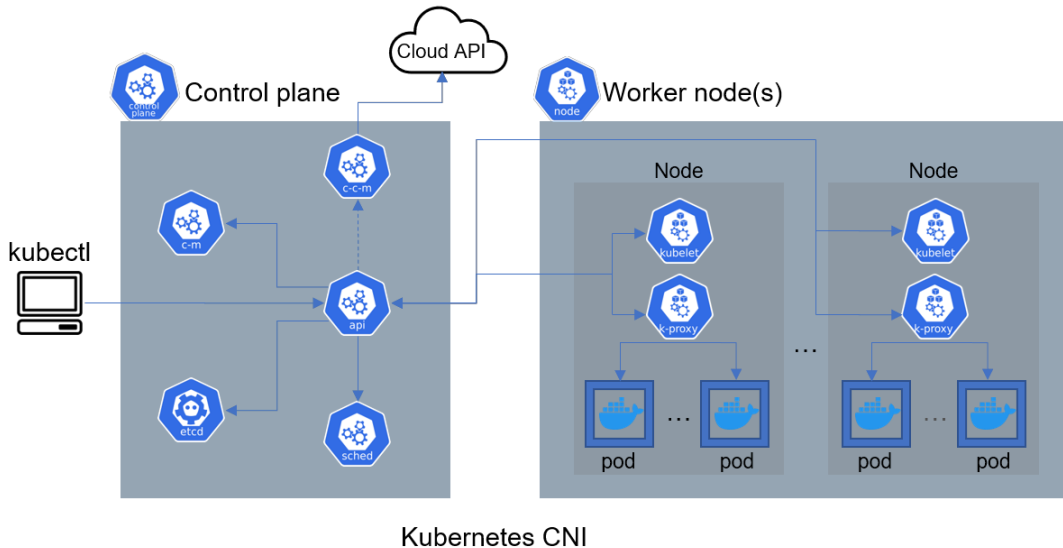
### Microservices and Kubernetes

Microservices (sometimes referred simply as containers) are at the heart of cloud-native business transformation initiatives, and they are a natural evolution from virtual machines to a more granular and portable application environment.



Containers are designed to support rapid development and deployment of cloud-native applications in a DevOps model. This DevOps model is basically a set of practices that combines software development and IT operations, where development and operations teams are no longer separated silos but are now working together in a more flexible work model.

As applications grew to span multiple containers across multiple servers, Kubernetes emerged to become the *de-facto* container orchestration platform, designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability, and high availability.



Kubernetes, a powerful open-source system, initially developed by Google and now being managed by the community and the Cloud Native Computing Foundation (CNCF), it's not just an orchestration system. Kubernetes comprises a set of independent, composable control processes that continuously drive the current state towards the provided desired state. Centralized control is also not required. This results in a system that is easier to use and more powerful, robust, resilient, and extensible, while eliminating the complexities of infrastructure management, deployment and scalability of cloud applications

There are many options to deploy Kubernetes, whether in the cloud or on-premises, depending on your business and technical requirements. Some of the more popular distributions to deliver Kubernetes are:

- RedHat OpenShift
- Amazon Web Services – Elastic Kubernetes Service (EKS)
- Microsoft Azure – Azure Kubernetes Service (AKS)
- Google Cloud Platform – Google Kubernetes Engine (GKE)
- SUSE Rancher
- VMware Tanzu

More information about what Kubernetes is and what Kubernetes is not, can be found in Kubernetes official documentation: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

## Stateful vs Stateless applications

Initially, early adopters of a Microservices approach and of Kubernetes were mainly focused on deliver stateless applications. Stateless applications don't keep data/information in it, which means no databases, no writes of any data inside the application, and no left-over files when the pod is deleted. Still, a stateless application can solve complicated problems by just receiving an input and performing actions which depend on that specific input and the "state" of the application. These states are pre-defined in the application itself.

Some examples of Stateless applications are Web Applications like Apache, NGINX, etc.

As the cloud-native business transformation initiatives started to growth and get more mature, companies began to deliver also stateful applications in Kubernetes. Stateful applications are basically applications that can store information. Here is important to highlight that one of the fundamental principles of microservices is that no data should be stored **within** the container itself. So, when we talk about Stateful applications keeping persistent data, this data is stored **outside** the pod itself, but can be accessed and/or processed by it.

There are different approaches to store this information, depending on application's requirements:

- Information could be stored on Persistent Volumes (PV) whether locally or using remote storage solutions.
- Information could be stored on Database services like MySQL, PostgreSQL, MongoDB, ElasticSearch, Microsoft SQL Server, AWS RDS, and so on.
- Information could also be stored in ConfigMaps, Secrets and others.

## Do we actually need to protect Kubernetes workloads?

It's a common mistake to think that protecting Kubernetes workloads by providing a Backup and Disaster Recovery solution isn't actually necessary, whether for the availability and reliability inherited to the Kubernetes architecture, for the redundancy and availability provided by IaaS providers like AWS with Elastic Block Storage (or even by Storage Snapshots available in on-premises solutions), or for the possibility of quickly redeploying applications from code using solutions like GitOps (Infrastructure as Code - IaC) or version control solutions.

These are myths and it's important to understand the Data Protection gap between the mentioned technologies or features, and a complete Kubernetes Disaster Recovery strategy. For instance:

- Kubernetes architecture is designed for fault tolerance, which makes easier to ensure application uptime even if we face partial infrastructure outages. However, it's important to remember that the high availability or replication provided natively by Kubernetes it's not actually a backup, as these features aren't meant to be a Data Protection solution by themselves. You still can suffer from data corruption, or accidental or malicious data deletion, all of which can lead to a catastrophic data loss.
- Etcd backups only capture the state of the Etcd cluster, which includes configuration data, secrets, and other important information necessary for the operation of a Kubernetes cluster. While this information is critical, it doesn't capture the state of the actual application workloads running in the cluster. Kubernetes is often used to deploy complex, multi-tiered applications, which include data stored in persistent volumes and application state stored in memory. These workloads can be spread across multiple containers, nodes, and namespaces. Backing up just Etcd does not provide an effective way to restore these workloads in the event of a failure.
- IaaS providers like AWS advertise a **non-zero** annual failure rate.

- Volume Snapshots for on-premises solutions could look appealing, but these volume snapshots are often not resistant to hardware failure and even worse, the deletion of a volume usually leads to an automatic deletion of all related snapshots.
- It's true we can automate the deployment of application by using GitOps (or any other Infrastructure as Code solution) or using CI/CD pipelines (used mainly for version control), which could of course streamline the application recovery process after a disaster. But there is a critical point we are missing here: **the Data!** Any of these automation methods that we can use to deploy back the applications after a disaster, will bring back only the Kubernetes objects and their configurations, but what about the persistent data? Remember, stateful applications can store data using Persistent Volumes (PV) or databases (relational or NoSQL databases). All these persistent data isn't captured when we use any of the automation solutions already mentioned, so they can't be considered a Data Protection solution by themselves, though they could be part of the Data Protection and Disaster Recovery strategy.

## Choosing the right solution

When we start looking for a Data Protection and Disaster Recovery solution for our Kubernetes platform, it's important to understand the fundamental differences between Kubernetes and any other compute platform that has come before.

In the past, applications were usually deployed on Virtual Machines (VM) or physical servers creating silos. Even with distributed applications we were able to map specific applications components to a specific VM or server. Therefore, traditional backup solutions are usually designed to capture the entire VM or Server when we need to protect a specific application. In other words, these backups solutions were infrastructure-centric solutions.

The main challenge to protect Kubernetes workloads is that there is no mapping of applications to servers or VMs. Now, a single application could be composed by hundreds of Kubernetes resources (deployments, services, PV, secrets, ConfigMaps, etc.), all of them distributed among the available Kubernetes nodes.

In addition, Kubernetes uses its own placement policy to distribute applications components across all servers for fault tolerance and performance, which of course makes kind of impossible to map an application to a specific server.

We also have to consider the dynamic nature of Kubernetes and cloud-native applications:

- New applications could be deployed every single day whether manually or, most likely, using automation solutions like GitOps or CI/CD pipelines.



- Application management can involve periodic rolling upgrades, and new application components can be added or removed at any time.
- Pods can be dynamically rescheduled or scaled on different nodes for better load balancing.
- IP address management it's also quite dynamic and handled directly by the Kubernetes platform, where every Pod will have an IP address dynamically assigned.

And of course, we won't have a single application running on Kubernetes. In a Kubernetes cluster we could find multiple components of multiple applications running altogether, which means the solution should be able to understand and protect thousands or even millions of components. All of this of course makes it even more challenging to have a proper Data Protection and Disaster Recovery solution for Kubernetes.

Therefore, the Data Protection and Disaster Recovery solution should be:

- **Compatibility with Kubernetes architecture:** Kubernetes has a specific architecture and set of components, including etcd, kubelet, and the API server. A data protection solution for Kubernetes should be compatible with this architecture and should be designed to work seamlessly with all these components.
- **Application-centric**, being able to understand Kubernetes constructs, and not infrastructure-centric like they were in the past, in order to understand the relationship between the application components, and between the components and their data.
- This solution should also be able to **scale-up and down** and/or scale-out and scale-in alongside with the Kubernetes platform and applications to provide the proper performance for Data Protection operations.
- In case of using solutions like CI/CD pipelines to redeploy the applications, the solution should be smart enough to keep track the relationship between stateful applications and their data (PV, databases, etc), and therefore being able to perform a **data-only restore**, whether restoring the PVs used by the application or restoring data to the database used by it, as required.
- This solution should also integrate with the **Kubernetes API** to manage backup and recovery operations. This includes the ability to manage PVs, pods, and namespaces, as well as the ability to schedule backups and restores.
- A good data protection solution for Kubernetes should be **automated**, making it easy to backup and recover data quickly and efficiently. This includes automated backups of Persistent Volumes (PVs), as well as automatic recovery in the event of a failure.
- Finally, a proper Kubernetes Data Protection and Disaster Recovery solution should be able to offer multiple storage options to store the backup data. **Immutability** it's also a paramount need nowadays to protect our data against threats like ransomware attacks.



# Kubernetes Disaster Recovery Strategy

It's always important to have a Data Protection strategy in place, alongside with a proper Disaster Recovery Plan. This of course applies to cloud-native applications running on Kubernetes.

So, what components, solutions and best practices should or could be part of this Disaster Recovery strategy? As minimum, a proper Disaster Recovery Strategy should include:

- A Kubernetes native backup solution to protect all the applications and their persistent data.
- Automation solutions to speed up the recovery process (RTO) and reduce human errors.
- A highly available and secure location for backup data, including immutability features.
- Design a Disaster Recovery strategy that best suit the protection requirements and available infrastructure.
- Document the entire Disaster Recovery plan.

In this section we will discuss the components to be included in a Disaster Recovery strategy using Kasten by Veeam.

## Kubernetes Native Backup Solution

As we already mentioned, we need a Kubernetes native Data Protection solution, capable of protecting cloud-native applications running in these highly dynamic Kubernetes platforms. Veeam's Kasten K10 is a leading Data Protection solution in the industry specifically designed for Kubernetes platforms, making it an ideal choice for effectively protecting cloud-native applications in these highly dynamic environments.

