NKE->NKP migration workflow using Velero and Nutanix objects

| Version Number | Published | Author |
| --- | --- | --- |
| 1.0 | July 2024 | Christophe Jauffret  Arvind Bhoj |

# 

[**Disclaimer 3**](#_oieuqs3e8sjg)

[**Determine what workloads will be migrated and in what order (if any) 3**](#_83mtoeldmvty)

[**Check if Infrastructure has enough capacity 4**](#_o6f6wcytsxjq)

[**Pick a Migration Strategy 4**](#_p29j7tarjsz5)

[**1. Big Bang 5**](#_yzd0642k860o)

[**2. Blue Green 6**](#_zaw92gm0rahk)

[**3. Rolling 7**](#_o7n3i5quk484)

[**4. Canary 8**](#_bqpz9grr8k3k)

[**Create a Migration Plan 9**](#_63gipdmvp3uv)

[**Identify Workload Migration Methodology 10**](#_lomfqatmc9v)

[**Implement Migration Plan and Begin Migration 11**](#_fdyr3krnxco)

[**Complete Migration 11**](#_c45q2t2c77ld)

[**Examples 12**](#_zhg5ls24qakq)

[**Design 12**](#_2et92p0)

[Migrating a Workload with Velero 13](#_o02xyzkhuc56)

[**Limitations 14**](#_gahftejn0sau)

[**Prerequisites 14**](#_3znysh7)

[**Velero installation 17**](#_tyjcwt)

[Deploy Velero on NKE cluster 17](#_ycihqu42ui56)

[**Deploy or Reconfigure Velero on NKP cluster 19**](#_f59azh5eybyq)

[**Migrate Stateless applications 25**](#_jqskk411798n)

[**Deploy Stateless Application to the NKE (Source) Cluster 25**](#_l2sl35o8t5tb)

[**Backup Stateless Application on NKE Cluster 25**](#_b9u1jkozzjuy)

[**Migrate Stateless Application to NKP Cluster 26**](#_f6wx5t11y7re)

[**You can verify migrate is successful and test if your application is running as expected. 27**](#_6lag0usnfaiy)

[**Migrate Nutanix Volumes based applications 27**](#_wmsfe0kgr03y)

[**Deploy Volumes based Application to the NKE (Source) Cluster 27**](#_6l7lowxcxcxq)

[**Backup Volumes based Application on NKE Cluster 28**](#_756wcpva9tjt)

[**Migrate Volumes based Application to NKP Cluster 30**](#_98n6z559nlxp)

[**Migrate Nutanix files (dynamic or static) based applications 31**](#_dtt7vkd539jf)

[**Deploy Wordpress app to the NKE (Source) Cluster 32**](#_ixzx5t46aa6s)

[**Backup Application on NKE Cluster 34**](#_zg6o4ceiozsv)

[**Migrate Application to NKP Cluster 36**](#_1d1e64c5mhli)

[Velero post-migration cleaning step 41](#_ot7chfq50hnr)

[**Deprecated Kubernetes API Migration 41**](#_rml15gcfqlwn)

[**Basic Troubleshooting 43**](#_bxa783n8c7ce)

[**For general velero troubleshooting, here is a link to official velero troubleshooting documentation 43**](#_3i6xa3bi0jt2)

[Partially Failed Backup 43](#_rfd4tpbc9iw6)

[BackupStorageLoaction (BSL) not in “Available” Phase/State 43](#_t2r5w6vf893p)

[Troubleshooting for FSB 45](#_wnfhkpu3q2m3)

[**References 46**](#_2s8eyo1)

# 

# Disclaimer

This is a technical guide. It provides an overview of the migration process, some considerations and different migration strategies, including detailed steps with examples for migrating both stateless and stateful applications. The steps provided in this guide are to be used as reference and might need to be modified for specific use cases.

We strongly recommend that every customer running critical workload on NKE engages Nutanix Professional Services to migrate from NKE to NKP. Kubernetes migrations require deep Kubernetes expertise, knowledge of storage and other technologies, and are quite involved to perform.

# Determine what workloads will be migrated and in what order (if any)

The first step in the migration process would be to identify what workloads will be brought over (Migrated) and in what order (if any). Customers with NKP Pro and Ultimate licenses will get Logging (FluentBit, Logging Operator, Grafana Loki), Monitoring (Prometheus, Alert Manager) and other services (Gatekeeper, Dex etc.) out-of-the-box in the platform. As a result, customers migrating to one of those products will not necessarily need to migrate their corresponding applications, namespaces, etc.  
For example on NKE side you can exclude content within the ntnx-system and kube-system namespaces.  
Please use the following link to get the complete list of applications that will be available with the NKP Pro and Ultimate licenses:

<https://portal.nutanix.com/page/documents/details?targetId=Release-Notes-Nutanix-Kubernetes-Platform-v2_12:rel-supported-config-r.html>

In addition to this, some applications might need modernizing and migrations are a perfect time to do those. So, spend some time and plan carefully to bring over only what is required.

At the end of this exercise there will be two sets of customers:

* Customers that are not migrating any workloads.

Those customers don’t need to follow the rest of the guide. They can simply:

* + Deprovision the NKE cluster(s);
  + Provision new NKP cluster(s); and
  + Begin their journey on NKP.
* Customers that will be migrating at least some capacity and need to plan the migration. Those customers can follow the rest of this guide to plan for the migration.

# Check if Infrastructure has enough capacity

Customers that need to migrate workload over from NKE to NKP should do this as the next step after identifying what needs to be migrated (at least at a high level). In this step, the team managing the Kubernetes platform will determine if there is enough capacity to spin up NKP clusters in parallel. This would include the usual suspects:

* Compute - Is there enough capacity to allocate the required CPU/Memory
* Storage - Is the underlying storage enough to carve out disks for the OS and the PersistentVolumes
* Network:
  + Are there enough IP Addresses available in the subnet being used.
  + If a new subnet is being created, will there be any connectivity issues etc.

Please review the NKP resource requirements to get a good understanding of what capacity will need to be allocated for the new cluster(s). In addition to this ensure that for each cluster there is a Virtual IP reserved for the Control Plane HA and one for each Kubernetes service of type LoadBalancer.

<https://portal.nutanix.com/page/documents/details?targetId=Nutanix-Kubernetes-Platform-v2_12:top-resource-req-c.html>

# Pick a Migration Strategy

Once infrastructure capacity and workloads to be migrated (at least at a high level) have been identified, the next step is to pick a migration strategy that works. And finally, you will then migrate workloads depending on their storage needs (see This is a critical step as this will determine the overall process to be followed in the migration project/initiative.

# 

Note that, for customers who have multiple NKE clusters, the same strategy might not be applicable for all clusters. And also the same strategy might not work for all applications within a cluster. In which case a hybrid strategy would be the best.

Here is a brief overview of the different strategies that can be adopted. We have also included high level migration steps and defined scenarios where a given strategy is suitable, to help customers identify the correct strategy and plan accordingly.

#### **Big Bang**

As the name suggests, this method is for doing a migration and cutover from NKE to NKP in one shot without the requirement for both the environments to be up and running at the same time.

Suitable for:

* NKE deployments not running much workload to be brought over to NKP
* Environments with limited capacity (compute/network/storage) where it is difficult to deploy new cluster(s) without deprovisioning old and have a good handle on the process to do a one shot “migration/backup and restore” of resources to perform the migration.

Migration Steps:

* + Identify and backup any Kubernetes resources to be migrated (Velero/Helm Packages/Kustomization/Git Repo/Raw Manifests etc.). If there is nothing to be migrated then skip this step.
  + Power Off the NKE cluster nodes

Note: Do not delete but just power off for now to provide the ability to rollback in case of failure.

* + Build a NKP Cluster that will become the new Kubernetes cluster to run workloads currently running on NKE (or new workloads in case the NKE cluster was not used much.)

(Optional) - Reuse “IPs/DNS” for Ingress Service from the NKE cluster.

* + Deploy/Restore backed up workloads on the new NKP cluster
  + Cutover “Load Balancer/DNS” if they were not reused from the NKE cluster.
  + Test NKP environment
  + Once satisfied, delete NKE infrastructure.

Rollback Steps:

* + Power off NKP cluster nodes
  + Power on NKE cluster nodes.
  + Cutover “Load Balancer/DNS” back to the NKE cluster if they were not reused.

#### **Blue Green**

This migration strategy is a slight variation of the “Big Bang” strategy with the only difference being that it involves spinning up a new NKP cluster in parallel to the NKE cluster and thus requires additional capacity. This makes it easy to rollback in case of a failure during migration and also to “lookup/pickup” resources from the NKE cluster, in case a resource needs to be referred or needs to be brought over manually because it got left behind.

Suitable For:

* Environments which have enough capacity to spin up a new NKP cluster in parallel to NKE, and have a good handle on the process to do a one shot “migration/backup and restore” of resources to perform the migration.

