Dremio Software

Migrate a Dremio Standalone Cluster to Kubernetes

Introduction

The document describes how to migrate a Dremio standalone deployment to Kubernetes. A standalone deployment means that Dremio is deployed on virtual machines (VMs).

⚠ NOTICE
For simplification, the document does not cover any continuous integration/continuous delivery (CI/CD) implementation. It assumes you are installing from a local or remote terminal.

The document covers a high-level migration to Kubernetes. It does not cover any monitoring migration and is not intended to be a deep dive into cloud vendors' different Kubernetes distributions.

Why Deploy Dremio on Kubernetes?

There are several advantages of deploying Dremio on Kubernetes.

No failover coordinator is required for high availability (HA)

Kubernetes deployments do not require a failover coordinator for high availability (HA). If a pod or node fails, a new pod is deployed, and the persistent disk is automatically attached to the new pod. This reduces the complexity and cost of the entire deployment.

The hardware saved by not requiring a secondary node for HA can be used to host an additional executor or a scale-out coordinator.

If a fail-over coordinator fails, the entire system is down, so Kubernetes deployments tend to be robust. Dremio deployed on Kubernetes should survive multiple node failures as long as new and healthy nodes are added to the cluster's node pools.

No network file system (NFS) required

Kubernetes deployments do not require a network file system (NFS). Kubernetes uses Persistent Volume Claims (PVCs) to manage disks. If a pod or node fails, a new pod is deployed, and the persistent disk is automatically attached to the new pod. NFS would require support for locking. The Azure NFS or NetApp NFS offering is more costly than regular premium disks.

In addition, NFS throughput correlates with the requested size, so users typically need a larger and more expensive NFS disk. NFS could also be a source of error if mounted incorrectly or if other pods access it and put load on the disks or modify existing files.

No interaction is required in case of high-availability failover

In Kubernetes deployments, a failover for high availability typically requires no interaction with the cluster. If a node fails, cloud vendors should provide a new node to the cluster's node pool, and the pod should be automatically re-deployed on the new node. The system is self-healing. When a coordinator fails while using VMs, the coordinator must be repaired manually.

Complete coverage of support

When using Azure Kubernetes or AWS Kubernetes together with Dremio's Kubernetes deployment, all involved components should be covered by Support. Since AKS and EKS are managed services, customers should get complete Microsoft Azure or AWS support. In addition, the fully built Docker image for Dremio includes Dremio and the operating system base image, which Dremio fully supports.

When using VMs, customers need to maintain the operating systems as well. This means keeping them up-to-date, applying security fixes, and managing the entire configuration. Depending on the Linux distribution, a customer may not get support for the operating system (e.g. CentOS, Ubuntu) and may need to license support separately from Red Hat or Canonical. Customers may not require separate support when using AKS or EKS and Dremio's pre-built Docker image.

Lower total cost of ownership

Kubernetes deployments require fewer hardware resources, such as additional hardware for the fail-over coordinator, and do not require NFS.

Additionally, Kubernetes deployments require less operational interaction, which saves costs. For example, there is no operating system to manage and keep up-to-date. You can perform Azure Kubernetes updates with a single click or command, and everything automatically upgrades in a rolling fashion.

The same applies to a Dremio upgrade: it is just a single command you can integrate into CI/CD tools. In a standalone deployment without automation, you must connect to any single node and perform the upgrade manually. Upgrades for Dremio in Kubernetes ensure that all executors see the same configuration, so there is also less room for mistakes during deployment and upgrades.

Furthermore, Kubernetes deployments reduce the outage time required for upgrades and do not require you to license any operating system, such as Red Hat, to get support.

Faster upgrades

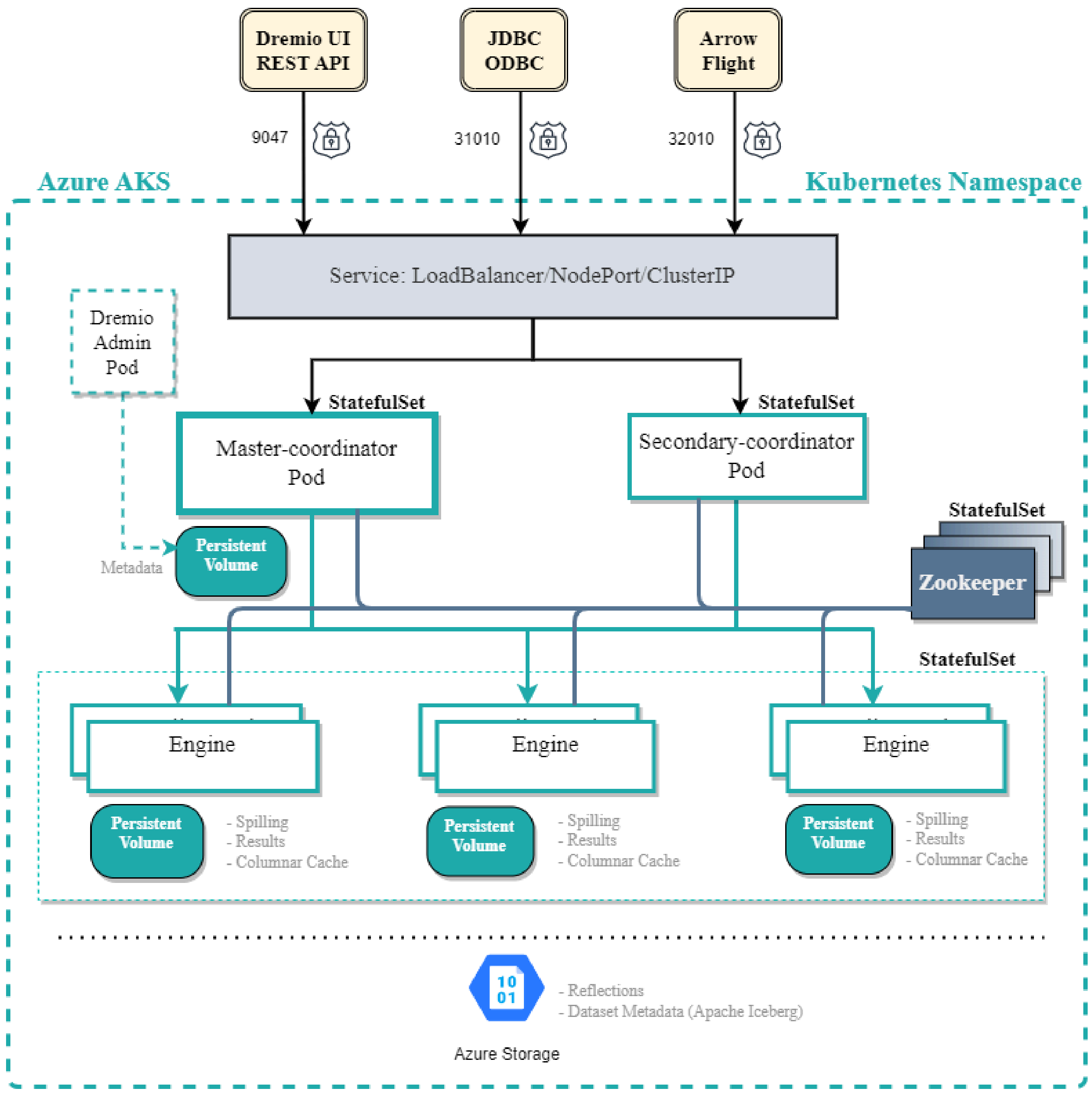
On Kubernetes, Dremio upgrades can be performed in just a few minutes. You only need to modify the `imageTag` value in the Helm Charts values.yml file and then execute the `helm upgrade` command. After a few minutes, the new images are pulled and deployed, and dremio-admin is automatically upgraded as well.