Kasten K10 has been designed and built to run as a **cloud-native application** running in the same Kubernetes cluster we want to protect. By using this approach, Kasten K10 can run natively in the Kubernetes cluster and discover all applications running on it with all their dependencies, in addition to be capable of scaling-up and down and/or scale-out and scale-in alongside the Kubernetes cluster.

By using a Kubernetes Native Backup solution like Kasten K10, we will be able to protect our cloud-native applications, including all their persistent data. In addition to the Backup and Restore operations, Kasten K10 can also provide **disaster recovery and migration** capabilities to Kubernetes applications.



## CI/CD for automation and rapid rollbacks

Continuous Integration and Delivery solutions to automate application's deployment could play an important role in our Data Protection strategy, as they provide a method to re-deploy an application, with all its components, whenever it's necessary. This kind of solutions could also leverage their version control features to "restore" the application with the desired version or configuration.

For example, let's say we have a Kubernetes cluster with GitOps in place for automation. GitOps can use a software agent to alert when there are differences between the Git repository and what is actually running on the Kubernetes cluster. Then we observe a situation where someone made a manual change to the application directly on the cluster, without leveraging the version control solution. We could leverage GitOps to get the declarative state of the Kubernetes cluster, and then just running a Git revert operation, effectively rolling back the application to its original state overwriting the changes done manually.

Still, remember this kind of solutions by themselves aren't enough, as they aren't able to protect the persistent data stored by the applications.

## Where to store my backups?

To design a proper Disaster Recovery strategy, it's necessary to choose the proper location for our backups. This is quite important as this data must be available in case of a disaster so we can restore the application and all its data in an alternate location. A proper storage solution for backup data should be designed considering:

- Redundancy and High Availability
- Proper performance to meet the required RPOs and RTOs
- Immutability to protect data against threats like ransomware.

**IMPORTANT:** The 3-2-1 backup rule is still relevant when it comes to data protection for Kubernetes. The 3-2-1 backup rule states that there should be at least three copies of data, stored on two different media, with at least one copy stored off-site. This rule helps ensure that data can be recovered in the event of data loss or other issues.

Kasten K10 allows us to export the backup data to Object Storage like AWS S3, Azure Blob or any S3 compatible bucket, including on-premises solutions. Regardless the storage solution chosen to keep the backup data, it's important to protect our backups in case a disaster, whether an AWS (or another vendor) region goes down, or our Datacenter is down.

For instance, we could send our backups to an AWS S3 bucket in a specific region, and then set a replication policy in the AWS S3 bucket to replicate it to a different region (this is, of course, supported by Kasten), which will allow us to restore the application data even when an entire AWS region is unavailable.

Similar configurations could be achieved with on-premises solutions like Scality RING or Cloudian HyperStore, where the data stored in the Object Storage is replicated and available in multiple data centers.

**NOTE:** By default, K10 encryption is enabled for data and metadata stored in an object store or NFS file store, by using AES-256-GCM encryption algorithm. Encryption cannot be disabled, but you can configure the method of encryption to be used in your cluster. More info in [Kasten documentation](#).

## Approaches for Disaster Recovery strategy with Kasten

By using Kasten as a Data Protection and Disaster Recovery solution, we have multiple options to restore our applications such as:

- Restoring single application's components/artifacts

- Restoring an entire application
- Restoring multiple applications
- Restoring the entire cluster and its applications in a new cluster for Disaster Recovery purposes.

In this document we are going to focus in how to recover the entire Kubernetes cluster after a disaster. For this, we have 2 options or Disaster Recovery strategies:

- **Option 1:** Restoring all the applications in a new Kubernetes cluster using the Kasten Import/Export features.
- **Option 2:** Restore the entire Kubernetes configuration and applications using Kasten DR features.

In the next sections we will describe both Disaster Recovery strategies, and how implement them with Kasten K10 step by step. Nevertheless, in a real-world scenario **time is of the essence**, and when a disaster strikes, we want to recover all our applications and data as soon as possible, **avoiding running manual tasks** whenever is possible.

In the Disaster Recovery context, **automation is a paramount**, and we should always include solutions and/or tools that allows to automate and streamline the recovery process. Kasten K10 allows automation using third-party solutions that can leverage the Kasten K10 API. Some **automation strategies will be described** in the last section of this document.

## Option 1: Disaster Recovery using Kasten Export/Import features.

In this section we will focus on how to implement a Disaster Recovery strategy to restore all Kubernetes applications into another Kubernetes cluster in a different location, using the native **Kasten Export/Import** capabilities. This strategy is meant for the following scenarios:

- **Hybrid environments:** We have applications running in an on-premises Kubernetes cluster like RedHat OpenShift, Suse Rancher or VMware Tanzu, and want to restore them into another Kubernetes cluster provided by a Public Cloud like AWS EKS, Azure AKS or GCP GKE for Disaster Recovery purposes.
- **Cross-cluster (on-premises to on-premises):** We have applications running on-premises, using a Kubernetes cluster like RedHat OpenShift, Suse Rancher or VMware Tanzu, and we want to restore them into another Kubernetes cluster running in the same site or in a remote location.
- **Cross-Cloud:** We have applications running in a Kubernetes Cluster provided by a Public Cloud like AWS EKS, and we want to restore them into another Kubernetes cluster provided by a different Public Cloud, like Azure AKS.
- **Cross-Region/Cross-Account:** We have applications running in a Kubernetes Cluster provided by a Public Cloud provider like AWS EKS, and we want to restore them into another Kubernetes cluster provided by the same Cloud Provider, but running in another Region or Account/Subscription.

For this scenario we will be using an on-premises RedHat OpenShift deployment as a Production Kubernetes Cluster. For the DR Kubernetes Cluster we will be using Red Hat OpenShift on Amazon Web Service (ROSA).

### Cluster Information

K10 Version	5.5.2
K10 Namespace	kasten-io
Kubernetes Version	v1.24.6+5658434
Kubernetes Release Type	OpenShift

### Cluster Information

K10 Version	5.5.2
K10 Namespace	kasten-io
Kubernetes Version	v1.24.6+5658434
Kubernetes Release Type	AWS, OpenShift

Both, Production and DR Kubernetes clusters will be using AWS S3 buckets as Location Profiles.

## Stage 1: Protecting our applications

The high-level workflow for data protection in Kasten K10 involves taking a **snapshot** of your application data and then **exporting** that snapshot to an external location for added data protection.

- First, you create a backup of your data as defined by your backup policy (snapshot).
- Then, you can choose to export the snapshot to another location, such as a secondary data center or cloud storage service, to provide an additional layer of protection for your data. For this we need to configure the proper Location Profiles in Kasten as explained next.

### Configuring Location Profiles

Before we can protect our applications using Kasten Policies, we must set one or more Location Profiles. Location Profiles are basically repositories where Kasten can export the application data. For Location Profiles we can use:

- Object Storage like AWS S3, Google Cloud Storage, Azure blob, or any other S3-compliant object store.
- NFS file storage.

**TIP:** When using this DR strategy, it's very important that the DR Kubernetes cluster has access to the Object Storage buckets or NFS shares that contains the actual backup data (in the image above, the Location Profile is an AWS S3 Bucket call **pcerda-k10**). This can be accomplished in two different ways:

- **Object Storage buckets or NFS shares are accessible from source (Production) and target (DR) Kubernetes cluster.** For instance, we can use AWS S3 buckets, so even if we lose the entire Production Kubernetes cluster, the S3 bucket is still available and visible for the DR Kubernetes cluster.
- **Backup data is replicated to another Object Storage bucket or NFS share.** For instance, we can enable replication for an S3 bucket (AWS S3 or S3 compatible), so we can have a second S3 bucket, in a different location, with the replicated backup data. While backups of Production Kubernetes cluster are sent to the S3 bucket, the DR Kubernetes cluster can access to backup data from the replicated bucket (alternate location).

## Location Profiles

Create profiles that **define credentials and locations** needed to move data in and out of the cluster. You'll select from these profiles when creating policies or exporting a restore point.

[New Profile](#)

LOCATION PROFILE **pcerda-k10**

CLOUD PROVIDER	REGION	BUCKET NAME
AWS S3	EU (Paris) • eu-west-3	pcerda-k10
STATUS <b>Valid</b>		

LOCATION PROFILE **pcerda-k10immutable**

CLOUD PROVIDER	REGION	BUCKET NAME	OBJECT IMMUTABILITY	PROTECTION PERIOD
AWS S3	EU (Paris) • eu-west-3	pcerda-k10immutable	Enabled	1 day
STATUS <b>Valid</b>				

For this example, in Production Cluster we are using an AWS S3 Bucket called **pcerda-k10** (image above) to backup/export the application data using Kasten policies. This bucket has a replication rule in AWS S3 to replicate all the data from this bucket to another bucket called **pcerda-k10-dr**, which can be used later to import the application data.

Amazon S3 > Buckets > pcerda-k10

**pcerda-k10**

Objects | Properties | Permissions | Metrics | **Management** | Access Points

---

**Lifecycle rules (0)**

Use lifecycle rules to define actions you want Amazon S3 to take during an object's lifetime such as transitioning objects to another storage class, archiving them, or deleting them after a specified period of time. [Learn more](#)

[Refresh](#) [View details](#) [Edit](#) [Delete](#) [Actions](#) [Create lifecycle rule](#)

Lifecycle rule name	Status	Scope	Current version actions	Noncurrent versions actions	Expired object delete markers
No lifecycle rules					
There are no lifecycle rules for this bucket.					
<a href="#">Create lifecycle rule</a>					

---

**Replication rules (1)**

Use replication rules to define options you want Amazon S3 to apply during replication such as server-side encryption, replica ownership, transitioning replicas to another storage class, and more. [Learn more](#)

[Refresh](#) [View details](#) [Edit rule](#) [Delete](#) [Actions](#) [Create replication rule](#)

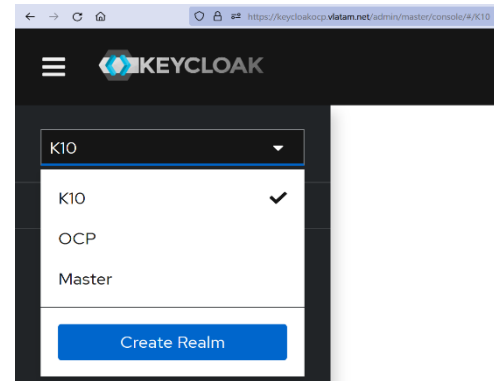
Replication rule name	Status	Destination bucket	Destination Region	Priority	Scope	Storage class	Replica owner	Replication Time Control	KMS-encrypted objects	Replica modification sync
<input type="radio"/> <b>KastenK10</b>	Enabled	s3://pcerda-k10-dr	EU (Paris) eu-west-3	0	Entire bucket	Same as source	Same as source	Enabled	Do not replicate	Enabled

[View replication configuration](#)

## Creating Kasten K10 Export policies

Once Location Profiles are configured in Kasten, to provide Disaster Recovery capabilities, the next step is of course to backup our applications in the source Kubernetes cluster (further referred as Production cluster). In this case we will use Kasten K10 to protect several applications running in the Production Cluster. We can create a Kasten Policy to protect every application or create a Kasten Policy to protect multiple applications together.