High Level Steps:

* + Build a NKP Cluster that will become the new Kubernetes cluster to run workloads currently running on NKE.
  + Identify and backup any Kubernetes resources to be migrated (Velero/Helm Packages/Kustomization/Git Repo/ Raw Manifests etc.)
  + In the NKE cluster, scale down resources (e.g. Deployments/StatefulSets) that can’t run in parallel in both environments to zero.
  + Deploy/Restore backed up workloads on the new NKP cluster.
  + Cutover “Load Balancer/DNS” to point to services in the NKP environment.
  + Test NKP environment
  + Once satisfied, delete NKE infrastructure.

Rollback Steps:

* + In the NKP cluster, scale down resources (e.g. Deployments/StatefulSets) that can’t run in parallel in both environments to zero. And scale them back to the original replica count in the NKE cluster.
  + Update “Load Balancer/DNS” to point to services/ingress in the NKE environment.

#### **Rolling**

In this strategy a new NKP cluster is spun up in parallel to the NKE cluster just like “Blue/Green” but workloads are not migrated in one shot. Instead they are brought over one application at a time. Each Application cutover is done separately or as a group of Applications. This allows handling complex applications and scenarios.

Suitable For:

* Environments which have enough capacity to spin up a new NKP cluster in parallel to NKE but it is not possible to migrate all the resources from NKE to NKP in one shot.
* Many independent teams use the same NKE cluster but the ownership of deploying applications and managing its lifecycle is not centrally controlled.
* A specific order and timeline to move applications over is determined due to internal/external dependencies and factors for those applications.

High Level Steps:

* + Build a NKP Cluster that will become the new Kubernetes cluster to run workloads currently running on NKE.
  + Identify and backup any Kubernetes resources to be migrated as a part of the Application or set of Applications to be migrated together (Velero/Helm Packages/Kustomization/Git Repo/ Raw Manifests etc.)
  + In the NKE cluster, scale down resources for the Application(s) being migrated (e.g. Deployments/StatefulSets) that can’t run in parallel in both environments to zero.
  + Deploy/Restore backed up workloads for the Application(s) on the new NKP cluster.
  + Cutover “Load Balancer/DNS” to point to services/ingress for the Application(s) in the NKP environment.
  + Test NKP environment
  + Repeat the steps for other Applications till all the workload that is identified to be migrated has been brought over.
  + Delete the NKE infrastructure once all the applications have been migrated successfully and the environment is stable.

Rollback Steps:

* + In the NKP cluster, scale down resources for the Application(s) that were not migrated successfully (e.g. Deployments/StatefulSets) to zero. Optionally delete them from the NKP cluster. Scale them back to the original replica count in the NKE cluster.
  + Cutover “Load Balancer/DNS” to point to services in the NKE environment for the Application(s).

#### **Canary**

This is a variation of the “Rolling” strategy and makes the migration even more granular. In this strategy, instead of cutting over an Application’s traffic completely from NKE to NKP in one shot, only a percentage of traffic is shifted to the equivalent service on the NKP cluster. Then tests are executed. The percentage of traffic is gradually incremented (example, 10% to 20%, then 20%-50% and so). After all the test cases pass and there is good confidence on the Application’s performance in the NKP environment 100% of the traffic is moved to NKP.

Note: This method can only be used for Microservices and Applications that can run in parallel in two clusters without corrupting the integrity of the Application. For environments where some applications fit this profile and some don’t, a “Rolling-Canary” hybrid strategy can be used.

Suitable For:

* Applications and Stateless Microservices which can run in parallel in both NKE & NKP environments at the same time.
* Applications which are customer facing and no downtime/performance degradation is acceptable.
* Environments that have the capability to control splitting percentage of the traffic, either using external load balancer/proxy or a multi-cluster service mesh.

High Level Steps:

* + Build a NKP Cluster that will become the new Kubernetes cluster to run workloads currently running on NKE.
  + Identify and backup any Kubernetes resources to be migrated as a part of the Application or set of Applications to be migrated together (Velero/Helm Packages/Kustomization/Git Repo/ Raw Manifests etc.)
  + Deploy/Restore backed up workloads for the Application(s) on the new NKP cluster.
  + Cutover percentage of the traffic to point to services/ingress for the Application(s) in the NKP environment. For example 10% of the traffic goes to the Application in the NKP cluster while 90% still goes to the instance running in NKE.
  + Perform tests and if all goes well, incrementally increase the percentage of traffic going to NKP till 100% of the traffic goes to the Application instance running in NKP
  + Scale down the Application instance to zero in the NKE cluster
  + Follow this for other applications that fit the profile.
  + Once all the applications have been moved over, decommission the NKE cluster.

Rollback Steps:

* + Cutover 100% traffic back to the Application instance in the NKE cluster

The approach that works best for a customer depends on the current workload that is running on NKE and how much of it needs to be brought over vs redeployed. For some customers that are running minimal workload on NKE, “Big Bang” or “Blue/Green” might be the best approach. While for others that are running complex workloads with multiple dependencies, “Rolling” or “Canary” will work better. For some a hybrid approach would be best. So review and discuss the options carefully.

# Create a Migration Plan

After a Migration Strategy has been finalized, the next step is to create a Migration plan.

For customers that will be following either “Big Bang” or “Blue/Green” strategies described above, this could be a simple plan as everything needs to be moved all at once. However, customers that have to use “Rolling” or “Canary” strategies, as they have a complex environment and can’t bring all services over at the same time, could have a more complex plan that could span weeks/months. These customers should begin planning by identifying which services will be migrated first. These could then be used as a reference by the rest of the organization. An approach that usually works is picking low risk applications first so that the critical applications are migrated after everyone involved is comfortable with the new platform and a stable migration process has been established (which is usually improved iteratively as workloads are migrated).

Like any other project, put everything in a project plan or another tool that works and put timelines against the entire migration.

Identify dependencies on internal and external services and components. This could be:

* Planning for additional capacity
* Network changes, firewall rules to ensure connectivity to required resources
* Mapping out Application dependencies like a Database, Cache server etc.
* Load Balancer / DNS cutover for switching to new Environment

Other considerations:

* Identify which applications will require a downtime and for how long.
* Note down the rollback strategy and notifications etc.
* Create a test plan if not already in place.
* Plan Change Windows if necessary.
* If currently using a CD tool like Argo CD. Assess impact on CI/CD pipeline.
* Check Kubenetes versions and compatibility of resources. The resources being migrated should be compatible with the API version of the target.

Please read [Deprecated API Migration](#_rml15gcfqlwn)

# Identify Workload Migration Methodology

This process can begin in parallel with Migration Planning, the technical team can start looking at tooling that can make the migration a pain free experience.

The primary points to consider while identifying tooling for each workload are:

* What kind of workload is being migrated?
  + Stateless workloads
  + Stateful workloads
  + Short running batch jobs
* How are resources being deployed to the Kubernetes clusters?
  + Using Raw Manifests, Helm Charts or Kustomize
  + CD Tools like Argo CD or Flux CD
* Do any of these applications have their own migration tools?

The answers to those questions will then indicate the correct migration methodology for each workload:

* For applications like Kafka, that have their own migration tools, consider using the native migration tool.
* For stateless workloads (workloads that don’t employ their own storage)
  + If a CD tool is being used, simply point the selected application resources to the new NKP Cluster (Eg, and Application/ApplicationSet in ArgoCD or if using Flux CD (which uses a pull mechanism), create a GitRepository and Kustomization/HelmRelease resource in the NKP cluster to pull the application resources). Consider how this impacts the GitOps or CI/CD pipeline.
  + If there is no CD tool in place then either bring over the resources manually (using helm / kustomize etc.) or use a tool like Velero to move the resource over.

Note: We have included examples of backing up and restoring Stateless & Stateful Applications using Velero in this guide

* For stateful workloads (workloads that do require stateful storage):
  + If a CD tool is being used and the goal is to continue using it for the given application, bring over the resources as described above and then use Velero to migrate the Persistent Volumes over. To achieve this:
    - Migrate Application including PVs using Velero
    - Now delete the Workload Resource (e.g. Statefulset / Deployment)
    - Move the Application Resources over using the CD tool (This will give the resource ownership back to the CD tool)
  + If there is no CD tool being used or if the goal is to stop using the CD tool to manage the stateful application
    - Migrate Application including PVs using Velero

Note: We have included examples of backing up and restoring Stateless & Stateful Applications using Velero in this guide

Update the Migration plan with the detailed findings and plan for the migration.

This guide includes examples on how to use Velero to backup workloads (including stateful workloads) in the NKE cluster and restore them in the NKP cluster to automate the migration process. Please reference those as required. Note that these examples begin by deploying either a Stateless Application or a Stateful application for demonstrating the end to end migration flow. These Applications are not required for the migration itself. After getting a good understanding of the flow, the same procedures can be used to migrate real Stateful/Stateless workload running on NKE.

# Implement Migration Plan and Begin Migration

Begin Migration once a migration plan has been formulated and tooling (if required) has been identified.

Follow the migration plan. Use tooling and instructions as described in the plan. Like any implementation, begin with the easiest applications in non-production environments (like sandbox or dev) and then move to more complex applications and production environments. For large/complex implementations, iteratively build upon the migration plan to add any environment specific or application specific nuances and solutions/workaround.