In comparison, a VM upgrade requires more effort. Users need to connect to each node, stop the service, distribute the package, upgrade the RPM package, run the upgrade command, and start the service again. This process could be automated using Ansible, but Kubernetes has an existing automated solution.

Additionally, Azure provides an easy way to upgrade the Kubernetes version of Dremio in the UI or with a single command. The Kubernetes upgrade happens in a rolling fashion. An additional node is deployed in the node pool with the new version. Then an old node gets marked as *unschedulable*, pods move to the new node, and the old node is unprovisioned. This repeats until all old nodes have been replaced.

Dremio Kubernetes Architecture

The architecture below shows an example of an Azure Kubernetes Service deployment.



The following table contains a high-level mapping of the standalone deployment components to Kubernetes terminology:

Standalone Deployment	Kubernetes
External Load-Balancer for the Web-UI, ODBC and Arrow-Flight (Ports 9047, 31010, 32010)	The load balancer in Kubernetes is represented as a service component. A service can be a load balancer or just a cluster IP address. Kubernetes takes responsibility for provisioning the load balancer. Sometimes, no load balancer integration is available, especially in on-premise Kubernetes deployments. In this case, an external load balancer can be provided and bound to the local Kubernetes node ports.
The Dremio coordinator and executors run on standalone VMs and are started and managed with a systemctl service.	The Dremio coordinator and executors run in pods. This is similar to a Docker container instance. Pods can move to another node if a node fails. Pods are orchestrated and managed by Kubernetes. A pod itself is managed by a stateful set. This facilitates scaling the executor nodes and ensures all executor pods see the same configuration.
VMs usually have local disks, block devices, or NFS attached. These disks store Dremio's metadata or store caches for the executors. Dremio recommends SSD or NVMe local disks since they store C3 and Spill data.	Kubernetes has an abstraction layer for disks called Persistent Volume Claim (PVC). A PVC is created based on a storage class. These storage classes define the underlying implementation type, such as NFS or block storage. Avoid local storage, which can have negative impacts in case of failures and does not allow you to reassign pods to other nodes.
Dremio's distributed storage resides on the data lake. This includes metadata, reflections, and uploads. No data is stored on the local disk.	There is no difference for Kubernetes; the distributed storage works the same: Dremio's distributed storage is hosted on the data lake. This includes metadata, reflections, and uploads. No data is stored on the local disk.

Prerequisites

Prerequisites and requirements	Check
Existing Kubernetes cluster must be available	

Prerequisites and requirements	**Check**
Create the cluster with three node pools: ● System node pool: Hosts the Kubernetes system pods and the Zookeeper instances. These nodes require only 4 CPUs and 8 GB RAM. ● Coordinator pool: Hosts the Dremio coordinator (usually one). Nodes can have the same size as in the standalone VM deployment (16 cores, 32 GB RAM). ● Executor pool: Hosts the Dremio executor pods. This pool can be auto-scalable in clouds like AWS, Azure or GCP. Nodes can have the same size as in the standalone VM deployment (16 cores, 128 GB RAM). ● Ensure the Kubernetes cluster has the same firewall openings as the current standalone VM cluster. Possible options and vendors for Kubernetes: ● AWS: Amazon Elastic Kubernetes Service (EKS): https://aws.amazon.com/eks/ ● Azure: Azure Kubernetes Service (AKS): https://azure.microsoft.com/en-us/products/kubernetes-service **Note: Use AzureCNI, not Kubenet! This cannot be changed later. The subnet should have at least 1024 IPs (subnet /22 or larger).** ● Google Cloud Platform (GCP): Google Kubernetes Engine (GKE): https://cloud.google.com/kubernetes-engine ● On-premises: Any Kubernetes distribution should work, but we recommend using a distribution that provides full commercial support. The network bandwidth must be 10 GBits per second or higher.	
Load balancer integration in Kubernetes or standalone load balancer If you use large cloud services like AWS, Azure, or GCP, the load balancer integration should already be in place, and no further action is required. The load balancer in Kubernetes is represented as a service component. Kubernetes takes responsibility for provisioning the load balancer. At times, no load balancer integration is available, especially in on-premise Kubernetes deployments. In such cases, an external load balancer can be deployed and bound to the local Kubernetes node ports. Dremio recommends utilizing integrated load balancer provisioning in Kubernetes.	

Prerequisites and requirements	**Check**
Storage classes must be available Storage in Kubernetes is provisioned with storage classes and PVCs. These storage classes should already be available in the cluster. We recommend using block storage or an NFS integration that supports file system locking. **Common Internet File System (CIFS) is not supported.** The recommended minimum throughput for production environments is 100 MB/s. For an on-premises deployment, the team responsible for Kubernetes administration must provide these storage classes. The three large cloud providers already have storage classes available. The performance correlates with the size of the disk. ● AWS EKS: We recommend EBS CSI. To add the storage class, follow this document: https://docs.aws.amazon.com/eks/latest/userguide/managing-ebs-csi.html The coordinator disk size should start from 500 GB and use IO2 storage with 5000 IOPS (10 IOPS per GB in the storage class). IO2 storage is recommended because of its higher durability and performance. The executors should use gp3 storage with a minimum size of 300 GB and use the free 3000 IOPS and 125 MB/s throughput. For Zookeeper, a 16 GB disk should be sufficient with gp2 or gp3 storage. The storage classes for io2 and gp3 need to be created since they are not available out-of-the-box. ● Azure AKS: The storage class is already available. For Zookeeper, use “managed” and 16 GB. The coordinator needs a 512 GB managed-premium disk. The executors should use a minimum of 256 GB of managed-premium storage. https://learn.microsoft.com/en-us/azure/aks/concepts-storage#storage-classes ● Google GKE: The storage class is already available. For Zookeeper, use “premium-rwo” and 16 GB. The coordinator needs a minimum of 512 GB and “premium-rwo” (PD SSD) and the executors go with a minimum of 256 GB of “premium-rwo” storage. https://cloud.google.com/kubernetes-engine/docs/how-to/persistent-volumes/ssd-pd	
Access to Docker Hub for EE image Register a Docker Hub account. Open a support case via https://support.dremio.com and request access to the Dremio EE repository. Provide the Docker Hub account name in your support case. You should receive access to https://hub.docker.com/repository/docker/dremio/dremio-ee/general.	