In the next example, we will create a Kasten Policy to backup one application called “Keycloak”. [Keycloak](#) is an open-source Identity and Access Management that provides features like Single-Sign On, Identity Brokering and User Federation that can be used by multiple applications. This application includes the use of MySQL to keep all Keycloak configuration and a PVC to store the MySQL database.



```
rke@K8sJumpbox:/$ oc get pods -n keycloak
NAME                                READY   STATUS    RESTARTS   AGE
kc-mysql-c7dd76b56-8xknz           1/1     Running   0           5d23h
keycloak-1-4852d                   1/1     Running   0           5d23h
rke@K8sJumpbox:/$ oc get pvc -n keycloak
NAME            STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
kc-pv-claim     Bound    pvc-6f51416c-e37f-4736-abbf-13c5f369f744  20Gi       RWO             thin-csi       5d23h
rke@K8sJumpbox:/$
```

The application is accessible by using OpenShift routes as we can see in the picture bellow. There are 2 routes, the first using the default domain set in OpenShift during OpenShift deployment, and the second is a route using a custom FQDN with a custom SSL certificate.

Project: keycloak ▼

**Routes** Create Route

Filter ▼ Name ▼ Search by name... /

Name	Status	Location	Service
keycloak	Accepted	https://keycloak-keycloak.apps.ocpdemo.homelab.local	keycloak
keycloak-ext	Accepted	https://keycloak.vlatam.net	keycloak

**NOTE:** The same procedure will be followed to protect all applications in Production Cluster.

When we create a new Policy, first we must specify the Policy **name**, and **Snapshot** for the action. Next can set the **backup frequency**, for instance we can set this Policy to run every hour.

In this scenario we need to enable the **Enable Backup via Snapshot Exports** option to send the backup to an Object Storage or NFS file storage. Then you need to enable the **Export Snapshot Data** option (default) in order to export both the application data and metadata. Without it, the application data will not be exported, and the subsequent import will fail.

When the **Export Snapshot Data** option is enabled, we must specify a **Location Profile** (repository) where the exported data and metadata will be stored. As mentioned before, we have some options for Location Profile:

- Object Storage like AWS S3, Google Cloud Storage, Azure blob, or any other S3-compliant object store.
- NFS file storage.

**New Policy**

**Name**  
The display name for this policy  
keycloak-backup

**Comments**

**Action**  
The action that should be taken when this policy is executed  
☒ Snapshot ☐ Import

☐ **Use a Preset**  
Choose a pre-configured group of schedule and export settings

**Backup Frequency**  
☒ Hourly ☐ Daily ☐ Weekly  
☐ Monthly ☐ Yearly ☐ On Demand

☐ **Advanced Frequency Options**  
☐ **Backup Window**  
Snapshot at :00 each hour

**Note:** Times are stored in UTC, which does not change with Daylight Savings Time.

**Snapshot Retention**  
Customize the snapshot retention schedule if needed. [Set to Zeros](#)

VMware recommends 2 to 3 snapshots. A maximum of 32 are supported.

3	hourly snapshots	0	daily snapshots
0	weekly snapshots	0	monthly snapshots
0	yearly snapshots		

## Getting the Import Data

As a last step, after the Kasten backup policy is already created, we need to take note of the **Import Data**. This code will be required when we create the Import Policy in the target cluster (further referred as DR cluster) and can be obtained from the policy by clicking **Show import details**, which will result in a code like the one displayed in the next image.

**Importing Data**

The encoded text below contains import data needed by the receiving cluster. You'll be asked to paste this text when you create an import policy on the receiving cluster.

Visit the Policies Page at any time to see this information.

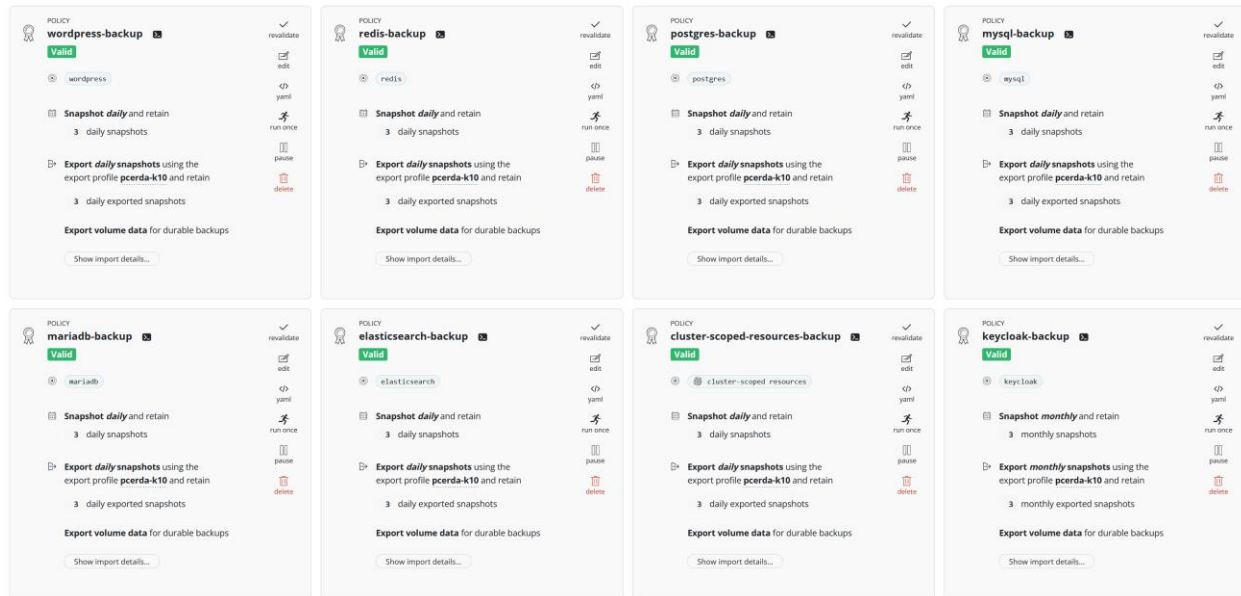
[Copy to Clipboard](#)

```
hTzAPpseemE4oW1InjJzodc1ekE1DXSTjwaCqB4qP1SPBoTvkYfcP8Heu+su+H/Pi3ozKwNGU+u7
/xH2+ZmN2andYEU0gA1Tvs4k
/w1Rk555hTbZmuE8E00o3MTLL99290XSzHSJrp1N8H4zbaJd1oYyeZS3AdosMuGSFyo31lc2DP6y
/FGDGr21wrJawY+hp506wh03KY8Fg3Rbzodwz3NveEzowwBUQGG3Vw2SH
/uvGgzhPP3srqpnF8zkzq3m9vY+1KYoez/su6113BmCEv9m4gKAdV
/9u4u0PR2Nzg1N820wzCBd13p8os+wee7Jum8gRQGA
/g8hTvxz2x17r18akupdn1b7px1+QQFku8G8QndzwTVvL1F1DvQLCg198qyBF1wCy3Hccpw18N
/8P9hag
```

[Dismiss](#)

**IMPORTANT:** For the Disaster Recovery Plan we need to keep a copy of the Import Data for every single application we want to restore in a different Kubernetes cluster after a disaster.

For our scenario, in the Production Cluster we have several stateful applications, running different types of databases and applications. We have created a Kasten Policy for each one of these applications:





## Stage 2: Importing the applications in the target cluster

### Configuring Location Profiles

Just like we mentioned in Stage 1, we need to set one or more Location Profiles also in DR Cluster. These Location Profiles will be used to Import the data from Production Cluster, but also can be used later to protect the application running in DR Cluster using Kasten policies.

In this example we have two options to be used as a Location Profile in DR Cluster to import the application's data:

1. The AWS S3 bucket used in Stage 1 to store the exported backup data using a Kasten Policy (if it's still available). In this case, the bucket name is **pcerda-k10**.
2. As we have an AWS S3 replication rule in place, replicating all the data in **pcerda-k10** bucket to another bucket called **pcerda-k10-dr**, we can use the later to import the data using the "Alternate Location" option. This option is useful when the Location Profile used in Production Cluster isn't available anymore after a disaster occurs.

### Location Profiles

Create profiles that **define credentials and locations** needed to move data in and out of the cluster. You'll select from these profiles when creating policies or exporting a restore point.

[New Profile](#)

LOCATION PROFILE

pcerda-k10

revalidate

yml

edit

delete

CLOUD PROVIDER

AWS S3

REGION

EU (Paris) • eu-west-3

BUCKET NAME

pcerda-k10

STATUS

Valid

LOCATION PROFILE

pcerda-k10-dr

revalidate

yml

edit

delete

CLOUD PROVIDER

AWS S3

REGION

EU (Paris) • eu-west-3

BUCKET NAME

pcerda-k10-dr

STATUS

Valid

### Creating Kasten K10 Import policies

Once we have all our applications protected with Kasten policies in the Production Cluster, and the Location Profiles properly set in DR Cluster, it's time to set the Kasten Import Policies in the DR Cluster. Creating an Import Policy is very similar to the process of creating a Policy to protect the applications as we described in Stage 1, so in the Policy page we select **Create New Policy**.

To import the protected applications in the DR cluster, we select **Import** for the action instead of Snapshot at the Policy creation. Next you can set a frequency for the import task, which will control how often the Policy will check the Object Storage or NFS File Storage location for new data to import. In case a frequency is not set, the Policy must be run manually on-demand.

**NOTE:** The frequency for the Import task doesn't need to match the frequency set in the protection Policy. For instance, the protection Policy in the Production Cluster could be set to run every hour, as showed in Stage 1, and then the Import Policy in the DR Cluster could be set to run every day. The frequency for both policies mainly depend on customer's requirements for Data Protection and Disaster Recovery, and the required RPO.

One of the options we can enable when creating an Import Policy is **Restore After Import**. This option, as the name implies, will restore the entire application, including all its data, and bring the application up in this cluster after the metadata import is complete. By default, this option isn't enabled, so an Import Policy will import just the application metadata to Kasten, but not the data itself and the application won't be running in this cluster.

If we decide to enable the **Restore After Import** option, every time the Import policy runs it will restore the entire application and its data to the DR Kubernetes Cluster. This means, the application will be running simultaneously on both, Production and DR Kubernetes Cluster. In this scenario, we will basically have an **Active-Active** configuration, where the application would be available in both sides, the Production and DR Kubernetes cluster. So, the decision of whether enable the **Restore After Import** option or not, depends on how the applications will behave after the restore process, and if the applications are designed to be accessible from multiple locations.

**IMPORTANT:** According to the official Kasten documentation, care should be taken when auto-restoring the application during import (Restore After Import option enabled). In particular, ensure that the newly restored application **does not conflict** with the application running in the source cluster. Examples of potential conflicts include accidental credential reuse, access to and use of external services, and services conflicting for exclusive ownership of shared resources.

**New Policy**

**Name**  
The display name for this policy  
keycloak-import

**Comments**

**Action**  
The action that should be taken when this policy is executed  
☐ Snapshot ☒ Import

☒ **Restore After Import**  
Automatically restore after importing

**Import Frequency**  
☐ Hourly ☒ Daily ☐ Weekly  
☐ Monthly ☐ Yearly ☐ On Demand

**New Policy**

☒ **Restore After Import**  
Automatically restore after importing

☐ **Data-Only Restore**  
Restore only the volume data and exclude other artifacts such as config files.