# Complete Migration

A migration is complete once a cutover takes place. Cutovers are usually controlled by a change in Load Balancer backends or DNS entries. Work closely with the Network team to ensure a smooth cutover once the workload has been migrated and some basic testing has been completed. And rollback in case there were issues identified.

Depending on the complexity/scale and the Migration Strategy picked in the first step, either cutover completely to the new environment or do it more gradually.

At any point please feel to contact your Account Representative for a Professional Service engagement. We wish you a happy migration to NKP - a modern Kubernetes Platform with rich production ready capabilities out of the box.

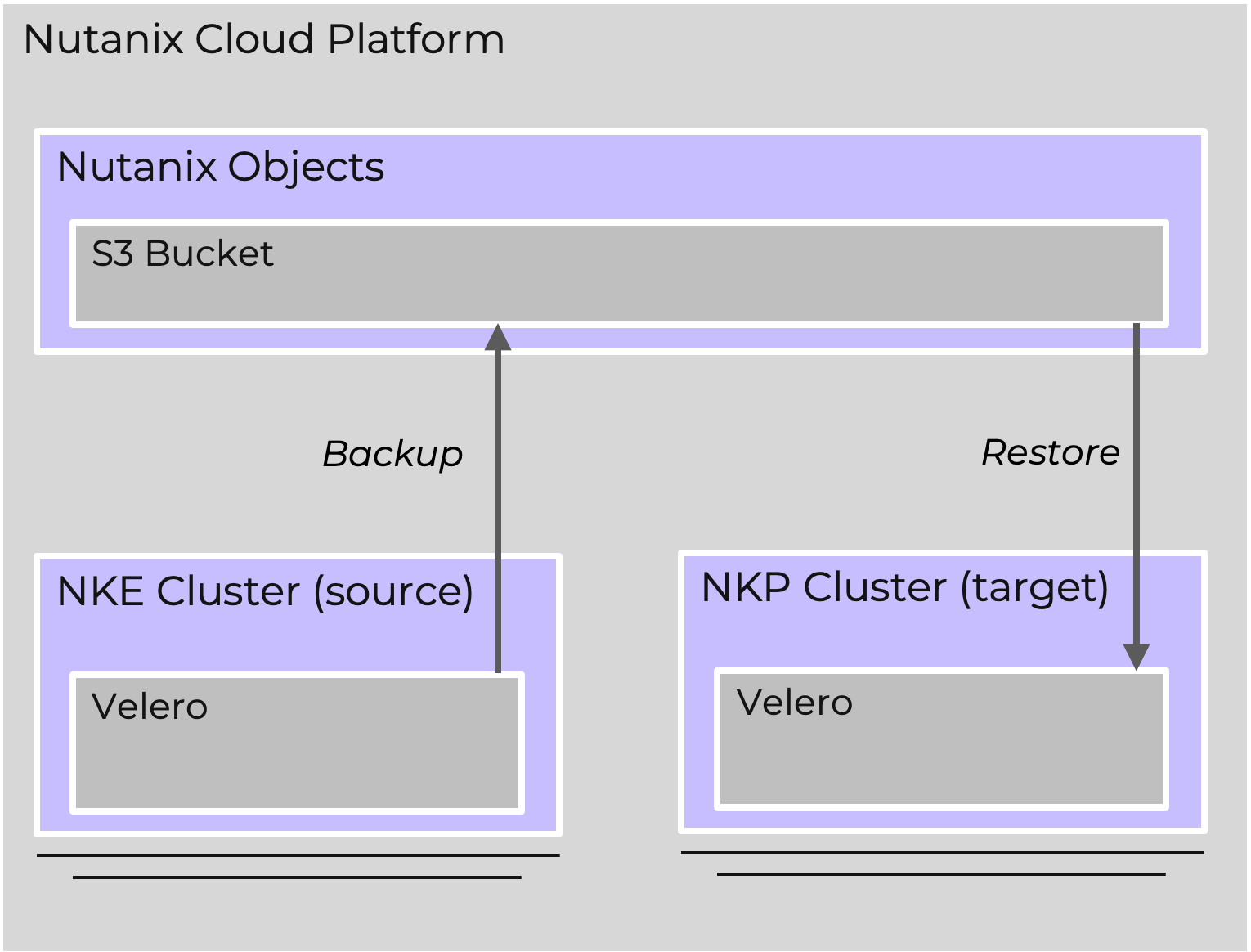
# Examples

Below are examples of moving several different classes of workload: Stateless workloads, stateful workloads with persistent volumes, stateful workloads

### Design

We will leverage Velero and Nutanix Object to perform migration. The main idea is to migrate individually each application between the NKE and the NKP cluster. Of course, system components/namespace will not be migrated and will be directly reinstalled on the NKP cluster first.

# 



# Migrating a Workload with Velero

Here are some examples with detailed steps to use Velero to migrate an application from NKE cluster to NKP cluster. Note that Nutanix Objects is used to store the backups, so it is not necessary for the two clusters (i.e. NKE cluster and NKP cluster) to be up and running at the same time.

To demonstrate the Application migration end to end, we'll begin by deploying sample applications. To demonstrate a Stateless Application migration we are simply creating a Nginx deployment and service. For a Stateful Application example, we use helm to deploy a Wordpress deployment with a MariaDB backend, using persistent volumes (PVs) backed by NutanixVolumes or NutanixFile. We selected Wordpress as the stateful application in the example as it requires one PV for the database and one for files.

With this migration method, the migration scope is the namespace. If you apps spreads between multiple namespaces you need to launch the migration process for each related namespaces.

Note that the step to deploy Wordpress has been added to create a complete end to end example for demonstration purposes only and is not required for the migration itself.

In these examples we call the:

* NKE cluster from where the workload is being migrated as the “Source” cluster; and
* NKP cluster to which we are migrating at they “Target/Destination” cluster

### 

### Limitations

* Application using multiple storage class need advanced approach to migrate, if you are unsure best is to contact Nutanix Professional Services
* Difficulty to have coherent Nutanix files backup with FSB method
* File System backup is changing destination pods, which can cause issue with some deployment tools

### 

### Prerequisites

* Kubectl installed on your laptop => <https://kubernetes.io/docs/tasks/tools/>
* Install the Velero CLI 1.13.2 on your local machine => <https://velero.io/docs/v1.13/basic-install/>

If your cluster don’t have access to internet please follow the airgap install documentation => <https://velero.io/docs/v1.13/on-premises/#air-gapped-deployments>

| Warning |
| --- |
| it is imperative to install the version 1.13.2 of velero, more recent version will cause issue |

* Configure Velero CLI on your desktop to use the correct namespace

velero client config set namespace=kommander



Note: **For NKP starter and Pro licenses, and any self-managed cluster, the namespace will always be “kommander”. However, in case the NKP (target) cluster is a Workload cluster attached to NKP Ultimate's workspace and Velero is deployed via the Workspace Applications, the namespace will be the “workspace namespace” instead of "kommander"**. Please see documentation for [DKP Workspaces](https://portal.nutanix.com/page/documents/details?targetId=Nutanix-Kubernetes-Platform-v2_12:top-workspaces-c.html) for more details in case the target cluster is a Workload cluster.

| Warning |
| --- |
| There is a bug/limitation in Velero with blocks restoring CSI volumes if Velero is not installed in the same namespace in the source and destination clusters.  it is therefore mandatory to use the kommander namespace when deploying Velero in the NKE cluster as this is also the default namespace used for kommander in NKP |

* A Nutanix environment with the following version:
  + PC 2024.1+
  + AOS 6.5.4.5 or 6.8+
* A source NKE k8s cluster is already deployed with the following prerequisites:
  + NKE 2.10 is the tested version for the migration, if you want to use an older version please read carefully the [Deprecated API Migration paragraph](#_rml15gcfqlwn)
  + K8s 1.28.5-1
  + OS 1.7
* A destination NKP cluster is already deployed with the following prerequisites:
  + NKP 2.12
  + K8s 1.29.6
  + Nutanix CSI v3 driver is installed
  + CSI is correctly configured for every storage classed that needs to be used
  + Same Storage Class name as source cluster

*Note: Optionally if the names don’t match or a different StorageClass is being used in the target cluster then create a “change-storage-class-config” ConfigMap to map the StorageClass between the two clusters. More details can be found* [*here*](https://www.google.com/url?q=https://velero.io/docs/v1.3.0/restore-reference/&sa=D&source=docs&ust=1724076436326732&usg=AOvVaw1EAlu0HiXL4YtbFl3oWHQR)*.*

* A Nutanix Objects Store is deployed and configured:
  + An empty bucket is available
  + A key with RW access on this bucket is available

If you don’t have a Nutanix Object Store available, you need to follow this 5 main steps to install it:

1. [Enabling objects](https://portal.nutanix.com/page/documents/details?targetId=Objects-v5_0:top-enable-objects-t.html)
2. [Create an Object Store](https://portal.nutanix.com/page/documents/details?targetId=Objects-v5_0:top-object-store-deployment-t.html)
3. [Create an S3 Bucket](https://portal.nutanix.com/page/documents/details?targetId=Objects-v5_0:top-create-configure-buckets-t.html#ntask_cxx_m2t_ggb)
4. [Generate an Access Key](https://portal.nutanix.com/page/documents/details?targetId=Objects-v5_0:top-generate-access-key-t.html)
5. [Set Full Access Permission for the Access Key on the Bucket](https://portal.nutanix.com/page/documents/details?targetId=Objects-v5_0:top-edit-user-access.html)

# 

### Velero installation

Make sure Velero CLI is installed on the desktop

#### **Deploy Velero on NKE cluster**

From NKE, retrieve and install locally your Kubeconfig

<https://portal.nutanix.com/page/documents/details?targetId=Nutanix-Kubernetes-Engine-v2_10:top-download-kubeconfig-t.html>

Follow the below steps in your shell to install Velero:

1. Set environment variables

Take care to configure correctly the following value in the below command

* `S3\_URL` with your correct s3 target
* `YOUR-BUCKET-NAME` with your exact bucket name
* `S3\_ACCESS\_KEY` and `S3\_ACCESS\_SECRET` with your bucket access information

S3\_BUCKET=YOUR-BUCKET-NAME

S3\_URL=<https://my.s3.url>.or.ip.address

S3\_REGION=us-east-1

S3\_ACCESS\_KEY=""

S3\_ACCESS\_SECRET=""



1. Create credential file

cat <<EOF > object-access

[default]

aws\_access\_key\_id=${S3\_ACCESS\_KEY}

aws\_secret\_access\_key=${S3\_ACCESS\_SECRET}

EOF



1. Install Velero

Set SOURCE\_KUBECONFIG environment variable to point to the NKE (Source) Cluster

export SOURCE\_KUBECONFIG=<path-to-nke-kubeconfig>





kubectl create namespace kommander

velero install --kubeconfig=${SOURCE\_KUBECONFIG} --provider aws --plugins velero/velero-plugin-for-aws:v1.9.2,velero/velero-plugin-for-csi:v0.7.1 --bucket ${S3\_BUCKET} --backup-location-config region=${S3\_REGION},insecureSkipTLSVerify="true",s3ForcePathStyle="true",s3Url=${S3\_URL},checksumAlgorithm='' --snapshot-location-config region=${S3\_REGION},insecureSkipTLSVerify="true",s3ForcePathStyle="true",s3Url=${S3\_URL},checksumAlgorithm='' --secret-file object-access --features=EnableCSI --use-node-agent -n kommander --node-agent-pod-cpu-limit 0 --node-agent-pod-mem-limit 0



insecureSkipTLSVerify is set to true to avoid any cert issues, it is strongly recommended to configure a correctly signed certificate on your S3 target and to pass this option to false.

When you are installing Velero on NKE, you need to patch the node-agent daemon-set to define the correct NKE kubelet path, the node-agent pod will crash until the patch is applied :

# This command need to be run on NKE only not on NKP

kubectl patch -n kommander ds node-agent -p '{"spec":{"template":{"spec":{"volumes":[{"hostPath":{"path":"/var/nutanix/var/lib/kubelet/pods","type":""},"name":"host-pods"}]}}}}'



You can verify if Velero is correctly linked to your s3 repo with

velero get backup-locations

NAME PROVIDER BUCKET/PREFIX PHASE LAST VALIDATED ACCESS MODE DEFAULT

default aws YOUR-BUCKET-NAME Available 2024-06-24 18:58:13 +0200 CEST ReadWrite true



### 

#### Deploy or Reconfigure Velero on NKP cluster

There are two cases here depending of you NKP license level:

* If you are using starter you need also to install Velero by yourself using the same NKE method above
* If you are using Pro or Ultimate you can use the Velero available in the apps catalog by modifying the configuration as explained below

##### On NKP starter cluster

* Set TARGET\_KUBECONFIG environment variable to point to the NKP (Target) Cluster

export TARGET\_KUBECONFIG=<path-to-nkp-kubeconfig>



* Launch the following command in your shell to install Velero:

1. Set environment variables

Take care to configure correctly the following value in the below command

* `S3\_URL` with your correct s3 target
* `YOUR-BUCKET-NAME` with your exact bucket name
* `S3\_ACCESS\_KEY` and `S3\_ACCESS\_SECRET` with your bucket access information

S3\_BUCKET=YOUR-BUCKET-NAME

S3\_URL=<https://my.s3.url>

S3\_REGION=us-east-1

S3\_ACCESS\_KEY=""

S3\_ACCESS\_SECRET=""



1. Create credential file

cat <<EOF > object-access

[default]

aws\_access\_key\_id=${S3\_ACCESS\_KEY}

aws\_secret\_access\_key=${S3\_ACCESS\_SECRET}

EOF



1. Install Velero



velero install --kubeconfig=${TARGET\_KUBECONFIG} --provider aws --plugins velero/velero-plugin-for-aws:v1.9.2,velero/velero-plugin-for-csi:v0.7.1 --bucket ${S3\_BUCKET} --backup-location-config region=${S3\_REGION},insecureSkipTLSVerify="true",s3ForcePathStyle="true",s3Url=${S3\_URL},checksumAlgorithm='' --snapshot-location-config region=${S3\_REGION},insecureSkipTLSVerify="true",s3ForcePathStyle="true",s3Url=${S3\_URL},checksumAlgorithm='' --secret-file object-access --features=EnableCSI --use-node-agent -n kommander --node-agent-pod-cpu-limit 0 --node-agent-pod-mem-limit 0