Prerequisites and requirements	Check
Container Registry Dremio does not provide service level agreements (SLAs) on the Docker Hub repository. We strongly recommend setting up a container registry to decouple the deployment dependencies and get better performance with a local registry. The three large cloud providers already have container registries available: ● Amazon Elastic Container Registry: https://aws.amazon.com/ecr/ ● Azure Container Registry: https://azure.microsoft.com/en-us/products/container-registry ● Google Artifact Registry: https://cloud.google.com/artifact-registry There are also on-premises solutions, such as JFrog Container Registry. ⚠ NOTICE When importing the images, import the exact same version deployed in your standalone deployment. Migration from a standalone deployment to Kubernetes and upgrading should be two separate steps.	
Kubectl and Helm tools Kubectl is the primary tool to run administrative commands against a Kubernetes cluster. To install Kubectl, read https://kubernetes.io/docs/tasks/tools/. Helm is a tool for managing and orchestrating more complex installations. The Dremio installation on Kubernetes is managed via Helm Charts. To install Helm, read https://helm.sh/docs/intro/install/.	

Prerequisites and requirements	**Check**
Credentials to access the cluster Credentials should be provided in the kube config format: https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/. Cloud providers usually provide their own tools to authenticate against a Kubernetes cluster: ● AWS EKS: https://docs.aws.amazon.com/eks/latest/userguide/create-kubeconfig.html ● Azure AKS: https://learn.microsoft.com/en-us/cli/azure/aks?view=azure-cli-latest#az-aks-get-credentials ● Google GKE: https://cloud.google.com/kubernetes-engine/docs/how-to/cluster-access-for-kubectl	
Kubernetes Namespace to deploy Dremio Dremio is usually installed within a namespace called 'dremio'. If the cluster is exclusive to Dremio, you can use the 'default' namespace instead.	

Migration Steps

Import the Docker EE Image into Your Container Registry

Before you start with the migration, import the Dremio EE docker image into your company's container registry.

⚠ NOTICE
Dremio does not provide any SLAs on the Docker Hub repository. We strongly recommend setting up a container registry to decouple the deployment dependencies and get better performance with a local registry.

The way the import works might differ among vendors, but there is also a generic approach using Docker. The registry already needs to be connected with Docker:

```
# Login into Docker Hub
$ docker login
# Pull the Dremio EE image
$ docker pull dremio/dremio-ee:24.3.0
```

```
# Re-tag the so that it can be pushed to the new registry
$ docker tag dremio/dremio-ee:24.3.0 yourregistry.azurecr.io/dremio-ee:24.3.0
# Push the re-tagged image to the registry
$ docker push yourregistry.azurecr.io/dremio-ee:24.3.0
```

Collect Configurations from the Standalone Deployment

The following configurations need to be collected from the existing standalone deployment:

- The TLS certificate and private key in a PEM format. For more information about the PEM format, read https://en.wikipedia.org/wiki/Privacy-Enhanced_Mail.
- The distributed storage credentials and configuration from the "/opt/dremio/conf/core-site.xml" file and any core-site.xml custom settings.
- Authentication configuration for LDAP, AD, AAD, or OAuth2 (usually a separate file that you can copy as-is and reuse in the Kubernetes deployment).
- Sizing and memory configuration.
- Additional configuration for the Java Virtual Machine, such as garbage collection settings, logging changes, and additional flags for Dremio.

Perform a Backup of Dremio MetaStore

Take a backup from the VM standalone deployment. The backup is stored in the data lake with the credentials of the distributed storage. Depending on the size of the metastore, the backup can take a while.

Backup with AWS S3

⚠ NOTICE
This step does not apply to Dremio AWSE Edition. Please continue at "Backup for AWSE Edition"

Run the following command, replacing *<s3_bucket>* with the name of your S3 bucket.

```
$ /opt/dremio/bin/dremio-admin backup -l -d dremioS3:///<s3_bucket>/dremio-backup
```

Backup with Azure ADLS

The backup is stored in the storage account configured in core-site.xml. The backup container name is "dremio-backup".

```
/opt/dremio/bin/dremio-admin backup -l -d dremioAzureStorage://:///dremio-backup
```

Backup with Google GCS

Run the following command, replacing *<gcs_bucket>* with the name of your S3 bucket.

```
$ /opt/dremio/bin/dremio-admin backup -l -d dremiogcs:///<gcs_bucket>/dremio-backup
```

⚠ NOTICE
After the backup completes, keep Dremio stopped so that no other changes can be applied to the cluster (e.g. VDS changes).

Backup for AWSE Edition

The backup and restore process for Dremio AWSE differs from Dremio EE.

Using the Dremio REST APIs to implement a migration process is recommended. You can find the Dremio REST API reference here: https://docs.dremio.com/24.0.x/reference/api/

An example implementation for the REST API usage is Dremio Cloner. Dremio Cloner is a **community-maintained** project that allows the export and import of Dremio's catalog via the REST APIs: https://github.com/deane-dremio/dremio-cloner

⚠ NOTICE
As of Dremio version 24.3, there is only a limited REST API available to manage Dremio's User Defined Functions (UDFs) and scripts.

When using **Dremio Cloner,** the following objects are not exported and need to be copied manually:

- Home spaces (Views/VDS, Uploads, etc)
- Scripts
- User-defined functions (UDFs)

Clone and Configure the Helm Charts

The Dremio Helm Charts are available at https://github.com/dremio/dremio-cloud-tools.

Clone the Repository

Clone the repository to your local machine:

```
$ git clone https://github.com/dremio/dremio-cloud-tools.git
```

Go to the Helm Charts (v2). Create a copy of the values.yml file and name it after your environment (for example, “dev” or “prod”). The original values.yml file should not be modified so that it can be used as a template.

```
# Go into the main folder of the Helm Charts
$ cd dremio-cloud-tools/charts/dremio_v2
# Create a copy of values.yml and name it after your environment
$ cp values.yml values-myenvname.yml
# Open the configuration file with your favorite editor
$ vi values-myenvname.yml
```

Change the Container Image to Your Container Registry

The default configuration in the values.yml file points to the Docker Hub images and uses a ‘latest’ image tag. The image tag should always point to a specific version, like ‘24.2.0’.

```
# Point the container image to your company's container registry
image: yourregistry.azurecr.io/dremio-ee
# Never use 'latest'
imageTag: 24.2.0
```

Optional: If the container registry is not connected to the cluster, you can also create an image pull secret:

```
$ kubectl create secret docker-registry mycontainerregistry \
  --docker-server=DOCKER_REGISTRY_SERVER \
  --docker-username=DOCKER_USER \
  --docker-password=DOCKER_PASSWORD \
  --docker-email=DOCKER_EMAIL
```

Add the image pull secret to values-myenv:

```
imagePullSecrets:
  - mycontainerregistry
```

CPU, Memory, Disks, and Node Pool Configuration

It is possible to deploy Dremio only on one node pool. Nevertheless, we recommend splitting up the deployment and allocating it to different node pools for the following reasons:

- To ensure better and faster failover behavior, Zookeeper should not run on Dremio coordinator or executor nodes.
- The Kubernetes-system pods should run on dedicated nodes.
- In case of overutilization of executor pods, they would not be affected (same applies to Zookeeper).
- All node resources can be allocated to the executors since there is fine-grained control over where the pods live.
- You can choose a different size for coordinator and executor nodes if needed.
- You can replace node pools with larger or smaller instances
- The executor pool can be auto-scalable and does not need to be touched in Azure when scaling up with Helm Charts.