☐ **Don't wait for workloads to be ready**  
Specifies whether the restore action should skip waiting for all workloads (Deployments, StatefulSets or DeploymentConfigs) to be ready before completing.

☐ **Restore cluster-scoped resources**  
If the restore point contains cluster-scoped (non-namespaced) resources, they will **not be restored unless you select this option**. This helps prevent against unintended overwriting of this cluster's resources.

☐ **Apply transforms to restored resources**  
On restore, change the contents of spec resources. This may be useful when migrating between environments. For example, you can change storage classes or edit container image names.

**Select Application Resources**  
Optionally create filters to include/exclude specified application resources.  
☒ All Resources ☐ Filter Resources

As we continue creating the Import policy, the next step is pasting the Import Data we got in the Stage 1 in **Config Data for Import**, to allow Kasten K10 to create the data export/import relationship for this application across the clusters.

Finally, a location profile must be selected, which contains the backup data created by Kasten in Production Cluster (the DR Cluster usually only needs read and list permissions on all the data in the Location Profile).

When selecting a Location Profile, the list of location profiles will show:

- **A Matching Profile:** This is the original Export Location (If this location is visible to DR Cluster), which should contain the exported restore points.
- **A list of "Other Profiles":** Selecting a profile from the "Other Profiles" section can be useful if, for example, a restore point has been replicated or moved from its original export location.

Now we can save the Import Policy to complete the process.

**New Policy**

**Config Data for Import**  
Paste the text that was presented to you when the restore point was exported from the source cluster.

kVv9fp191u1m28k1y7p1e3up5e9Rv0RgP15C1B2MBNA1yvAKC5vmmuKfHPInShzouR3kBy1k5NP4L8mG1n1ZF1U1oC1OBRA9H4Dng

**Profile for Import**  
Select the profile that defines the location for importing data.

pcerda-k10-dr

+ Create new profile...

1 MATCHING PROFILE

pcerda-k10  
S3, eu-west-3, "pcerda-k10"

2 OTHER PROFILES

pcerda-k10-dr  
S3, eu-west-3, "pcerda-k10-dr"

pcerda-k10immutable  
S3, eu-west-3, "pcerda-k10immutable"

while restoring snapshots

## Stage 3: Restoring Applications after a Disaster

The Import Policy will run according to the schedule set in the policy or manually if required. If new data is detected in the Location Profile, its metadata will be imported into the cluster, automatically associated with the application stack already running, and be made available as a restore point.

Now, let's discuss about how to restore our applications after importing them with Kasten.

"Restore after Import" option NOT selected

Please remember that unless **Restore after Import** is selected, only metadata is brought into the cluster. If the data volumes reside in an object store or NFS file store, they will not be converted into native volumes until a restore operation is initiated.

In this case, the Import Policy was created to import just the application metadata, without actually restoring the application data. If we run this policy, we can see that the application and the Restore Point catalog has been successfully imported, but the application itself hasn't been restored, as that option wasn't enabled in the Import Policy.

Actions <span>1</span>					Filter Actions
COMPLETED	PHASES	PROTECTED OBJECT	ARTIFACTS	START	
<b>Import</b> scheduled-pr782	<ul style="list-style-type: none"> <li>Importing RestorePoint</li> <li>All phases completed successfully.</li> </ul>	none	No action was taken because the import artifacts already existed	Today, 1:16pm	
		ORIGINATING POLICY		DURATION	
		keycloak-import		9 secs	

So, what happens when a disaster occurs, and we need to recover all our applications in the DR cluster? Well, in that case we need to restore the applications in the DR cluster. This restore process can be done manually for every application, or it could be automated by using the [Kasten API](#) and third-party automation solutions like Ansible, Chef or ArgoCD.

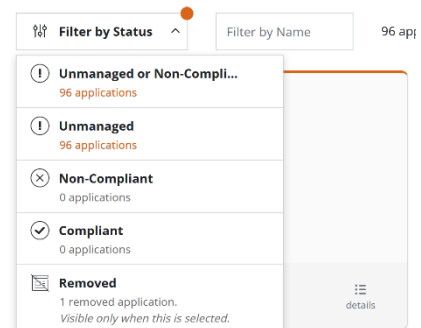
Here we are going to describe how to manually restore the applications in the DR cluster using the native Kasten features. Later in this document we will discuss about automation strategies for Disaster Recovery.

First, go to the **Applications** section in the Kasten UI, select **Removed** under the **Filter by status** drop-down menu. Then click restore under the application we want to restore and select a restore point to recover from.

## Applications

View details or perform actions on applications.

Once we select the restore point, we want to use to restore the application, then we need to specify the Namespace where the application will be restored. As we are restoring the applications in a new Kubernetes cluster, the original Namespaces don't exist. In this case, we can create a new Namespace, during the restore process, with the desired name as we can see in the image below:



Restore Point

Imported Restore Point

This restore point was imported from another cluster. It will require additional time to download, import, and restore from.

ORIGINATING POLICY

keycloak-io/keycloak-import

IMPORT PROFILE

EXPORT TYPE

pcerda-k10

Complete snapshot data was exported for portable backups.

KUBECTL COMMAND

\$ oc get --raw /apis/apps.kio.kasten.io/v1alpha1/restorepointcontents/keycloak-scheduled-dl -o copy

Application Name

Select a namespace to restore into. The contents of the selected namespace will be overwritten with the restored application.

Select a Namespace...

Create a New Namespace

New Namespace

keycloakrestored

Create

Cancel

**TIP:** If we try to import and restore in a different Kubernetes cluster a **stateful** Application using Persistent Volumes, we must check the **Storage Classes** available on both, Production and DR clusters, in order to allow the Import Policy to complete successfully.

In our scenario, we have an on-premises OpenShift cluster as the Production cluster, and we have two Storage Classes available as we can see in the next image:

```
rke@K8sJumpbox:~$ oc get storageclasses
NAME                                PROVISIONER                                RECLAIMPOLICY
thin                                kubernetes.io/vsphere-volume              Delete
thin-csi (default)                 csi.vsphere.vmware.com                   Delete
rke@K8sJumpbox:~$
```

For our DR Cluster we are using RedHat OpenShift on AWS, and we have four Storage Classes available as we can see in the next image:

```
rke@K8sJumpbox:~$ oc get storageclasses
NAME                                PROVISIONER                                RECLAIMPOLICY
gp2                                kubernetes.io/aws-ebs                     Delete
gp2-csi                           ebs.csi.aws.com                          Delete
gp3 (default)                     ebs.csi.aws.com                          Delete
gp3-csi                           ebs.csi.aws.com                          Delete
rke@K8sJumpbox:~$
```

After importing it, if we try to restore the Keycloak application as it is, the restore process will fail with an error message (see the image) reporting that the Storage Class required by the Persistent Volume Claim (PVC) is not available in the Kubernetes Cluster.

The application was using a Storage Class with name **thin-csi** in the Production Cluster (RedHat OpenShift), while that Storage Class isn't available in the DR Cluster (RedHat OpenShift on AWS). In order to fix this issue, and allow the import process to complete successfully, we need to set a Transform in the Import policy.

A Transforms enable modifications to Kubernetes resources during a restore/import task. In this case we need a Transform to allow the Imported application to use a different Storage Class in the DR Cluster, specifically the Storage Class called **gp3** in AWS EKS.

**Phases (1)**

Restoring Application Components

STARTED

Today, 2:29pm

ATTEMPTS

3

ERROR MESSAGES

Job failed to be executed ^

Failed to validate Storage Class for PVC artifact

Provisioner not registered in the cluster. If this provisioner is valid - Create the StorageClass and retry operation

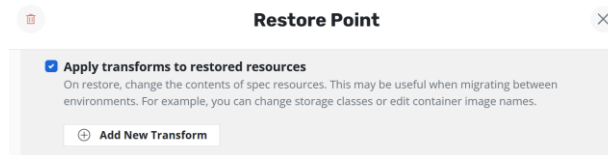
storageclasses.storage.k8s.io "thin-csi" not found

Job failed to be executed ✓

Job failed to be executed ✓

[Show Details](#)

So, during the Restore process we must enable the **Apply transforms to restored resources** option and click in **Add New Transform**.



In the **New Transform** window, we must specify a name for the Transform and then the resource that will be transformed during the Restore process. In this case, we need to transform the application's PVC to use a different Storage Class in the DR Cluster.

We click in **Resource** and set *persistentvolumeclaim* as the resource to be transformed. Then, in **Operations**, we create a **Replace** operations to change the PVC's Storage Class to one of the Storage Classes available in DR Cluster (**gp3**).

**TIP:** You can click in **Use an Example** to get a sample template to create a Storage Class transform more easily.

**Display Name** Reset Form Use an Example

changeStorageClass

**Resources**  
Specify which resource artifacts to apply this transform to.

Group	Version
<input checked="" type="checkbox"/> Resource	<input type="checkbox"/> Name

persistentvolumeclaims

> Apply to resources where resource type is persistentvolumeclaims

**Operations**  
A transform can have one or more operations. For example, you might use a test operation to test that an element in the resource exists before using a replace operation.

New Operation Test All Operations

> **Replace**

path	value
/spec/storageClassName	gp3

Next, we can choose whether we want to restore the entire application or just specific Volumes or Artifacts. For instance, if you are using a CI/CD solution like ArgoCD to redeploy the entire application stack in the target Kubernetes cluster, then you can use Kasten to restore just the Persistent Volumes (PVs). Remember using CI/CD solutions can't provide protection for persistent data.

Restore Point

Snapshot (1)

Restoring 1 of 1 volumes

Deselect Volume Snapshots

☒

KUBERNETES VOLUME PVC

kc-pv-claim

STORAGE CLASS NAME

thin-csi

SIZE

190.7 MiB

Spec (24)

Restoring 24 of 24 spec artifacts

Select Artifacts by Type

Deselect All Spec Artifacts

☒

TYPE

configmaps

VERSION

v1

NAMESPACE

keycloak

NAME

kube-root-ca.crt

☒

TYPE

configmaps

VERSION

v1

NAMESPACE

keycloak

NAME

openshift-service-ca.crt

☒

TYPE

deploymentconfigs

VERSION

v1

GROUP

apps.openshift.io

NAMESPACE

keycloak

NAME

keycloak

In case the Location Profile where the application data was sent to by using a Kasten Policy is not available, we can use an Alternate Location Profile to restore the application from. In our scenario remember we have an AWS S3 bucket (pcerda-k10) with a replication policy to replicate all data to another AWS S3 bucket (pcerda-k10-dr). This bucket with the replicated data can be used to restore the applications by clicking in **Alternate Location Profile** option, and then choosing the proper Location Profile to restore the data from.