You can verify if Velero is correctly link to your s3 repo with

velero get backup-locations

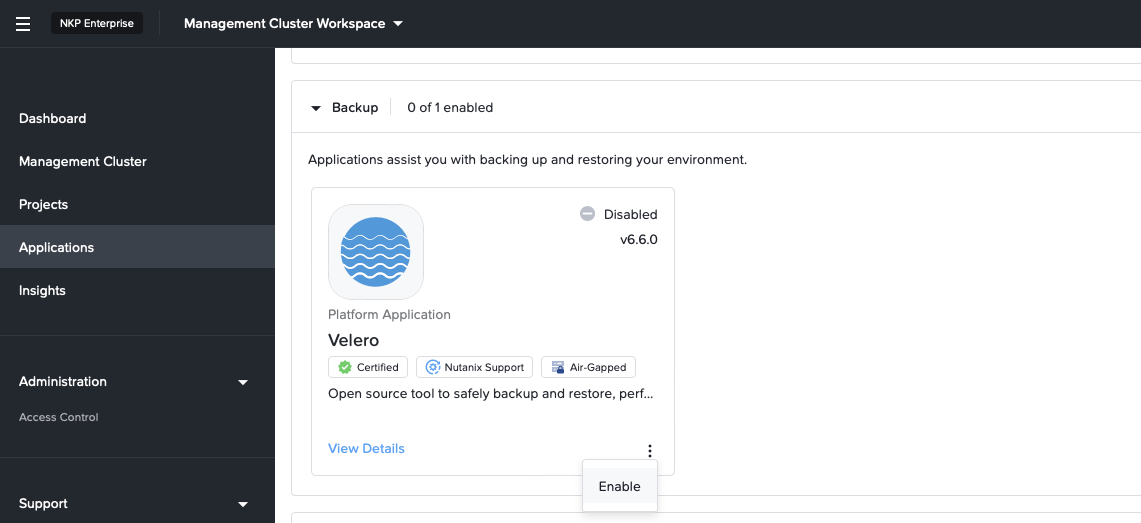
NAME PROVIDER BUCKET/PREFIX PHASE LAST VALIDATED ACCESS MODE DEFAULT

default aws S3\_BUCKET Available 2024-06-24 18:58:13 +0200 CEST ReadWrite true

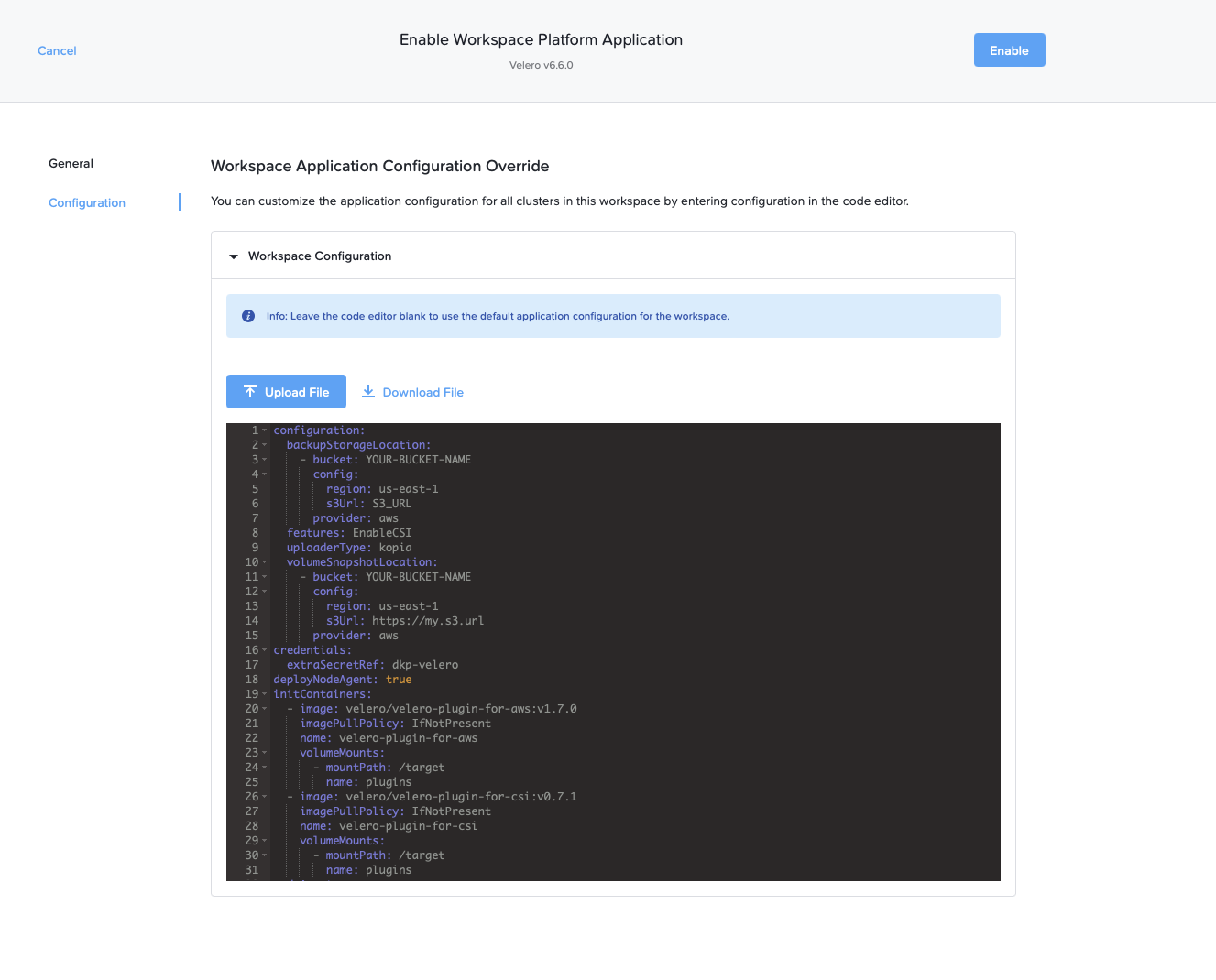


##### On NKP Pro/Ultimate cluster

With NKP Pro/Ultimate license you can install Velero directly from the Apps catalog



During Velero installation you need to modify the existing Velero config



The change to apply are detailed below

Take care to configure correctly the following value in the below command

* `S3\_URL` with your correct s3 target
* `YOUR-BUCKET-NAME` with your exact bucket name
* `S3\_ACCESS\_KEY` and `S3\_ACCESS\_SECRET` with your bucket access information

configuration:

backupStorageLocation:

- bucket: YOUR-BUCKET-NAME

config:

region: us-east-1

s3Url: S3\_URL

s3ForcePathStyle: true

insecureSkipTLSVerify: true

profile: default

provider: aws

credential:

key: aws

name: dkp-velero

features: EnableCSI

uploaderType: kopia

volumeSnapshotLocation:

- bucket: YOUR-BUCKET-NAME

config:

region: us-east-1

s3Url: https://my.s3.url

provider: aws

deployNodeAgent: true

initContainers:

- image: velero/velero-plugin-for-aws:v1.7.0

imagePullPolicy: IfNotPresent

name: velero-plugin-for-aws

volumeMounts:

- mountPath: /target

name: plugins

- image: velero/velero-plugin-for-csi:v0.7.1

imagePullPolicy: IfNotPresent

name: velero-plugin-for-csi

volumeMounts:

- mountPath: /target

name: plugins

nodeAgent:

annotations:

secret.reloader.stakater.com/reload: dkp-velero

priorityClassName: dkp-critical-priority

resources:

limits: null



To finish you need to create the corresponding secret with the access and private key to use your bucket

S3\_ACCESS\_KEY=""

S3\_ACCESS\_SECRET=""

kubectl apply -f - <<EOF

apiVersion: v1

kind: Secret

metadata:

name: dkp-velero

namespace: kommander

type: Opaque

stringData:

aws: |

[default]

aws\_access\_key\_id = ${S3\_ACCESS\_KEY}

aws\_secret\_access\_key = ${S3\_ACCESS\_SECRET}

EOF



## Migrate Stateless applications

This is the simple approach to migrate a stateless application. As mentioned earlier, we will be using nginx as a sample stateless application to demonstrate the end to end migration process and it is not really required for the migration itself. Once comfortable with the process, replace nginx with your own stateless application and follow the same migration steps for it.

### Deploy Stateless Application to the NKE (Source) Cluster

On the source NKE cluster

Set SOURCE\_KUBECONFIG environment variable to point to the NKE (Source) Cluster

export SOURCE\_KUBECONFIG=<path-to-nke-kubeconfig>



kubectl create ns app-stateless --kubeconfig ${SOURCE\_KUBECONFIG}

kubectl create deployment webserver --image=nginx -n app-stateless --kubeconfig ${SOURCE\_KUBECONFIG}

kubectl expose deployment webserver --port=80 --type=ClusterIP -n app-stateless --kubeconfig ${SOURCE\_KUBECONFIG}



In this case a basic stateless webserver (nginx) is used as the sample stateless application.

### Backup Stateless Application on NKE Cluster

On the source NKE cluster, set SOURCE\_KUBECONFIG environment variable to point to the NKE (Source) Cluster

export SOURCE\_KUBECONFIG=<path-to-nke-kubeconfig>



velero backup create app-stateless-backup --kubeconfig ${SOURCE\_KUBECONFIG} --include-namespaces app-stateless



You can verify backup is successful with

velero backup get app-stateless-backup --kubeconfig ${SOURCE\_KUBECONFIG}

NAME STATUS ERRORS WARNINGS CREATED EXPIRES STORAGE LOCATION SELECTOR

app-stateless-backup Completed 0 0 2024-06-24 19:25:36 +0200 CEST 29d default <none>



In case of issue you can delete your backup with the following command

velero backup delete app-stateless-backup --kubeconfig ${SOURCE\_KUBECONFIG}



In case backup is not successful you can explore <https://velero.io/docs/v1.13/troubleshooting/>

### Migrate Stateless Application to NKP Cluster

On the target NKP cluster

Set TARGET\_KUBECONFIG environment variable to point to the NKP (Target) Cluster

export TARGET\_KUBECONFIG=<path-to-nkp-kubeconfig>



velero restore create --from-backup app-stateless-backup --kubeconfig ${TARGET\_KUBECONFIG}



### You can verify migrate is successful and test if your application is running as expected.

velero get restore --kubeconfig ${TARGET\_KUBECONFIG}

NAME BACKUP STATUS STARTED COMPLETED ERRORS WARNINGS CREATED SELECTOR

app-stateless-backup-20240624193132 app-stateless-backup Completed 2024-06-24 19:31:32 +0200 CEST 2024-06-24 19:33:04 +0200 CEST 0 1 2024-06-24 19:31:32 +0200 CEST <none>



In case restore is not successful you can check https://velero.io/docs/v1.13/debugging-restores/

## 

## Migrate Nutanix Volumes based applications

Below are the steps to migrate a stateful application that is using Nutanix volumes PV. As mentioned earlier, we will be using Wordpress as a sample stateful application to demonstrate the end to end migration process and it is not really required for the migration itself. Once comfortable with the process, replace Wordpress with your own Stateful application and follow the same migration process for it.

### Deploy Volumes based Application to the NKE (Source) Cluster

Let’s deploy a wordpress as a sample application.