The following table describes an example layout for the node pools:

Node Pool Name	Node Count	Mode	Node Size	Description
agentpool	3	System	2x CPU 8 GB RAM	Hosts kube-system pods, monitoring and Zookeeper
coordpool	1	User	16x CPU 32 GB RAM	Hosts the coordinator pod. When scaling out, the number can be increased.
executorpool	1 to n static or auto-scalable	User	32x CPU 128 GB RAM	Hosts the executor pods. The scaling happens automatically, based on the number of executors in the Helm Charts.

Not all CPU and memory resources of a node can be allocated to Dremio because there will be overhead for the operating system and Kubernetes management pods. To find out how many resources can be allocated to Dremio, use the following command:

```
$ kubectl describe node aks-executorpool-11591932-vmss00002n
...
Allocatable:
  cpu:                15740m
  ephemeral-storage:  119703055367
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  memory:             59341828Ki
  pods:               30
...
# Remember to subtract the Kubernetes system pods from the allocatable resources.
```

👍TIP

Rule of Thumb for CPU and Memory Resource Allocation

For CPUs, reserve 2 CPUs for Kubernetes system pods and operating system. For example, if the node has 16 CPUs, allocate 14 CPUs to Dremio. If the node has 32 CPUs, allocate 30 CPUs.

For memory, multiply the total available memory by 0.875. For example, if the machine has 128 GB of physical memory, allocate 112 GB of memory to Dremio.

The storage classes and disk sizes depend on your Kubernetes cluster implementation. Read the prerequisites for storage class availability for more information.

The example below shows a configuration for Azure. For the hardware setup in the table above, you might use the following Dremio Helm Chart configuration in the *values-myenv.yml* file:

```
# This example contains only the parts necessary to clarify how to allocate resources
and define the node selectors
coordinator:
  cpu: 14
  memory: 28672
  # Count is 0 if only one coordinator is required
  count: 0
  volumeSize: 512Gi
  # Based on your Kubernetes version; this example is for Azure
  storageClass: managed-premium
  nodeSelector:
    agentpool: coordpool
...
executor:
  cpu: 30
  memory: 114688
  engines: ["default"]
  # Replace with the desired number of nodes
  count: 5
  volumeSize: 256Gi
  # Based on your Kubernetes version; this example is for Azure
  storageClass: managed-premium
  nodeSelector:
    agentpool: executorpool
...
# Zookeeper
zookeeper:
  # The Zookeeper image used in the cluster.
  image: zookeeper
  imageTag: 3.8-temurin
  cpu: 1
  memory: 1024
  count: 3
  volumeSize: 8Gi
  # Based on your Kubernetes version; this example is for Azure
  storageClass: managed
  nodeSelector:
     agentpool: agentpool
```

Additional Java Options (recommended)

It is possible that your standalone deployment includes customizations like additional monitoring and garbage collection optimizations.

⚠ NOTICE
We recommend adding a custom logging path to the configuration so that logs always persist. Additionally, we recommend an improved garbage collection configuration, such as the example below.

The following example describes what to add to the values-myenv.yml file:

```
# Java options for the coordinator: changed logging path + improved GC
coordinator:
  extraStartParams: >-
    -Ddremio.log.path=/opt/dremio/data/log
    -XX:+UseG1GC -XX:G1HeapRegionSize=32M -XX:MaxGCPauseMillis=500
-XX:InitiatingHeapOccupancyPercent=25
    -Xloggc:/opt/dremio/data/log/gc-%t.log -XX:+PrintGCTimeStamps
# Java options for the executors: changed logging path + improved GC
```

```
executor:
  extraStartParams: >-
    -Ddremio.log.path=/opt/dremio/data/log
    -XX:+UseG1GC -XX:G1HeapRegionSize=32M -XX:MaxGCPauseMillis=500
-XX:InitiatingHeapOccupancyPercent=25
    -Xloggc:/opt/dremio/data/log/gc-%t.log -XX:+PrintGCTimeStamps
```

Add additional parameters into the configuration by adding new lines.

Distributed Storage Configuration

The distributed storage configuration is mandatory. Dremio does not support any local distributed storage.

There are several supported storage types in the Helm Charts:

- AWS S3 (or S3 compatible)
- Azure Storage Gen 2
- Google Cloud Storage

The following storage types are not recommended:

- Local (only Dremio < 21.0.0)
- Azure Storage Gen 1 (will be discontinued in 2024 by Azure)
- HDFS and MapR-FS (not yet supported by Helm Charts)

The configuration for the distributed storage should be available in the core-site.xml file of the Dremio standalone deployment that is being migrated.
The configuration happens in the *values-myenv.yml* file.

AWS S3 - Distributed Storage Configuration

Choose the desired authentication option and replace values like bucket name and path with the correct values for your configuration.

```
distStorage:
  type: "aws"
  aws:
    bucketName: "AWS Bucket Name"
    path: "/"
    authentication: "metadata"
    # If using accessKeySecret for authentication against S3, uncomment the lines below
and use the values
```

```
    # to configure the appropriate credentials.
    #
    #credentials:
    #  accessKey: "AWS Access Key"
    #  secret: "AWS Secret"
    #
    # If using awsProfile for authentication against S3, uncomment the lines below and
use the values
    # to choose the appropriate profile.
    #
    #credentials:
    #  awsProfileName: "default"
    #
    # Extra Properties
    # Use the extra properties block to provide additional parameters to configure the
distributed
    # storage in the generated core-site.xml file.
    #
    #extraProperties: |
    #  <property>
    #    <name></name>
    #    <value></value>
    #  </property>
```

Azure ADLS - Distributed Storage Configuration

Replace the values for storage account name, container name, and storage account access key with the correct values for your configuration.

```
distStorage:
  type: "azureStorage"
  azureStorage:
    accountName: "<Storage Account Name>"
    filesystem: "<Container Name>"
    path: "/"
    credentials:
      accessKey: "<Storage Account Access Key>"
```

Google GCS - Distributed Storage Configuration

Choose the desired authentication option and replace values like bucket name and path with the correct values for your configuration.

```
distStorage:
  type: "gcp"
  gcp:
    bucketName: "<GCS Bucket Name>"
    path: "/"
    authentication: "auto"
    # If using serviceAccountKeys, uncomment the section below, referencing the values 
from
    # the service account credentials JSON file that you generated:
    #
    #credentials:
    #  projectId: GCP Project ID that the Google Cloud Storage bucket belongs to.
    #  clientId: Client ID for the service account that has access to Google Cloud 
Storage bucket.
    #  clientEmail: Email for the service account that has access to Google Cloud 
Storage bucket.
    #  privateKeyId: Private key ID for the service account that has access to Google 
Cloud Storage bucket.
    #  privateKey: |-
    #    -----BEGIN PRIVATE KEY-----\n Replace me with full private key value.
\n-----END PRIVATE KEY-----\n
```

TLS Configuration

The certificate and private key must be provided in PEM format to use them in Kubernetes.