Restore Point

☒ Alternate Location Profile

By default, an external restore point will be restored using the same location profile it was exported with (pcerda-k10). Select a different profile if, for example, the restore point was copied to another cloud location.

pcerda-k10-dr

Finally, we click in “Restore” to start the restore process for the selected application. As we can see in the dashboard, the application was successfully restored:

Actions (3)

Filter

Page 1

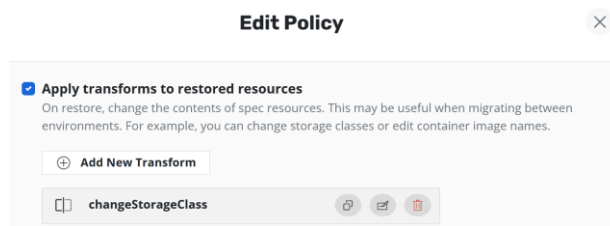
COMPLETED	PHASES	TARGET NAMESPACE	ARTIFACTS	START	...
<b>Restore</b>	<input checked="" type="checkbox"/> Restoring Application Components <input checked="" type="checkbox"/> All phases completed successfully.	<b>keycloakrestored</b>	none	Today, 1:39pm	
keycloakrestore...		RESTORE POINT		DURATION	
		<b>keycloak-scheduled-6btq2dtwg2</b>		1 min, 15 secs	

“Restore after Import” option selected

Another option we have for restoring the applications, is to enable the **“Restore after import”** option in the Import Policy as mentioned previously in **Stage 2**. In this case, the applications will be restored automatically in the DR Cluster every time the Import Policy runs. No additional step is required in order to recover our applications.

Please remember that, as mentioned before, as we are recovering our applications in a different Kubernetes cluster, in a different location, some transformations could be required. For instance, one of the most common transformations is the use of a different Storage Class in the target Kubernetes cluster.

In our case, the on-prem Production OpenShift Cluster and the DR OpenShift Cluster running in AWS are both using different Storage Classes. Thus, during the Import Policy creation, or by editing the policy afterwards, we must enable the **Apply transforms to restored resources** option and then add a new Transform to use the proper Storage Class during the application restore process, as we explained before.



Now we can run the Import Policy manually or let Kasten to run it automatically according to the schedule previously set in the policy. As we can see in the picture below, Kasten has imported the application from the Location Profile, including all its metadata and the Restore Point catalog, and then it has restored the application with all its artifacts and data:

The screenshot shows the Kasten dashboard. At the top, there's a header with 'KASTEN by Veeam', 'Docs', 'Settings', and a user profile 'k10-admin'. Below the header, the 'Dashboard' section shows a 'COMPLETED SUCCESSFULLY' status for the 'keycloak-import' policy. It includes a start time of 'Today, 12:32pm', an end time of 'Today, 12:34pm', and a duration of '2 mins, 22 secs'. There's also a 'Show Details' link. Below this, the 'Actions' section shows two completed actions: 'Restore' and 'Import'. The 'Restore' action shows 'Restoring Application Components' and 'All phases completed successfully.' The 'Import' action shows 'Importing RestorePoint' and 'All phases completed successfully.' Both actions show their start times and durations. At the bottom, there's a table with columns for 'COMPLETED', 'PHASES', 'TARGET NAMESPACE', 'ARTIFACTS', and 'START'. The 'Restore' action is listed with 'keycloak' as the target namespace and 'none' as artifacts. The 'Import' action is listed with 'kanister' and 'spec' as artifacts.

COMPLETED	PHASES	TARGET NAMESPACE	ARTIFACTS	START
Restore scheduled-nghbd	Restoring Application Components All phases completed successfully.	keycloak	none	Today, 12:32pm
Import scheduled-dn92g	Importing RestorePoint All phases completed successfully.	keycloak-import	1 kanister 25 spec	Today, 12:32pm



## Stage 4: Make the applications accessible

We have already described the process to protect the applications running in our Kubernetes cluster, and then how to recover them in an alternate location. There is still a missing step in order to make sure the applications can be actually used.

Some of the applications running in a Kubernetes cluster are designed to be accessible by users, like web applications. In the same way, other applications could be designed to be accessible for other applications running out of the Kubernetes clusters. Usually, all these applications are accessible by using external load balancers, Ingresses or Routes (OpenShift).









When we recover all our applications in a different cluster, in a different location, it's very likely that we need to make some additional changes to make sure the applications are available after being recovered.

In our previous example, the Keycloak application, we have 2 OpenShift routes that can be used to access the Keycloak portal in the Production cluster. The first Route uses the default domain set during the OpenShift cluster deployment. The second Route uses a custom domain with an SSL certificate to secure the connection.

Project: keycloak ▾

### Routes

Filter ▾ Name ▾ Search by name... /









Name ▴	Status	Location ▴	Service ▴
 keycloak	✓ Accepted	<a href="https://keycloak-keycloak.apps.ocpdemo.homelab.local">https://keycloak-keycloak.apps.ocpdemo.homelab.local</a>  	 keycloak
 keycloak-ext	✓ Accepted	<a href="https://keycloakocp.vlatam.net">https://keycloakocp.vlatam.net</a>  	 keycloak

After we restore the application in the DR Cluster (OpenShift on AWS), we can see both Routes are also restored alongside with the rest of artifacts and data, and with the exact same configuration as the application in Production cluster.

Project: keycloakrestored ▾

### Routes

Filter ▾ Name ▾ Search by name... /

Name ▴	Status	Location ▴	Service ▴
 keycloak	✓ Accepted	<a href="https://keycloak-keycloak.apps.ocpdemo.homelab.local">https://keycloak-keycloak.apps.ocpdemo.homelab.local</a>  	 keycloak
 keycloak-ext	✓ Accepted	<a href="https://keycloakocp.vlatam.net">https://keycloakocp.vlatam.net</a>  	 keycloak

So, we have two Routes in the DR Cluster after the application is restored, and that won't work if we try to access the application. What should we do?

The Route using the internal domain should be modified to use the proper domain in OpenShift on AWS. This could be done by using one of these options:

- Modifying the Route YAML file to change the hostname, using the proper default domain in OpenShift

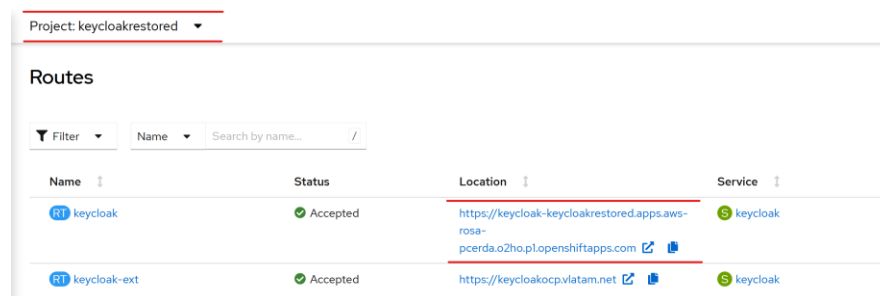
```
spec:  
  host: keycloak-keycloakrestored.apps.aws-rosa-pcerda.o2ho.p1.openshiftapps.com
```





- Deleting and recreating the Route in the DR Cluster. OpenShift will automatically create the new Route using the proper default domain.
- Changing the hostname using a Transform during the Import/Restore process. This can be done using the same steps describe earlier to change the Storage Class used by the application.

In the other hand, the Route using the custom domain can still be used in the DR Cluster as long as we modified the DNS records in order to point to the proper address exposed in the DR Cluster.

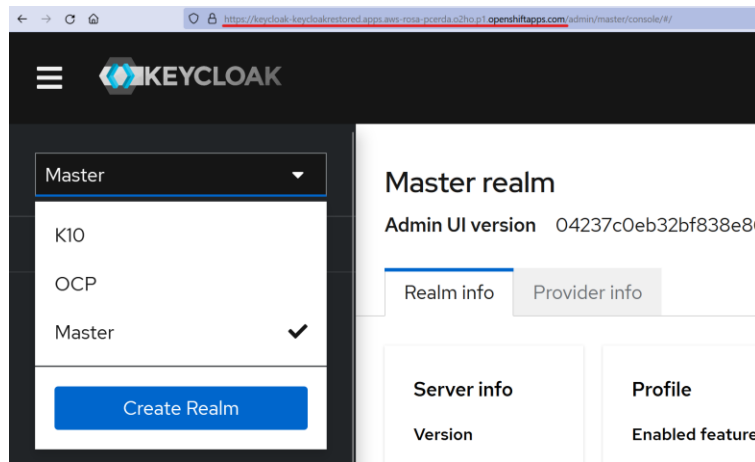
**Tip:** In scenarios where the application will be recovered in the DR Cluster using a CI/CD pipeline, and using Kasten just to recover the persistent data, the CI/CD pipeline could be configured to create the OpenShift Routes with the proper hostnames. Using Continuous Deployment, we can even automatically update the DNS Records to point to the proper addresses in the DR Cluster after the applications are restored.

The final configuration could be something like this:



Project: keycloakrestored			
Routes			
Filter	Name	Search by name...	
Name	Status	Location	Service
 keycloak	Accepted	<a href="https://keycloak-keycloakrestored.apps.aws-rosa-pcerda.o2ho.p1.openshiftapps.com">https://keycloak-keycloakrestored.apps.aws-rosa-pcerda.o2ho.p1.openshiftapps.com</a>	 keycloak
 keycloak-ext	Accepted	<a href="https://keycloakocp.vlatam.net">https://keycloakocp.vlatam.net</a>	 keycloak

Once the OpenShift routes are properly set, we can access our Keycloak application from the DR Cluster as we can see in the next image:



## Option 2: Disaster Recovery using Kasten DR feature

In this section we will focus on how to implement a Disaster Recovery strategy to restore all Kubernetes applications into another Kubernetes cluster in a different location, using the native **Kasten Disaster Recovery** feature.

According to Kasten official documentation, K10 Disaster Recovery (DR) aims to protect K10 from the underlying infrastructure failures. In particular, this feature provides the ability to recover the K10 platform in case of a variety of disasters such as the accidental deletion of K10, failure of underlying storage that K10 uses for its catalog, or even the accidental destruction of the Kubernetes cluster on which K10 is deployed.

By recovering Kasten K10 configuration with the Disaster Recovery feature allows us to recover the entire Kasten catalog which includes information about the applications backed up with Kasten, and all the available restore-points on the Location Profiles. By using this catalog, and once Kasten K10 is restored, it will be possible to restore all applications in the DR Kubernetes cluster.

This strategy is meant for the following scenarios:

- **Hybrid environments:** We have applications running in an on-premises Kubernetes cluster like RedHat OpenShift, Suse Rancher or VMware Tanzu, and want to restore them into another Kubernetes cluster provided by a Public Cloud like AWS EKS, Azure AKS or GCP GKE for Disaster Recovery purposes.
- **Cross-cluster (on-premises to on-premises):** We have applications running on-premises, using a Kubernetes cluster like RedHat OpenShift, Suse Rancher or VMware Tanzu, and we want to restore them into another Kubernetes cluster running in the same site or in a remote location.
- **Cross-Cloud:** We have applications running in a Kubernetes Cluster provided by a Public Cloud like AWS EKS, and we want to restore them into another Kubernetes cluster provided by a different Public Cloud, like Azure AKS.
- **Cross-Region/Cross-Account:** We have applications running in a Kubernetes Cluster provided by a Public Cloud provider like AWS EKS, and we want to restore them into another Kubernetes cluster provided by the same Cloud Provider, but running in another Region or Account/Subscription.

**TIP:** The main difference between using Kasten DR feature and using the export/import approach, is:

- By using the export/import approach it is possible to have the Kubernetes cluster and all applications ready to be used in the DR cluster at any time, as we can have all the applications data and metadata imported and restored every time the Import policy runs according to schedule. This can significantly reduce the RTO (recovery time objective) when a disaster occurs, as the applications data has been already restored.
- When using the Kasten DR feature, the Kasten configuration will be restored at the time the disaster occurs, and then all the applications data must be restored accordingly, which of course will take some time depending on the number of applications and the size of their data.