On the source NKE cluster

Set SOURCE\_KUBECONFIG environment variable to point to the NKE (Source) Cluster

export SOURCE\_KUBECONFIG=<path-to-nke-kubeconfig>



helm install wordpress oci://registry-1.docker.io/bitnamicharts/wordpress --create-namespace -n app-volume --set global.storageClass=nutanix-volume --kubeconfig ${SOURCE\_KUBECONFIG}



Here we are deploying a wordpress apps in the namespace `app-volume` and we create the persistent volumes in a custom created storage class nutanix-volume

### Backup Volumes based Application on NKE Cluster

On the source NKE cluster

| Warning |
| --- |
| In order to guarantee a consistent backup of the application, we strongly advise you to stop it first. You can do it by setting every replica to 0 before launching the backup.  Be sure to note the number of initial replicas so that you can reconfigure them once the restore command is run and the Workload resources (e.g. Deployment/StatefulSet etc. are created). Note: The PVs will not be created if there is no pod running due to StorageClass’ VolumeBindMode being “WaitForFirstConsumer”. As a result, the restore operation will fail to restore the volume unless the replicas for the Workload resource are set to at least 1.   Example with the wordpress above  kubectl scale --replicas=0 deploy/wordpress -n app-volume kubectl scale --replicas=0 sts/wordpress-mariadb -n app-volume |

Set SOURCE\_KUBECONFIG environment variable to point to the NKE (Source) Cluster

export SOURCE\_KUBECONFIG=<path-to-nke-kubeconfig>



Create a volumesnapshotclass in your NKE (Source) Cluster



#!/usr/bin/bash

SECRET=$(kubectl get sc -o=jsonpath='{.items[?(@.metadata.annotations.storageclass\.kubernetes\.io\/is-default-class=="true")].parameters.csi\.storage\.k8s\.io\/provisioner-secret-name}')

DRIVER=$(kubectl get sc -o=jsonpath='{.items[?(@.metadata.annotations.storageclass\.kubernetes\.io\/is-default-class=="true")].provisioner}')

cat << EOF | kubectl apply -f -

apiVersion: snapshot.storage.k8s.io/v1

kind: VolumeSnapshotClass

metadata:

name: default-snapshotclass

driver: $DRIVER

parameters:

storageType: NutanixVolumes

csi.storage.k8s.io/snapshotter-secret-name: $SECRET

csi.storage.k8s.io/snapshotter-secret-namespace: kube-system

deletionPolicy: Delete

EOF



Create a backup called `wordpress` which contains everything from the app-volume namespace

velero backup create wordpress --include-namespaces app-volume --snapshot-move-data --kubeconfig ${SOURCE\_KUBECONFIG}



If needed you can backup multiple namespaces in same time with --include-namespaces ns1,ns2,ns3

Ensure Backup is successful

The backup should complete successfully with no errors. Run the

velero --kubeconfig ${SOURCE\_KUBECONFIG} backup get wordpress



In case backup is not successful you can check <https://velero.io/docs/v1.13/troubleshooting/>

### Migrate Volumes based Application to NKP Cluster

On the target NKP cluster

Set TARGET\_KUBECONFIG environment variable to point to the NKP (Target) Cluster

export TARGET\_KUBECONFIG=<path-to-nkp-kubeconfig>



Ask to restore the wordpress backup

velero restore create --from-backup wordpress --kubeconfig ${TARGET\_KUBECONFIG}



After initiating the restore command set the replicas for the corresponding back to either the original count or at least 1 (As mentioned earlier, this is important as else the PVs won’t be created due to the StorageClass’ WaitForFirstConsumer VolumeBindMode)

kubectl scale deploy -n app-volume wordpress --replicas 1 --kubeconfig ${TARGET\_KUBECONFIG}

kubectl scale sts -n app-volume wordpress-mariadb --replicas 1 --kubeconfig ${TARGET\_KUBECONFIG}



Ensure Restore is successful

The restore should complete successfully with no errors. Run the following command to check the status.

velero --kubeconfig ${SOURCE\_KUBECONFIG} restore get wordpress



Another useful command to check the current status is

velero --kubeconfig ${SOURCE\_KUBECONFIG} restore describe wordpress --insecure-skip-tls-verify --details



In case restore is not successful you can check <https://velero.io/docs/v1.13/debugging-restores/>

## Migrate Nutanix files (dynamic or static) based applications

| Warning |
| --- |
| The lack of full Snapshot support with Nutanix CSI Files forces us to use Velero File System Backup (FSB) without snapshot. FSB backs up data by copying files from the live file system from a running pod that has mounted the persistent volume. As a result the entire data is not captured at the same point in time, so files can be inconsistent if there are changes during the backup process. Since the pod should have the volume mounted to take the backup, it can’t be stopped (i.e. can’t scale the corresponding Deployment or Statefulset to zero) while taking the backup. To maintain data integrity, it is highly recommended to pause any activities on the application during the backup process. |

### 

**Note:** Although Wordpress and MariaDB might not be the most ideal applications to use NFS backed volumes, we have used these to demonstrate the procedure to keep things simple. As mentioned earlier, we are using Wordpress as a sample stateless application to demonstrate the end to end migration process and it is not really required for the migration itself. Once comfortable with the process, replace Wordpress with your own Stateful application and follow the same migration process for it.

### Deploy Wordpress app to the NKE (Source) Cluster

Step 1: Set SOURCE\_KUBECONFIG environment variable to point to the NKE (Source) Cluster

export SOURCE\_KUBECONFIG=<path-to-nke-kubeconfig>



Step 2: (Optional) If not already defined, create NutanixFiles (dynamic) StorageClass resource.



SECRET\_NAME=<ntnx-files-secret>

SECRET\_NAMESPACE=kube-system

SC\_NAME=ntnx-dfs

SQUASH\_TYPE=none

PE\_HOST=<prism-element-host-name-or-ip>

PE\_USER=<prism-element-user>

PE\_PASSWD=<prism-element-password>

FILES\_SERVER=<name-of-nfs-server> # Note this is the resource name and not the hostname

# Create secret with PE connection details for the StorageClass

kubectl --kubeconfig=${TARGET\_KUBECONFIG} apply -f - <<EOF

apiVersion: v1

kind: Secret

metadata:

name: ${SECRET\_NAME}

namespace: kube-system

stringData:

key: "${PE\_HOST}:9440:${PE\_USER}:${PE\_PASSWD}"

EOF

# Create StorageClasss

kubectl --kubeconfig ${TARGET\_KUBECONFIG} apply -f - <<EOF

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: ${SC\_NAME}

provisioner: csi.nutanix.com

parameters:

csi.storage.k8s.io/node-publish-secret-name: ${SECRET\_NAME}

csi.storage.k8s.io/node-publish-secret-namespace: ${SECRET\_NAMESPACE}

csi.storage.k8s.io/controller-expand-secret-name: ${SECRET\_NAME}

csi.storage.k8s.io/controller-expand-secret-namespace: ${SECRET\_NAMESPACE}

dynamicProv: ENABLED

nfsServerName: ${FILES\_SERVER}

csi.storage.k8s.io/provisioner-secret-name: ${SECRET\_NAME}

csi.storage.k8s.io/provisioner-secret-namespace: ${SECRET\_NAMESPACE}

storageType: NutanixFiles

squashType: ${SQUASH\_TYPE}

reclaimPolicy: Delete

volumeBindingMode: Immediate

allowVolumeExpansion: true

EOF

export SC\_NAME\_VOL=ntnx-vol

export STORAGE\_CONTAINER=<name-of-prism-storage-container>

export FS\_TYPE=ext4 # ext4 or xfs

# Create StorageClasss for NutanixVolumes

kubectl --kubeconfig ${TARGET\_KUBECONFIG} apply -f - <<EOF

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: ${SC\_NAME\_VOL}

provisioner: csi.nutanix.com

parameters:

csi.storage.k8s.io/node-publish-secret-name: ${SECRET\_NAME}

csi.storage.k8s.io/node-publish-secret-namespace: ${SECRET\_NAMESPACE}

csi.storage.k8s.io/controller-expand-secret-name: ${SECRET\_NAME}

csi.storage.k8s.io/controller-expand-secret-namespace: ${SECRET\_NAMESPACE}

csi.storage.k8s.io/provisioner-secret-name: ${SECRET\_NAME}

csi.storage.k8s.io/provisioner-secret-namespace: ${SECRET\_NAMESPACE}

storageContainer: ${STORAGE\_CONTAINER}

csi.storage.k8s.io/fstype: ${FS\_TYPE}

description: nutanix-volume

storageType: NutanixVolumes

reclaimPolicy: Delete

volumeBindingMode: Immediate

allowVolumeExpansion: true

EOF



Step 3: Deploy Wordpress using Helm

Ensure SC\_NAME environment variable is set to a NutanixFiles (Dynamic) (either created in Step 2 above or already present in the environment).

Note: This will take approximately 2 minutes to complete. Optimally run without the “--wait” flag and manually watch the pods to ensure they are up.

export SC\_NAME=ntnx-dfs

export APP\_NAMESPACE=wordpressdemo

export RELEASE\_NAME=mywordpress



helm --kubeconfig=${SOURCE\_KUBECONFIG} upgrade --install ${RELEASE\_NAME} oci://registry-1.docker.io/bitnamicharts/wordpress --set global.storageClass=${SC\_NAME} --set service.type=NodePort -n ${APP\_NAMESPACE} --create-namespace --version 22.4.16 --wait



Step 4: Ensure both pods are healthy

kubectl --kubeconfig=${SOURCE\_KUBECONFIG} get po -n ${APP\_NAMESPACE}



### Backup Application on NKE Cluster

Step 1: Set SOURCE\_KUBECONFIG environment variable to point to the NKE (Source) Cluster

SOURCE\_KUBECONFIG=<path-to-nke-kubeconfig>



Step 2: Take a backup pointing to the BackupStorageLocation created in the last step

APP\_NAMESPACE=wordpressdemo

BACKUP\_NAME=fsb-wordpress

BSL\_NAME=default # Use the BackupStorageLocation created in last step

velero --kubeconfig ${SOURCE\_KUBECONFIG} backup create ${BACKUP\_NAME} --include-namespaces=${APP\_NAMESPACE} --snapshot-volumes=false --storage-location ${BSL\_NAME} --default-volumes-to-fs-backup -w



Note: This will also backup “emptyDir” volumes. If that is not desired then omit the flag to backup selected volumes only by annotating them as shown below.