If the certificate is in PKCS12 format, you can use the following command to extract the private key and certificate in PEM format:

```
# Optional if keys are in PKCS12
$ openssl pkcs12 -in mydomain.p12 -out mydomain.crt.pem -clcerts -nokeys
$ openssl pkcs12 -in mydomain.p12 -out mydomain.key.pem -nocerts -nodes
```

If the keys are in JKS format, convert the store into PKCS12 and then extract the PEM from PKCS12:

```
# Optional if keys are in JKS
$ keytool -importkeystore -srckeystore mydomain.jks -destkeystore mydomain.p12 
-srcstoretype jks -deststoretype pkcs12
$ openssl pkcs12 -in mydomain.p12 -out mydomain.crt.pem -clcerts -nokeys
$ openssl pkcs12 -in mydomain.p12 -out mydomain.key.pem -nocerts -nodes
```

⚠ IMPORTANT NOTICE
The certificate file ‘mydomain.crt.pem’ should contain the entire certificate chain. The most specific certificate comes first, then the intermediate certificate, and then the root certificate.
If 'mydomain.crt.pem' does not contain the entire certificate chain, this might lead to issues with certain browsers, especially during PowerBI integration.

Once you have the certificate and private key in PEM format, run the following command to create a secret in Kubernetes:

```
$ kubectl create secret tls dremio-tls-secret --key mydomain.key.pem --cert
mydomain.crt.pem
```

Next, enable TLS, reference the ‘dremio-tls-secret’ in the values-myenv.yml file, and change the web port to 443:

```
coordinator:
  web:
    port: 443
    tls:
      enabled: true
      secret: dremio-tls-secret
  # ODBC/JDBC Client
  client:
    port: 31010
    tls:
      enabled: true
      secret: dremio-tls-secret
  # Flight Client
  flight:
    port: 32010
    tls:
      enabled: true
      secret: dremio-tls-secret
```

Ensure that service type ‘LoadBalancer’ is set:

```
service:
 type: LoadBalancer
```

Authentication - LDAP/AAD/OAuth2 Integration

An existing authentication integration should be available in the standalone deployment. You can skip this step if no authentication integration is configured and you work with Dremio local users.

Copy the authentication configuration file (e.g. azuread.json or ldap.json) to:

```
dremio-cloud-tools/charts/dremio_v2/config
```

Open the values-myenv.yaml file and add one of the following lines to extraStartParams (choose only one):

```
coordinator:
 ...
 # Example for OAuth2
 extraStartParams: >-
   -Dservices.coordinator.web.auth.type=oauth
   -Dservices.coordinator.web.auth.config=oauth.json
 ...
 # Example for Azure AD/AAD
 extraStartParams: >-
   -Dservices.coordinator.web.auth.type=azuread
   -Dservices.coordinator.web.auth.config=azuread.json
 ...
 # Example for LDAP
 extraStartParams: >-
   -Dservices.coordinator.web.auth.type=ldap
   -Dservices.coordinator.web.auth.config=ldap.json
```

Run the Helm Deployment

After the configuration is complete, you can run the Helm deployment.
Verify that you work on the correct cluster:

```
$ kubectl config current-context
my-prod-kubernetes-cluster
```

Verify that the connection to the Kubernetes cluster works:

```
$ kubectl get nodes
NAME                                   STATUS   ROLES   AGE     VERSION
aks-agentpool-25792647-vmss00003z      Ready    agent   9m46s   v1.25.6
aks-coordpool-33677896-vmss00001o      Ready    agent   9m38s   v1.25.6
aks-executorpool-21759880-vmss00003w   Ready    agent   9m43s   v1.25.6
aks-executorpool-21759880-vmss00003x   Ready    agent   9m36s   v1.25.6
…
```

Run the Helm command to start the installation:

```
$ helm upgrade dremio --install -n dremio dremio-cloud-tools/charts/dremio_v2 -f
dremio-cloud-tools/charts/dremio_v2/values-myenv.yaml --wait

Release "dremio" has been installed. Happy Helming!
NAME: dremio
LAST DEPLOYED: Fri Jun 16 11:11:25 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Verify that all pods are up and running:

```
$ kubectl get pods
NAME                            READY   STATUS      RESTARTS      AGE
dremio-executor-0               1/1     Running     0             75m
dremio-executor-1               1/1     Running     0             75m
dremio-master-0                 1/1     Running     0             75m
```

Add DNS Entry for the Load Balancer

This step only applies if you provisioned the load balancer with the Helm Charts. If you created the load balancer externally, skip this step.