For this scenario we will be using AWS Elastic Kubernetes Service (EKS) to provide the Production and the DR Kubernetes Cluster. Both clusters will be running in different AWS regions, Production cluster in eu-west3 (Paris) and DR cluster in eu-west2 (London)

### Cluster Information

K10 Version	5.0.7 <a href="#">Upgrade to Version 5.0.9</a>
K10 Namespace	kasten-io
Kubernetes Version	v1.23.10-eks-15b7512
Kubernetes Release Type	AWS

Both, Production and DR Kubernetes clusters will be using AWS S3 buckets as Location Profiles.

## Stage 1: Enable Kasten Disaster Recovery feature

To enable K10 DR, a Location Profile needs to be configured. This will use an object storage bucket or an NFS file storage location to store data from K10's internal data stores and the cluster will need to have write permissions to this location. We can use one of the Location Profiles set previously to store the applications' backup data. It's recommended to use an Object bucket with immutability enabled to protect the Kasten configuration against ransomware attacks.

### Location Profiles

Create profiles that **define credentials and locations** needed to move data in and out of the cluster. You'll select from these profiles when creating policies or exporting a restore point.

[+ New Profile](#)


LOCATION PROFILE <b>pcerda-k10immutable</b> 				
CLOUD PROVIDER	REGION	BUCKET NAME	OBJECT IMMUTABILITY	PROTECTION PERIOD
AWS S3	EU (Paris) • eu-west-3	pcerda-k10immutable	<span>Enabled</span>	1 day
STATUS <span>Valid</span>				

K10 DR settings can be accessed from the Settings icon in the top-right corner of the dashboard or, for a new install, via the prompt at the bottom of the dashboard. On the **Settings** page, select **K10 Disaster Recovery** and then click the **Enable K10 DR** button to enable disaster recovery.

## K10 Disaster Recovery

K10 Disaster Recover creates a policy that snapshots K10 data stores and exports to cloud storage using settings defined in the provided location profile.

Also, K10 DR requires a user-provided passphrase for encryption. This **passphrase has already been created**. Please ensure it has been stored securely since it will be required for a recovery.


 **Enable K10 DR**

A Location Profile and a Passphrase will need to be provided to enable disaster recovery. The passphrase is required for encryption and needs to be saved securely outside the cluster.

### K10 DR

#### Cloud Location Profile


Select a location profile for exported K10 DR backups. When restoring K10 you will need to provide these credentials. **Please store them securely.**

 pcerda-k10immutable

#### Passphrase

A passphrase is required for encrypting the snapshot data. You will need to supply this passphrase if you perform a restore, so please **store this passphrase securely**.

●●●●●●●●●●●●●●●●



☒ Somewhat strong passphrase

After enabling K10 DR feature, it is essential that you copy and save the following to successfully recover K10 from a disaster:

- The **cluster ID** displayed on the disaster recovery page once the DR feature is enabled.
- The DR **passphrase** entered when K10 DR was enabled.
- The credentials and object storage bucket or the NFS file storage information (Location Profile) where DR data will be backed up.

## K10 Disaster Recovery

K10 Disaster Recover creates a policy that snapshots K10 data stores and exports to cloud storage using settings defined in the provided location profile.

Also, K10 DR requires a user-provided passphrase for encryption. This **passphrase has already been created**. Please ensure it has been stored securely since it will be required for a recovery.

**K10 Disaster Recovery is enabled.**

Save the cluster ID displayed below. It will be needed during the restore process.

628e8e24-0d61-4924-a836-61e500aa3ea

copy

Disable K10 DR

**NOTE:** Without this information, K10 Disaster Recovery will not be possible.

Once K10 Disaster Recovery is enabled and properly configured, we will have a new Policy created and scheduled to run by default every 4 hours, sending the backup data to the Location Profile selected previously.

## Stage 2: Protect our applications

As we have already described in the previous scenario (Disaster Recovery using Kasten Export/Import features), in Production cluster we need to complete the next steps in order to protect our applications:

1. We must set one or more Location Profiles. For this example, in Production Cluster we are using an AWS S3 Bucket called **pcerda-k10** to backup/export the application data using Kasten policies. This bucket has a replication rule in AWS S3 to replicate all the data from this bucket to another bucket called **pcerda-k10-dr**, which can be used later to import the application data in the DR cluster.
2. Create Kasten Policies with the **Enable Backup via Snapshot Exports** option to back up the applications and export the data to one of the Location Profiles. These steps were already described in depth in the previous scenario, so please refer to it for detailed steps.

### Policies

Policies are used to automate your data management workflows. To achieve this, they combine actions you want to take (e.g., snapshot), a frequency or schedule for how often you want to take that action, and a label-based selection criteria for the resources you want to manage.

Create New Policy

Filter by Name

**wordpress-backup**

Valid

wordpress

Snapshot **daily** and retain 4 daily snapshots

Export **daily** snapshots using the export profile **pcerda-k10** and retain 4 daily exported snapshots

Export volume data for durable backups

Show import details...

**postgresql-backup**

Valid

postgresql

Snapshot **daily** and retain 4 daily snapshots

Export **daily** snapshots using the export profile **pcerda-k10** and retain 4 daily exported snapshots

Export volume data for durable backups

Show import details...

**pacman-backup**

Valid

pacman

Snapshot **daily** and retain 4 daily snapshots

Export **daily** snapshots using the export profile **pcerda-k10** and retain 4 daily exported snapshots

Export volume data for durable backups

Show import details...

**TIP:** When using this DR strategy, it's very important that the DR Kubernetes cluster has access to the Object Storage buckets or NFS shares that contains the actual backup data. This can be accomplished in two different ways:

- **Object Storage buckets or NFS shares are accessible from source (Production) and target (DR) Kubernetes cluster.** For instance, we can use AWS S3 buckets, so even if we lose the entire production Kubernetes cluster, the S3 bucket is still available and visible for the DR Kubernetes cluster.
- **Backup data is replicated to another Object Storage bucket or NFS share.** For instance, we can enable replication for an S3 bucket (AWS S3 or S3 compatible), so we can have a second S3 bucket, in a different location, with the replicated backup data. While backups of production Kubernetes cluster are sent to the S3 bucket, the DR Kubernetes cluster can access to backup data from the replicated bucket (alternate location).

## Stage 3: Restore Kasten configuration after a Disaster

In case a disaster occurs in our Production Kubernetes cluster, we can use the Kasten DR feature to restore Kasten configuration, and then restoring all our applications in our DR cluster.

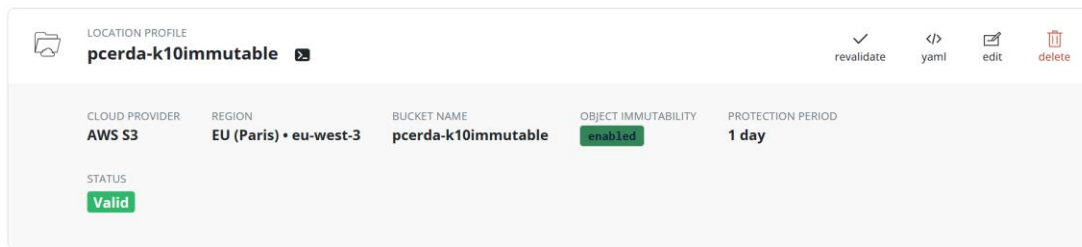
**IMPORTANT:** To be able to restore the Kasten configuration and our applications, we need first to provision a new Kubernetes cluster, whether on premises or using a Cloud provider like AWS EKS. The most popular Kubernetes solutions, like RedHat OpenShift, SUSE Rancher, AWS EKS o Azure AKS, provides features to automate the deployment of a new Kubernetes cluster which of course will speed up the recovery process.

Once we have a new Kubernetes cluster for DR up and running (in our case we will be using an AWS EKS cluster running in a different region), we need to follow the next steps:

1. Create a namespace for Kasten in the DR Kubernetes cluster. By default, **kasten-io**.
2. Create a Kubernetes Secret called k10-dr-secret, using the passphrase provided while enabling DR.

```
kubectl create secret generic k10-dr-secret \
  --namespace kasten-io \
  --from-literal key=<passphrase>
```

3. Then, we need to install a new K10 instance in our DR Kubernetes cluster, in the same namespace where the secret was created. Please follow the proper Kasten installation steps depending on the Kubernetes distribution and your configuration requirements.
4. Once K10 is installed and available, we need to set a new Location Profile using **the same** Object Storage or NFS File Share we used in **Stage 1** when we enable the DR feature in the Production Kubernetes cluster. This Location Profile contains all the K10 configuration of our Production Kubernetes cluster which we need to restore. In our scenario, we will use an AWS S3 bucket called *pcerda-k10immutable*.



- We will also need the Cluster ID we got when we enabled the DR feature in the Production Kubernetes cluster.
- Now, we can run the next command to restore the Kasten configuration from Kasten DR Backup. In this command we need to provide the Location Profile name where the DR Backup is stored (set in previous step) and the Cluster ID we got when we enabled the Kasten DR Feature.

```
helm install k10-restore kasten/k10restore --namespace=kasten-io \
  --set sourceClusterID=<source-clusterID> \
  --set profile.name=<location-profile-name>
```

The previous command will restore **all the Kasten configuration**, including Location Profiles, Policies, Secrets and the Restore Point catalog for all the applications that were protected by Kasten in the Production Kubernetes cluster.

## K10 Restore

✓ **Restore succeeded.**

Close

**NOTE:** While the restore process is running, you will see a “**Restore in Progress**” message in Kasten dashboard, which prevent you to run any action in

Kasten until the process is complete. When the restore process is complete, in the Kasten dashboard you will be presented with a message like the image above, reporting the restore has been completed successfully. At this point we can continue with the next steps of the Disaster Recovery plan. The entire Kasten restore process should take **just a few minutes**, as at this point we are not yet restoring any application.

When running the restore process for Kasten configuration on DR Kubernetes cluster, and the Location Profiles are **not available or accessible** from DR Kubernetes cluster, we can use an **alternate Location Profile** to restore the application data. For instance, we could be using a replication policy in the AWS S3 bucket used as a Location Profile (also mentioned in the Export/Import approach), to create a replica of the bucket with the backup data in another region. Then, we could use this replicated copy as an Alternate Location to restore the application data.

**NOTE:** To use an Alternate Location to restore the applications, the location must be added previously to Kasten configuration. For instance, in our case we have an AWS S3 bucket called *pcerda-k10-dr* with replicated backup data from the original AWS S3 bucket, called *pcerda-k10*, used as a Location Profile in the Production Kubernetes cluster.