Note: Omit the “--default-volumes-to-fs-backup” flag in the above command if only selected volumes in the target pods are going to be backed up using FileSystemBackup (FSB) by selectively annotating each pod.

APP\_NAMESPACE=wordpressdemo

RELEASE\_NAME=mywordpress

kubectl --kubeconfig ${SOURCE\_KUBECONFIG} annotate pod -n ${APP\_NAMESPACE} ${RELEASE\_NAME}-mariadb-0 [backup.velero.io/backup-volumes=](http://backup.velero.io/backup-volumes=mariadb-pv)data

kubectl --kubeconfig ${SOURCE\_KUBECONFIG} annotate pod -n ${APP\_NAMESPACE} -lapp.kubernetes.io/name=wordpress backup.velero.io/backup-volumes=wordpress-data



Step 3: Ensure Backup is successful

The backup should complete successfully with no errors. Run the

velero --kubeconfig ${SOURCE\_KUBECONFIG} backup get ${BACKUP\_NAME}

kubectl --kubeconfig ${SOURCE\_KUBECONFIG} get podvolumebackups --namespace=${VELERO\_NAMESPACE}



In case backup is not successful you can explore <https://velero.io/docs/v1.13/troubleshooting/>

### Migrate Application to NKP Cluster

Step 1: Set TARGET\_KUBECONFIG environment variable to point to the NKP (Target) Cluster

TARGET\_KUBECONFIG=<path-to-nkp-kubeconfig>



Step 2 (optional) : If the NutanixFiles (Dynamic) StorageClass with the respective name matching in the Source NKE cluster is not present then create it. Ensure that the Files instance has a FQDN set and is resolvable, else PVs can’t be mounted on the nodes.

Note:

* Optionally if the names don’t match or a different StorageClass is being used in the target cluster then create a “change-storage-class-config” ConfigMap to map the StorageClass between the two clusters. More details can be found [here](https://velero.io/docs/v1.3.0/restore-reference/) at the end of the page.
* “files-key:” field in the Storage Class secret is only needed for Nutanix CSI 3.1+ (i.e. NKP 2.13.0+)

SECRET\_NAME=<ntnx-files-secret>

SECRET\_NAMESPACE=kube-system

SC\_NAME=ntnx-dfs

SQUASH\_TYPE=none

PE\_HOST=<prism-element-host-name-or-ip>

PE\_USER=<prism-element-user>

PE\_PASSWD=<prism-element-password>

FILES\_SERVER=<name-of-nfs-server> # Note this is the resource name and not the hostname

FILES\_SERVER=<name-of-nfs-server> # Note this is the resource name and not the hostname

FILES\_FQDN=<fqdn-of-files-server> # e.g. fileserver01.sample.com

FILES\_REST\_USER=<rest-api-user> # e.g. csi

FILES\_REST\_PASSWD=<rest-api-password> # e.g. Nutanix.123

# Create secret with PE connection details for the StorageClass

kubectl --kubeconfig=${TARGET\_KUBECONFIG} apply -f - <<EOF

apiVersion: v1

kind: Secret

metadata:

name: ${SECRET\_NAME}

namespace: kube-system

stringData:

key: "${PE\_HOST}:9440:${PE\_USER}:${PE\_PASSWD}"

files-key: "${FILES\_FQDN}:${FILES\_REST\_USER}:${FILES\_REST\_PASSWD}"

EOF

# Create StorageClasss

kubectl --kubeconfig ${TARGET\_KUBECONFIG} apply -f - <<EOF

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: ${SC\_NAME}

provisioner: csi.nutanix.com

parameters:

csi.storage.k8s.io/node-publish-secret-name: ${SECRET\_NAME}

csi.storage.k8s.io/node-publish-secret-namespace: ${SECRET\_NAMESPACE}

csi.storage.k8s.io/controller-expand-secret-name: ${SECRET\_NAME}

csi.storage.k8s.io/controller-expand-secret-namespace: ${SECRET\_NAMESPACE}

dynamicProv: ENABLED

nfsServerName: ${FILES\_SERVER}

csi.storage.k8s.io/provisioner-secret-name: ${SECRET\_NAME}

csi.storage.k8s.io/provisioner-secret-namespace: ${SECRET\_NAMESPACE}

storageType: NutanixFiles

squashType: ${SQUASH\_TYPE}

reclaimPolicy: Delete

volumeBindingMode: Immediate

allowVolumeExpansion: true

EOF

export SC\_NAME\_VOL=ntnx-vol

export STORAGE\_CONTAINER=<name-of-prism-storage-container>

export FS\_TYPE=ext4 # ext4 or xfs

# Create StorageClasss for NutanixVolumes

kubectl --kubeconfig ${TARGET\_KUBECONFIG} apply -f - <<EOF

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: ${SC\_NAME\_VOL}

provisioner: csi.nutanix.com

parameters:

csi.storage.k8s.io/node-publish-secret-name: ${SECRET\_NAME}

csi.storage.k8s.io/node-publish-secret-namespace: ${SECRET\_NAMESPACE}

csi.storage.k8s.io/controller-expand-secret-name: ${SECRET\_NAME}

csi.storage.k8s.io/controller-expand-secret-namespace: ${SECRET\_NAMESPACE}

csi.storage.k8s.io/provisioner-secret-name: ${SECRET\_NAME}

csi.storage.k8s.io/provisioner-secret-namespace: ${SECRET\_NAMESPACE}

storageContainer: ${STORAGE\_CONTAINER}

csi.storage.k8s.io/fstype: ${FS\_TYPE}

description: nutanix-volume

storageType: NutanixVolumes

reclaimPolicy: Delete

volumeBindingMode: Immediate

allowVolumeExpansion: true

EOF



Step 3: Migrate Application

Migrate the application by restoring the backup that was taken from the NKE (Source) Cluster.



BACKUP\_NAME=fsb-wordpress

velero --kubeconfig ${TARGET\_KUBECONFIG} restore create --from-backup ${BACKUP\_NAME} -w



Step 4: Ensure Restore is successful

The restore should complete successfully with no errors. Run the

velero --kubeconfig ${SOURCE\_KUBECONFIG} restore get ${BACKUP\_NAME}



Another useful command to check the current status is

velero --kubeconfig ${SOURCE\_KUBECONFIG} restore describe wordpress --insecure-skip-tls-verify --details



In case restore is not successful you can check <https://velero.io/docs/v1.13/debugging-restores/>

Step 5: Ensure both pods are healthy

kubectl --kubeconfig=${TARGET\_KUBECONFIG} get po -n ${APP\_NAMESPACE}



Step 6: Login to the Wordpress app using a web browser and verify the changes

Get the url and connection details of the migrated applications by running the following commands and then paste it in the browser to login.

WP\_URL=$(echo http://$(kubectl --kubeconfig=${TARGET\_KUBECONFIG} get no -o wide --no-headers| awk '{print $6}' | head -1):$(kubectl --kubeconfig=${TARGET\_KUBECONFIG} get svc -n wordpressdemo mywordpress -o jsonpath='{.spec.ports[?(@.name == "http")].nodePort}')/login)

WP\_PWD=$(kubectl --kubeconfig=${TARGET\_KUBECONFIG} get secret --namespace wordpressdemo mywordpress -o jsonpath="{.data.wordpress-password}" | base64 -d)