```
$ kubectl get svc -n dremio
NAME                 TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)
AGE
dremio-client        LoadBalancer   10.0.106.137   4.156.48.192
31010:30762/TCP,443:32246/TCP,32010:30691/TCP   77d
dremio-cluster-pod   ClusterIP      None           <none>         9999/TCP
```

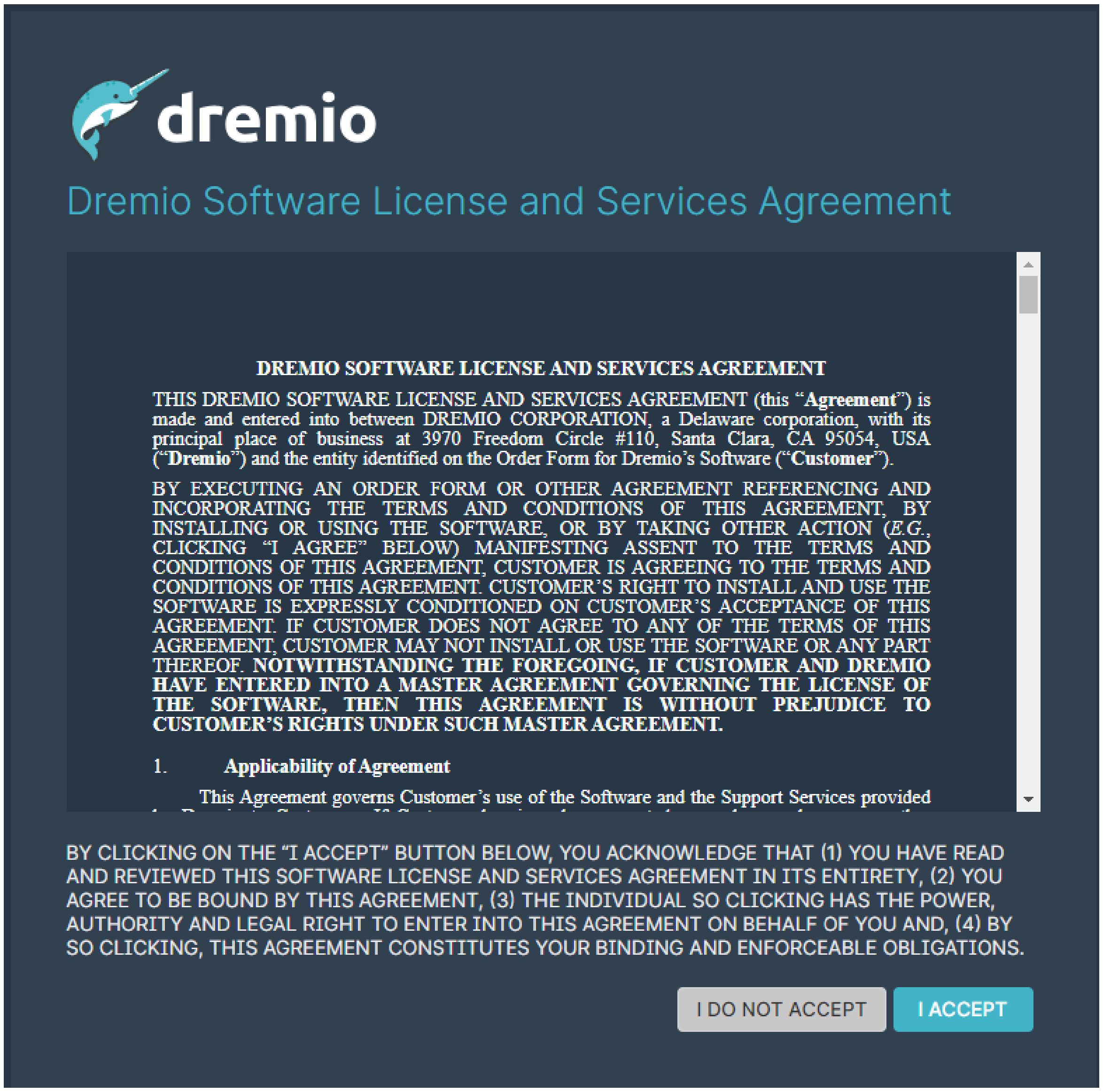
```
77d
kubernetes             ClusterIP      10.0.0.1       <none>         443/TCP
594d
zk-cs                  ClusterIP      10.0.81.202    <none>         2181/TCP
77d
zk-hs                  ClusterIP      None           <none>
2181/TCP,2888/TCP,3888/TCP                     77d
```

You can find the IP address at ‘dremio-client’ > EXTERNAL-IP (in this example, 4.156.48.192). Add the IP address to the DNS records with your Dremio domain (e.g. dremio.mydomain.com).

After you add the IP address to the DNS records with your Dremio domain, you can access Dremio at
https://dremio.mydomain.com.

If you did not change the port from 9047 to 443, the URL would be
https://dremio.mydomain.com:9047.

You should see the Dremio Software License and Services Agreement screen because no data has been restored yet:



Dremio Software License and Services Agreement

DREMIO SOFTWARE LICENSE AND SERVICES AGREEMENT

THIS DREMIO SOFTWARE LICENSE AND SERVICES AGREEMENT (this "**Agreement**") is made and entered into between DREMIO CORPORATION, a Delaware corporation, with its principal place of business at 3970 Freedom Circle #110, Santa Clara, CA 95054, USA ("**Dremio**") and the entity identified on the Order Form for Dremio's Software ("**Customer**").

BY EXECUTING AN ORDER FORM OR OTHER AGREEMENT REFERENCING AND INCORPORATING THE TERMS AND CONDITIONS OF THIS AGREEMENT, BY INSTALLING OR USING THE SOFTWARE, OR BY TAKING OTHER ACTION (*E.G.*, CLICKING "I AGREE" BELOW) MANIFESTING ASSENT TO THE TERMS AND CONDITIONS OF THIS AGREEMENT, CUSTOMER IS AGREEING TO THE TERMS AND CONDITIONS OF THIS AGREEMENT. CUSTOMER'S RIGHT TO INSTALL AND USE THE SOFTWARE IS EXPRESSLY CONDITIONED ON CUSTOMER'S ACCEPTANCE OF THIS AGREEMENT. IF CUSTOMER DOES NOT AGREE TO ANY OF THE TERMS OF THIS AGREEMENT, CUSTOMER MAY NOT INSTALL OR USE THE SOFTWARE OR ANY PART THEREOF. **NOTWITHSTANDING THE FOREGOING, IF CUSTOMER AND DREMIO HAVE ENTERED INTO A MASTER AGREEMENT GOVERNING THE LICENSE OF THE SOFTWARE, THEN THIS AGREEMENT IS WITHOUT PREJUDICE TO CUSTOMER'S RIGHTS UNDER SUCH MASTER AGREEMENT.**

1. **Applicability of Agreement**

This Agreement governs Customer's use of the Software and the Support Services provided

BY CLICKING ON THE "I ACCEPT" BUTTON BELOW, YOU ACKNOWLEDGE THAT (1) YOU HAVE READ AND REVIEWED THIS SOFTWARE LICENSE AND SERVICES AGREEMENT IN ITS ENTIRETY, (2) YOU AGREE TO BE BOUND BY THIS AGREEMENT, (3) THE INDIVIDUAL SO CLICKING HAS THE POWER, AUTHORITY AND LEGAL RIGHT TO ENTER INTO THIS AGREEMENT ON BEHALF OF YOU AND, (4) BY SO CLICKING, THIS AGREEMENT CONSTITUTES YOUR BINDING AND ENFORCEABLE OBLIGATIONS.

Restore the Backup of the Standalone Deployment

With Dremio running, you can restore the data from the backup.

Change the cluster to admin mode. If the coordinator node is running, there is a lock on the RocksDB and a restore will not work. The following command will stop the coordinator and launch an admin pod:

```
$ helm upgrade dremio -n dremio dremio-cloud-tools/charts/dremio_v2 -f
dremio-cloud-tools/charts/dremio_v2/values-myenv.yaml --set DremioAdmin=true --wait
```

Verify that the dremio-admin pod is available:

```
$ kubectl get pods
NAME                                            READY    STATUS       RESTARTS    AGE
dremio-admin                                    1/1      Running      0           98s
```

Connect to the dremio-admin pod:

```
$
```

Confirm that the following directory is empty (if it is not empty, the restore will fail):

```
/opt/dremio/data/db
```

If the directory is not empty, run the following command:

```
$ rm -Rf /opt/dremio/data/db/*
```

When the backup was taken at the beginning of the migration, it was written into the data lake of the distributed storage. The backup should have a timestamp and name like 'dremio_backup_2023-09-12_14.00'. Look up this name in S3, ADLS, or GCS.

Restore from S3 Data Lake

⚠ NOTICE
This step does not apply to Dremio AWSE Edition. Please see "**Backup for AWSE Edition**"

Run the following command (replace 'dremio_backup_2023-09-12_14.00'):

```
$ /opt/dremio/bin/dremio-admin restore -d
dremioS3:///my_s3_bucket//dremio-backup/dremio_backup_2023-09-12_14.00
```

Restore from Azure Data Lake

Run the following command (replace 'dremio_backup_2023-09-12_14.00'):

```
$ /opt/dremio/bin/dremio-admin restore -d
dremioAzureStorage://:///dremio-backup/dremio_backup_2023-09-12_14.00
```

Restore from Google Cloud Storage

Run the following command (replace 'dremio_backup_2023-09-12_14.00'):

```
$ /opt/dremio/bin/dremio-admin restore -d
dremiogcs:///<my_gcs_bucket>/<my_folder>/dremio_backup_2023-09-12_14.00
```

After the restore finishes, a “completed” message is displayed. The process can be terminated even if the command does not return the message to the bash prompt.

Stop the Dremio admin node and start it again:

```
$ helm upgrade dremio -n dremio dremio-cloud-tools/charts/dremio_v2 -f
dremio-cloud-tools/charts/dremio_v2/values-myenv.yaml --set DremioAdmin=false --wait
```

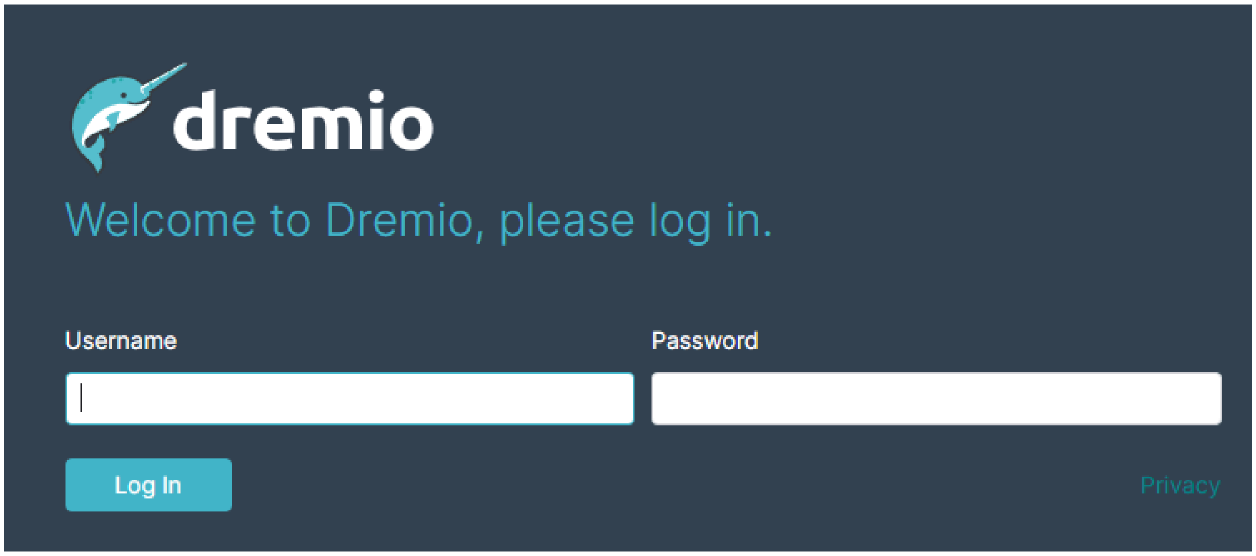
Verify that all pods started up and run again:

```
$ kubectl get pods
NAME                            READY   STATUS      RESTARTS      AGE
dremio-executor-0               1/1     Running     0             75m
dremio-executor-1               1/1     Running     0             75m
dremio-master-0                 1/1     Running     0             75m
```

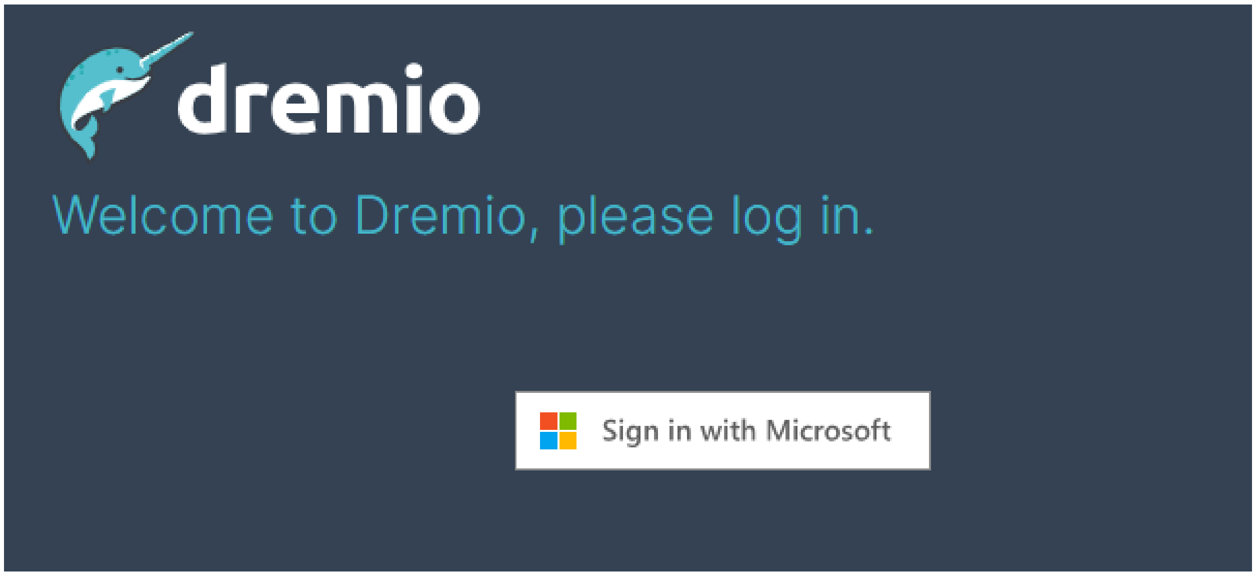
Verify the Deployment

Go to https://dremio.mydomain.com. You should see a login screen instead of the Dremio Software License and Services Agreement screen.

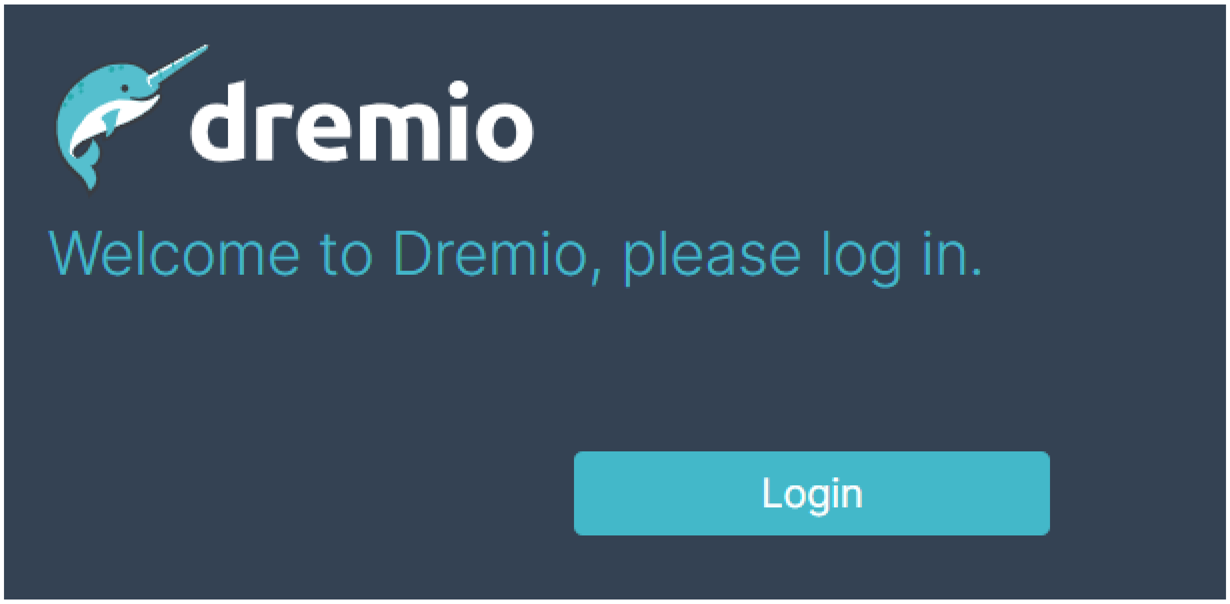
If LDAP or no additional authentication is configured. In that case, you should be prompted for username and password:



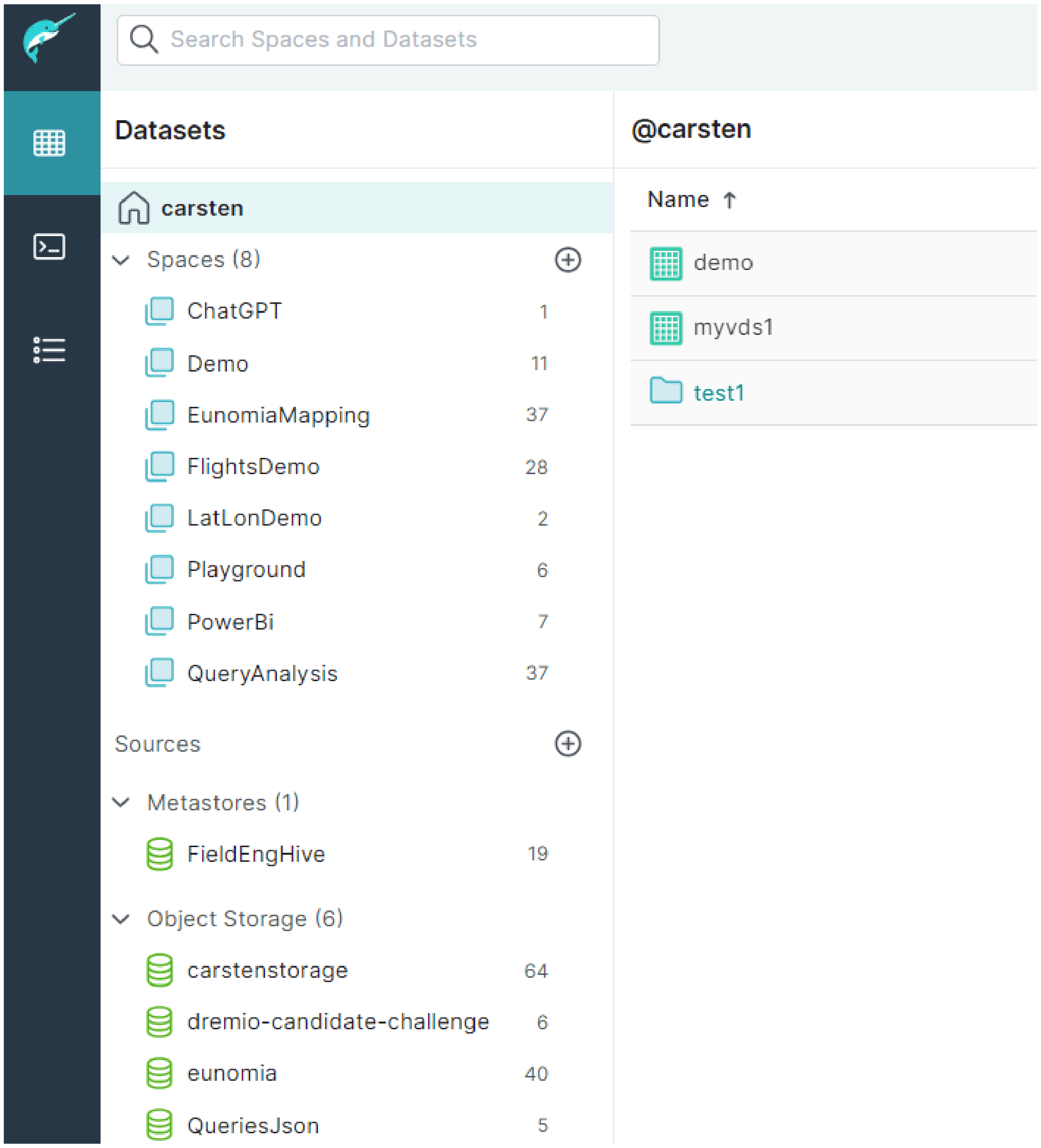
If you configured Azure Active Directory, the login screen should look like this:



If OAuth2 is configured, the login screen should look like this:



After you log in, you should be able to see all sources, spaces, and view definitions from the former standalone deployment:



Stop and Decommission Standalone Nodes

Let your team verify that everything has migrated properly. We also recommend that you keep the old standalone version of Dremio for a few days to restore anything missed during the migration. The instances need not be turned on and can be completely stopped.

Next Steps

After the migration is complete, consider the following steps:

- Use a static IP address for the load balancer so that the IP stays stable, even when the Helm Release is uninstalled.
- Use Nginx ingress to serve Dremio UI HTTP requests. This allows a seamless integration with certificate management, which can automatically renew certificates.
- Create an automated backup cron job.
- Add Prometheus or application-level monitoring.
- Integrate the deployment process into CI/CD (e.g. Azure DevOps, Jenkins, or GitHub Actions)