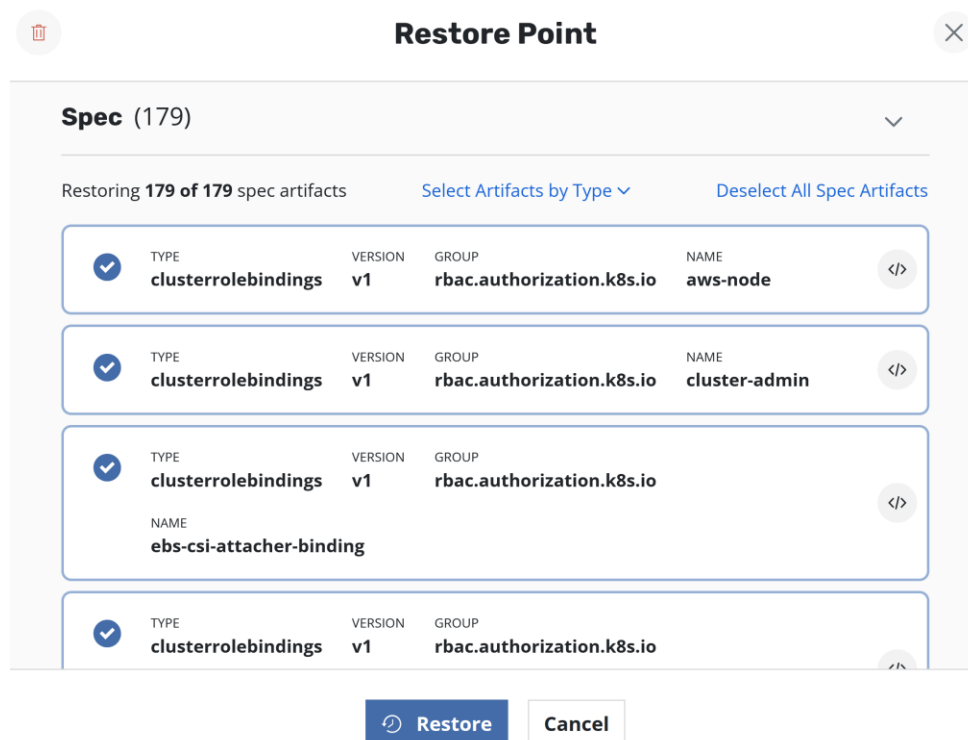
## Stage 4: Recover the cluster-scoped resources.

Prior to recovering applications, it may be desirable to restore cluster-scoped resources. Cluster-scoped resources may be needed for cluster configuration or as part of application recovery.

Once the tasks to restore of Kasten Configuration using the Kasten DR feature is completed as described in the previous step, go to the Applications section in Kasten UI, hover on the Cluster-Scoped Resources card, click on the restore icon, and select a cluster restore point to recover from.

When selected the proper restore point, we can choose whether to restore all cluster-scoped resources, or manually select which artifacts will be restored.

**NOTE:** This is one of the major differences between using Kasten and using ETCD backups to protect Kubernetes resources. By using Kasten we can run backup policies automatically with different scheduling options, having multiple restore points, and also, we can have a lot of granularity to choose what specific artifacts we want to restore, which provide a lot of flexibility in compare with ETCD backups.



**Restore Point**

**Spec (179)**

Restoring **179 of 179** spec artifacts [Select Artifacts by Type](#) [Deselect All Spec Artifacts](#)

TYPE	VERSION	GROUP	NAME
<input checked="" type="checkbox"/> clusterrolebindings	v1	rbac.authorization.k8s.io	aws-node
<input checked="" type="checkbox"/> clusterrolebindings	v1	rbac.authorization.k8s.io	cluster-admin
<input checked="" type="checkbox"/> clusterrolebindings	v1	rbac.authorization.k8s.io	ebs-csi-attacher-binding

**Restore** **Cancel**

Once selected the components we want to restore, just click in **Restore** to start the restore process.

## Stage 5: Restore Applications

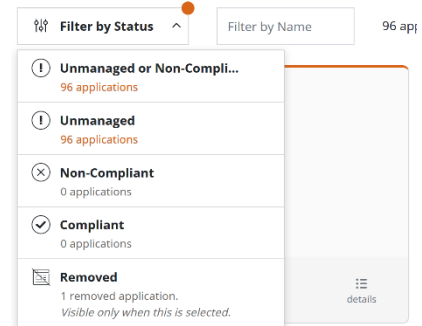
So far, we have restored the Kasten configuration, including the Restore Points catalog for all applications protected in Production Kubernetes cluster, and we have restored the cluster-scoped resources in the DR cluster. Now we can proceed with the restoring of all applications in the DR cluster using the native Kasten features.

This restore process can be done manually for every application, or it could be automated by using the [Kasten API](#) or third-party solutions like ArgoCD.

First, go to the **Applications** section in the Kasten UI, select **Removed** under the **Filter by status** drop-down menu. Then click restore under the application we want to restore and select a restore point to recover from.

### Applications

View details or perform actions on applications.



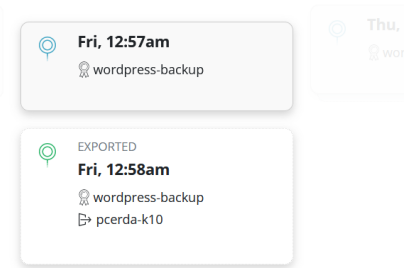
**TIP:** If we are trying to recover our applications in a Kubernetes cluster running in a different location (for instance in our scenario we are restoring the applications in a different AWS Region, eu-west-2), then we should choose a restore point of type “**Exported**” instead of using a Snapshot to restore the data. This means we should restore the applications from the data backed up (exported) to a Location Profile (AWS S3 Bucket in our case), as by default the Snapshots will be available only in the same AWS region where they were taken (eu-west-3 in our scenario).

If we try to use a snapshot to restore the application data, we could get an error message like this:

*Failed to create volume from snapshot. InvalidZone.NotFound: The zone 'eu-west-2b' does not exist. status code: 400, request id: 46215ee1-7fa2-445c-bfc8-491b2ebdc3fb*

### Select an instance...

This restore point has multiple instances - one that is native to the cluster and another that resides outside the cluster.



Once we select the restore point, we want to use to restore the application, then we need to specify the Namespace where the application will be restored. As we are restoring the applications in a new Kubernetes cluster, the original Namespaces don't exist. In this case, we can create a new Namespace, during the restore process, with the desired name as we can see in the image below:

Restore Point

Exported Restore Point

This exported restore point is stored outside the cluster  
It will require additional time to download, import, and restore from.

ORIGINATING POLICY

kasten-io/wordpress-backup

EXPORT PROFILE

EXPORT TYPE

pcerda-k10

Complete snapshot data was  
exported for portable backups.

KUBECTL COMMAND

\$ kubectl get --raw /apis/apps.k8s.io/v1alpha1/restorepointcontents/wordpress-sched

copy

Application Name

Select a namespace to restore into. The contents of the selected namespace will be  
overwritten with the restored application.

Select a Namespace...

Create a New Namespace

New Namespace

wordpress

Create

Cancel

Next, we can choose whether we want to restore the entire application or just specific Volumes or Artifacts. For instance, if we could use a CI/CD solution to redeploy the entire application in the target Kubernetes cluster, and then using Kasten to restore just the Persistent Volumes (PVs). Remember using CI/CD solutions can't provide protection for persistent data.

35 | Page

.Cerde v1.0 2023

### Artifacts

By default, all artifacts in the restore point will be restored. However, you can deselect artifacts to exclude them from the restore.

[Deselect All Artifacts](#)


#### Snapshot (2)

Restoring 2 of 2 volumes [Deselect Volume Snapshots](#)

<input checked="" type="checkbox"/>	KUBERNETES VOLUME PVC <b>wp-pv-claim</b>	STORAGE CLASS NAME <b>gp2</b>	SIZE <b>150.7 MiB</b>
<input checked="" type="checkbox"/>	KUBERNETES VOLUME PVC <b>mysql-pv-claim</b>	STORAGE CLASS NAME <b>gp2</b>	SIZE <b>199.3 MiB</b>

#### Spec (10)

Restoring 10 of 10 spec artifacts [Select Artifacts by Type](#) [Deselect All Spec Artifacts](#)


	TYPE	VERSION	NAMESPACE	NAME	
<input checked="" type="checkbox"/>	configmaps	v1	wordpress	kube-root-ca.crt	

In case the Location Profile where the application data was sent to by using a Kasten Policy is not available, we can use an Alternate Location Profile to restore the application from. In our scenario remember we have an AWS S3 bucket (pcerda-k10) with a replication policy to replicate all data to another AWS S3 bucket (pcerda-k10-dr). This bucket with the replicated data can be used to restore the applications by clicking in **Alternate Location Profile** option, and then choosing the proper Location Profile to restore the data from.

### Restore Point

☒ **Alternate Location Profile**

By default, an external restore point will be restored using the same location profile it was exported with (pcerda-k10). Select a different profile if, for example, the restore point was copied to another cloud location.


 pcerda-k10-dr





Please remember that, as mentioned in previous scenario, as we are recovering our applications in a different Kubernetes cluster, in a different location, some transformations could be required. For instance, one of the most common transformations is the use of a different Storage Class in the target Kubernetes cluster.

### Edit Policy

☒ **Apply transforms to restored resources**

On restore, change the contents of spec resources. This may be useful when migrating between environments. For example, you can change storage classes or edit container image names.

 Add New Transform

 changeStorageClass   

In our case, as we are using AWS EKS to host both, Production and DR clusters, we have the same Storage Classes available in both clusters, so no Transform is required in this matter. Other transforms could be required to make applications accessible from outside the cluster if required.

Finally, we click in “Restore” to start the restore process for the selected application. As we can see in the dashboard, the application was successfully restored:

Actions (2)		Filter		Page 1	
COMPLETED	PHASES	TARGET NAMESPACE	ARTIFACTS	START	
<b>Restore</b>	✓ Restoring Application Components	<b>wordpress</b>	2 volume • 40 GiB	Today, 6:55pm	...
wordpress-zwjc2	✓ All phases completed successfully.	RESTORE POINT		DURATION	
		<b>wordpress-scheduled-9fxvb</b>		37 secs	

## Stage 6: Make the applications accessible

We have already described the process to protect the applications running in our Kubernetes cluster, and then how to recover them in an alternate location. There is still a missing step in order to make sure the applications can be actually used.

Some of the applications running in a Kubernetes cluster are designed to be accessible by users, like web applications. In the same way, other applications could be designed to be accessible for other applications running out of the Kubernetes clusters. Usually all these applications are accessible by using external load balancers, Ingresses or Routes (OpenShift).

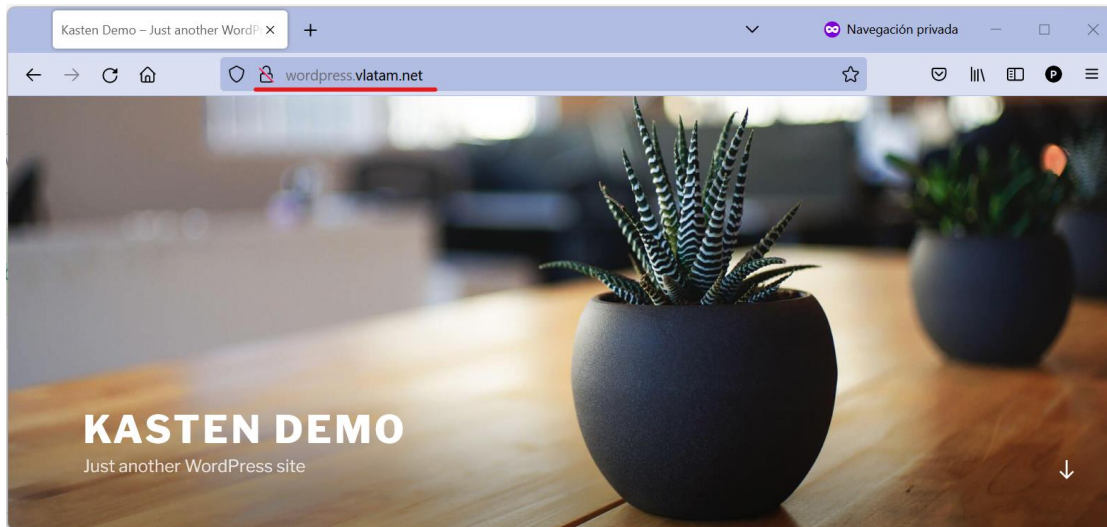
When we recover all our applications in a different cluster, in a different location, it's very likely we need to make some additional changes to make sure the applications are available after being recovered.