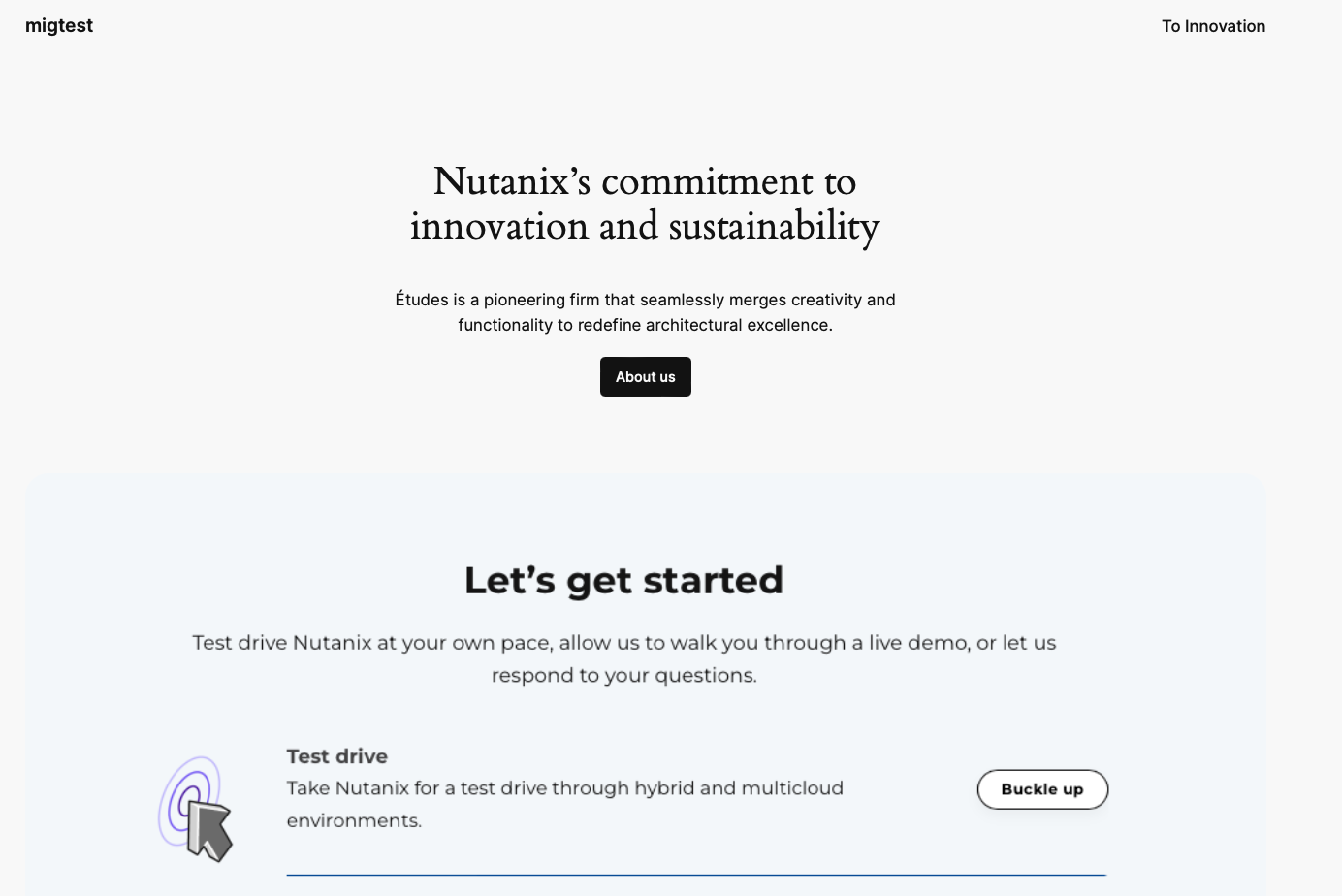
echo WordPress Admin URL: $WP\_URL

echo Username: user

echo Password: $WP\_PWD



Wordpress should open with all the customizations intact. Try making changes to ensure that the database and filesystem connectivity is good and that the application has been migrated successfully.



At this point the migration is complete and the “load balancer/DNS” can be updated to point to the migrated instance to completely cutover.

## Velero post-migration cleaning step

Once the migration is complete, you can choose to uninstall Velero from the NKP cluster if you no longer plan to use it in the future.

Installation is straightforward, just follow the steps below:

* Set TARGET\_KUBECONFIG environment variable to point to the NKP (Target) Cluster

export TARGET\_KUBECONFIG=<path-to-nkp-kubeconfig>



* Launch the following command in your shell to uninstall Velero:

velero uninstall --kubeconfig=${TARGET\_KUBECONFIG} -n kommander



# Deprecated Kubernetes API Migration

As the Kubernetes API evolves, APIs are periodically reorganized or upgraded. When APIs evolve, the old API is deprecated and eventually removed.

<https://kubernetes.io/docs/reference/using-api/deprecation-guide/>

If you try to migrate an application that is using an old API, you may be unable to restart on the target NKP cluster.

You need to carefully analyze your existing resources in your NKE cluster to be sure they are compatible with your NKP target cluster.

There is a tool who can help your to make this analysis: [Pluto](https://github.com/FairwindsOps/pluto?tab=readme-ov-file)

This is pretty simple to use:

Install Pluto by following the documentation [here](https://pluto.docs.fairwinds.com/installation/)

Set SOURCE\_KUBECONFIG environment variable to point to the NKE (Source) Cluster

export SOURCE\_KUBECONFIG=<path-to-nke-kubeconfig>



Launch the following command to check the content of your source cluster

pluto detect-all-in-cluster -o wide -t k8s=v1.29.6 --kubeconfig ${SOURCE\_KUBECONFIG}



You will get a list of ressources to modify before the upgrade

*Example below*

NAME NAMESPACE KIND VERSION REPLACEMENT DEPRECATED DEPRECATED IN REMOVED REMOVED IN REPL AVAIL REPL AVAIL IN

example-ingress default Ingress networking.k8s.io/v1beta1 networking.k8s.io/v1 true v1.19.0 true v1.22.0 true v1.19.0

example-hpa default HorizontalPodAutoscaler autoscaling/v2beta2 autoscaling/v2 true v1.23.0 true v1.26.0 false

example-cronjob default CronJob batch/v1beta1 batch/v1 true v1.21.0 true v1.25.0 true v1.21.0

my-app-csr <UNKNOWN> CertificateSigningRequest certificates.k8s.io/v1beta1 certificates.k8s.io/v1 true v1.19.0 true v1.22.0 true v1.19.0

example-ingress default Ingress networking.k8s.io/v1beta1 networking.k8s.io/v1 true v1.19.0 true v1.22.0 true v1.19.0

example-pdb default PodDisruptionBudget policy/v1beta1 policy/v1 true v1.21.0 true v1.25.0 true v1.21.0

# Basic Troubleshooting

# For general velero troubleshooting, here is a link to official velero troubleshooting documentation

<https://velero.io/docs/v1.13/troubleshooting/>

Here are some additional troubleshooting steps that might be useful:

## Partially Failed Backup

A good place to start troubleshooting a partially failed backup is to describe the backup resource itself using the velero cli

E.g.



BACKUP\_NAME=mybackup

VELERO\_NAMESPACE=kommander

velero backup describe ${BACKUP\_NAME} -n ${VELERO\_NAMESPACE} --insecure-skip-tls-verify



Look at the response carefully to spot any error messages

## BackupStorageLocaction (BSL) not in “Available” Phase/State

For the Backups and restore to work, the BSL being in “Available” Phase is key for both the source cluster and the target cluster. These commands should be run against both source and target clusters by setting kubeconfig accordingly.

E.g.

kubectl get bsl -A

NAMESPACE NAME PHASE LAST VALIDATED AGE DEFAULT

kommander default Available 8s 8s true



If for some reason that is not true then check the following

1. Ensure velero pods are running and healthy

kubectl get po -A | grep -i velero



1. Check for any message in the status of the BSL resource. The message field is added and populated with a useful error message if the BSL is not ready for some reason.

Here are a few examples:

E.g.:

kubectl get bsl -n kommander default -o jsonpath='{.status.message}'

####################  
## Sample output if the user tied to the Access Key does not have read/write permission to the Object Store Bucket being used  
####################

BackupStorageLocation "default" is unavailable: rpc error: code = Unknown desc = AccessDenied: Access Denied

status code: 403, request id: 703fa3c7-80c4-4f57-7d57-6e85de340e7d+310000+137647+2158050+465434, host id: 703fa3c7-80c4-4f57-7d57-6e85de340e7d+3100

######################

## Sample output if the object store url being used is not reachable from the cluster

BackupStorageLocation "default" is unavailable: rpc error: code = Unknown desc = RequestError: send request failed

caused by: Get "https://10.122.7.100/migtest?delimiter=%2F&list-type=2&prefix=": dial tcp 10.122.7.100:443: i/o timeout

######################



Note: If the BSL is healthy there is no message field in the status section and hence the above commands will not return any data.

## Troubleshooting for FSB

If using FSB (FileSystemBackups) for backup and restore and the backup restore finishes with a PartiallyFailed status, then here are some specific troubleshooting steps:

1. Ensure that node-agent pods are running and healthy on both source and target cluster

kubectl get po -A | grep -i node-agent



1. Ensure PodVolumeBackup resources were created for backups (on the Source NKE cluster) and PodVolumeRestoreresources were created for restores (on the Target NKP cluster)

kubectl get podvolumebackup -A # Run this on Source NKE cluster

kubectl get podvolumerestore -A # Run this on Target NKP cluster



## Troubleshooting Restore

To check the progress of the restore or to troubleshoot a restore, use the following command

Note: Part of the output will display a x509 certificate error without the “--insecure-skip-tls-verify” flag. This does not mean that the restore has failed, it just means the Velero command was not able to gather all the details from as it was not able to verify the authenticity of the Object Store.

velero --kubeconfig ${SOURCE\_KUBECONFIG} restore describe wordpress --insecure-skip-tls-verify --details



# 

# References

1. [Velero Documentation](https://velero.io/docs/v1.7/)
2. [D2iQ Kubernetes Platform Documentation](https://docs.d2iq.com/dkp/2.8/)
3. [Deprecated API Migration Guide](https://kubernetes.io/docs/reference/using-api/deprecation-guide/)
4. [Pluto: Find Kubernetes resources that have been deprecated](https://pluto.docs.fairwinds.com/)