For example, in our Production cluster we have a Wordpress application using a LoadBalancer service to expose the application in the Kubernetes cluster, so users can access the Wordpress website using the URL provided by AWS as we can see in the picture below:

```
rke@K8sJumpbox:/$ k get svc -n wordpress
NAME      TYPE          CLUSTER-IP   EXTERNAL-IP                                     PORT(S)          AGE
wordpress LoadBalancer 172.20.24.34  ab79bb59923fc4d1bbcf472915ca3717-1213128995.eu-west-3.elb.amazonaws.com  80:31704/TCP    20m
wordpress-mysql ClusterIP      172.20.101.219 <none>          3306/TCP         20m
rke@K8sJumpbox:/$
```

Of course, this kind of URL are not really user-friendly, so usually we will use DNS records to create a custom FQDN redirecting the traffic to the URL generated by AWS. For example:

Hostname	IP/URL	Action
*.vlatam.net		Modify
wordpress.vlatam.net	<u>ab79bb59923fc4d1bbcf472915ca3717-12131 28995.eu-west-3.elb.amazonaws.com</u>	Modify



After we restore the application in the DR Cluster (AWS EKS in a different region), we can see the LoadBalancer service was restored using a different URL, as these URLs are automatically generated when the LoadBalancer service is created. During the restore process Kasten will basically re-create the Services for the application, so AWS will automatically generate a new URL for the LoadBalancer service.

```
rke@K8sJumpbox:/$ k get svc -n wordpress
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
wordpress     LoadBalancer  172.20.22.8   a4f2995b6e05a4b2aa8163277226b44e-306304967.eu-west-3.elb.amazonaws.com  80:32033/TCP    2m29s
wordpress-mysql ClusterIP      172.20.10.114 <none>         3306/TCP         2m28s
rke@K8sJumpbox:/$
```

In this case, we basically need to update the DNS records in order to point to the proper addresses exposed in the DR Cluster. Once the DNS records are properly updated, we can access our Wordpress application from the DR Cluster:

Hostname	IP/URL	Action
*.vlatam.net	81.44.32.181	Modify
wordpress.vlatam.net	<u>a4f2995b6e05a4b2aa8163277226b44e-306304967.eu-west-3.elb.amazonaws.com</u>	Modify

## How to choose the Disaster Recovery Strategy

In this document we have presented two different Disaster Recovery strategies to protect the applications running in a Kubernetes cluster. Now the question is, how to choose the proper strategy that fits better with our scenario and specific requirements? Let's discuss about both options and their pros and cons.

### Using Kasten Export/Import features

The Export/Import features were designed in the first place for application migration, but it has become a great option to implement a Disaster Recovery strategy, specially to **reduce the recovery times (RTO)**.

#### Pros

- It can provide an active-active configuration. It reduces the overall time required to run the Disaster Recovery Process as we already have a second Kubernetes cluster up and running, with Kasten already installed, and the applications imported and ready to be restored in case of a disaster.
- It reduces the application's recovery time by having the applications running in two different Kubernetes clusters simultaneously (when we choose to restore the application's data after importing it).
- It uses a fresh new Kubernetes cluster and Kasten instance, which allows us to restore all our applications in a new and clean environment.

#### Cons

- It increases the cost of the overall solution, as we need to "duplicate" the entire Kubernetes environment, despite we can reduce the scale of the Kubernetes clusters in the DR site.
- Not all the applications are designed to run simultaneously in two different clusters and keep data consistency. In this case, we can import the applications, but leaving the actual restore process on hold until the production Kubernetes cluster fails. This will increase the recovery time as we need to restore the entire applications and their data AFTER a disaster strikes.

### Using Kasten Disaster Recovery feature

As already mentioned, the Kasten Disaster Recovery feature allows to restore a Kasten instance and all its configuration and catalog in case of a disaster, **without extra running costs**.

#### Pros

- It reduces costs because we don't need the DR Kubernetes cluster running simultaneously alongside the production Kubernetes cluster.
- Simple approach that allows to restore the entire Kasten instance and all its settings. No need for extra configuration in Kasten once restored.

## Cons

Recovery times are higher than the previous strategy, because when a disaster strikes, we need to deploy the Kubernetes cluster, install Kasten and restore Kasten configuration before we can actually start restoring our applications.

## How to Automate the Disaster Recovery Strategy

So far, we have described two different Disaster Recovery strategies that can be used to protect our applications running in a Kubernetes cluster. However, manually running the required steps to restore the applications once a disaster strikes, using either strategy, usually isn't an option because the number of applications and data to restore, and/or because we have aggressive RTOs that require to streamline the recovery process.

Thus, **automation is a key component** when designing any Disaster Recovery strategy. And of course, both strategies described in this document can be automated with third-party solutions and leveraging the Kasten API. In this section we will provide a brief description on how to automate the Kubernetes DR strategy and what solutions can be useful for this purpose.

## Deploying the Kubernetes clusters

Usually the most popular Kubernetes solutions, like RedHat OpenShift, AWS EKS, Azure AKS and others offer simple wizards to deploy a new Kubernetes cluster whenever is needed, reducing the manual steps.

In addition, in order to streamline and fully automate the process, it is possible to use third party solutions like [Terraform](#), which allows to deploy a new Kubernetes cluster with a single command.

## Deploying Kasten K10

Deploying Kasten K10 is fairly simple using Helm, which of course provide multiple parameters than can be used depending on the Kubernetes solution where Kasten is going to be installed. However, installing Kasten isn't enough, as it's also necessary to configure some basic settings like creating Location Profiles, creating policies, and so on.

Deployment and configuration of a new Kasten K10 instance can also be automated by using Configuration Management solutions like Ansible, Chef or Puppet.

Some examples of how to use Ansible to automate the deployment and configuration of Kasten K10 can be found in this GitHub project:

[https://github.com/prcerda/K10-ExportImport-Ansible/blob/main/playbook/dr\\_cluster/01\\_k10\\_install.yaml](https://github.com/prcerda/K10-ExportImport-Ansible/blob/main/playbook/dr_cluster/01_k10_install.yaml)

**IMPORTANT:** This GitHub project is just an example of a deployment and it's meant to be used for testing and learning purposes only. Do not use in production.



## Exporting and Importing Applications with Kasten

Once we have a functional Kasten instance, we need to create the Kasten Policies to protect (Export) the applications running in Kubernetes. In case of using the Export/Import strategy, we also need to create the Import policies in the DR Kubernetes cluster so we can import the applications in a different cluster for disaster recovery purposes.

Importing applications could also require the use of Transforms in Kasten policies, for example in case we are importing the applications in a Kubernetes cluster with different Storage Classes.

Kasten policy management can also be automated using Configuration Management solutions like Ansible, Chef or Puppet. Some examples of how to use Ansible to automate the creation of Export and Import policies in Kasten K10, including the use of Transforms, can be found in this GitHub project:

<https://github.com/prcerda/K10-ExportImport-Ansible>

**IMPORTANT:** This GitHub project is just an example of a deployment and it's meant to be used for testing and learning purposes only. Do not use in production.

## Using the Kasten Disaster Recovery feature

In case of using the native Kasten Disaster Recovery feature, after deploying Kasten K10 in a new Kubernetes cluster, we need to configure at least one Location Profile and then restore the Kasten configuration. Just once the Kasten configuration is restored we will be able to restore all the applications.

Recovering Kasten K10 using the Disaster Recovery feature of course can be automated using Configuration Management solutions like Ansible, Chef or Puppet. Some examples of how to use Ansible to restore the Kasten configuration and all the applications, can be found in this GitHub project:

<https://github.com/prcerda/k10-dr-ansible>

**IMPORTANT:** This GitHub project is just an example of a deployment and it's meant to be used for testing and learning purposes only. Do not use in production.

## Restoring applications

So far, we have mentioned how to automate the deployment of a Kubernetes cluster, installing and configuring Kasten, and creating/recovering the Kasten policies to export/import the applications. One last step of course is actually restoring the applications.

We can use Kasten K10 to restore every application manually using one of the available restore points, as we have already described in both Disaster Recovery strategies. However, in a real-world scenario this could be quite slow, and it would be really difficult to meet the RTO requirements, so again **automation** is important.

Here we have 2 different approaches to restore applications:

- Restoring the entire applications from the available Restore Points in Kasten. This can be fully automated using solutions like Ansible. An example can be found in this GitHub project: [https://github.com/prcerda/k10-dr-ansible/blob/main/playbook/aws\\_eks/03\\_k10\\_restoreapps.yaml](https://github.com/prcerda/k10-dr-ansible/blob/main/playbook/aws_eks/03_k10_restoreapps.yaml)
- Re-deploying the applications in the new Kubernetes cluster, and then restore only the application's data using Kasten. This can be fully automated by using:
  - CD solutions like ArgoCD, which can be used to deploy the applications in a specific cluster.
  - Solutions like Ansible to run commands and/or apply YAML manifests that can be used to restore the application's data from restore point available in Kasten.
  - An example about using ArgoCD and Ansible to restore the applications and application's data can be found in this GitHub Project: <https://github.com/prcerda/K10-RestoreApp-ArgoCD-Ansible>

**IMPORTANT:** This GitHub project is just an example of a deployment and it's meant to be used for testing and learning purposes only. Do not use in production.

So, as we can see we are plenty of solutions that can provide the tools to automate the whole Disaster Recovery strategy for all the applications running in a Kubernetes cluster.

## About the Author

Patricio Cerda is presently a Senior Solutions Architect based in Spain, specialized in microservices with focus on Kubernetes and Kasten. He has more than 20 years of experience in the IT industry, with special focus in VMware SDDC solutions and AWS before joining Veeam Software in 2021.

## About Kasten by Veeam

Kasten by Veeam® is the leader in Kubernetes backup. Kasten K10 is a Cloud Native data management platform for Day 2 operations. It provides enterprise DevOps teams with backup/restore, disaster recovery and application mobility for Kubernetes applications. Kasten K10 features operational simplicity and integrates with relational and NoSQL databases, all major Kubernetes distributions, and runs in any cloud to maximize freedom of choice. Our customers are confident that their Kubernetes applications and data are protected and always available with the most easy-to-use, reliable and powerful Cloud Native data management platform in the industry. For more information, visit [www.kasten.io](http://www.kasten.io) or follow [@kastenhq](https://twitter.com/kastenhq) on Twitter.