MarkLogic Server on Kubernetes

MarkLogic 11 Kubernetes 2.0

Publication date 2024-07-24

Table of Contents

1. Overview of Kubernetes

Containerization is a process that bundles application code and all its dependent components into a single package. The resulting package is known as a container. Containers include all the files, resources, and libraries needed to run an application on any computer operating system or infrastructure. Containers are lightweight and memory efficient when compared to virtual machines and other virtualization technologies.

Docker and Kubernetes are containerization platforms often used together. Docker is used to create containers. Kubernetes, on the other hand, is a container management tool. Kubernetes allows developers to deploy and manage containerized applications at scale across multiple hosts or cloud providers, and it provides a platform for building microservices-based applications. It automates the deployment of containers and provides load balancing, scaling, and self-healing functions. These functions make it easier for developers to manage their infrastructure so that they can focus on writing code.

By combining MarkLogic with containers using Docker and Kubernetes, developers can quickly collaborate and release code faster and more efficiently. Because containers are platform agnostic, applications can be built once and run in a variety of scenarios including on-premise environments; private, hybrid, or public clouds; and on AWS, Azure, and Google Cloud. By using containers, Docker, and Kubernetes, MarkLogic developers will realize the benefits of a flexible, light-weight, and cost-effective alternative to virtual machines.

1.1. Compatibility

MarkLogic Server

MarkLogic Server Version	Docker Image Version
9	Unsupported
10	10.0-9.5-centos-1.0.2 or later
11	11.0.2-centos-1.0.2 or later

Kubernetes

Kubernetes 1.23 or later.

Managed Kubernetes

The MarkLogic Helm Chart is currently tested on Amazon EKS and Azure AKS. Setup and operational instructions are currently only provided for Amazon EKS.

1.2. Terminology

The following terms are used throughout this guide:

Term	Definition
Container	A container is a unit of software containing application code and all the libraries, files, and dependent resources that enable an application to run efficiently and reliably in different environments.
Node	A node is a physical or virtual machine. There are two types of nodes: • A master node contains the control plane that manages the node. • A worker node processes data stored in the cluster and ensures that traffic to and from the application is properly facilitated.
Cluster	A cluster is a group of nodes.

Term	Definition
Control Plane	The control plane manages clusters and the workloads running on them. The control plane manages scheduling and detects and responds to events. The control plane operates on one or more machines within a cluster.
Pod	A pod is a group of one or more containers with shared storage, network resources, and a specification for how to run the containers. In Kubernetes, applications and the accompanying utilities are hosted in pods. A pod can also operate as a logical host. A MarkLogic pod is managed by StatefulSet workload resources.
StatefulSet	StatefulSet is used to manage stateful applications by managing the deployment and scaling of a set of pods. StatefulSet also guarantees the ordering and uniqueness of pods.
Namespace	A namespace is a mechanism for isolating groups of resources within a single cluster.
Service	A service is an abstract way of exposing an application running as a network service on a set of pods.
Ingress	Ingress is a Kubernetes resource that manages external access to the services in a cluster (typically using HTTP). An Ingress also provides load balancing functions.
ConfigMap	A ConfigMap is an API object used to store data in key-value pairs.
Secret	A secret is an object that contains a small amount of sensitive data, such as a password, a token, or a key.
Load Balancing	Load balancing is the methodical and efficient distribution of network or application traffic across multiple servers.

2. Set up the required tools

To run MarkLogic in Kubernetes, Helm and kubectl are required. Instructions for installing and configuring these tools are included in this section.

NOTE
Enter all commands referenced in this section into the command-line interpreter for your operating system (Linux - *Shell*, Windows- *PowerShell*, Mac - *Terminal*).

2.1. Install Helm

Helm is a package manager that makes it easy to install MarkLogic on Kubernetes.

To install Helm, follow these steps:

1. Follow the steps at Installing Helm.
2. For Windows computers, add the location of Helm to the path user environment variable.
3. Verify installation by entering this command:

```
helm-h
```

   - If the installation was successful, an explanation of the common actions appears.
   - If the installation was unsuccessful, the `command not found: helm` error appears.

2.2. Install kubectl

kubectl is a command-line tool used as a client to connect to a Kubernetes cluster. kubectl can also be used to run commands against a cluster, to pass Kubernetes object specifications in a YAML file, and to deploy and manage MarkLogic resources.

To install kubectl, follow these steps:

1. Follow the steps at Install Tools: kubectl.
2. Verify the installation by entering this command:

```
kubectl -h
```

   - If the installation was successful, the help content appears.
   - If the installation was unsuccessful, the `command not found: kubectl` error appears.

2.3. Tools for setting up the Kubernetes cluster

This section describes the tools needed to set up a Kubernetes cluster.

2.3.1. Install Minikube (for local development)

Minikube is a Kubernetes implementation that creates a virtual machine on a local machine and deploys a cluster containing a single node.

To install Minikube for local development, follow the installation instructions in the local development tutorial.

Start Minikube

To start Minikube, enter this command:

```
minikube start
```

Minikube Dashboard

To see the components that are created when Minikube is installed, enter this command:

```
minikube dashboard
```

2.3.2. Install Amazon Web Services Elastic Kubernetes Service (for production)

Amazon Web Services Elastic Kubernetes Service, or EKS, is a managed Kubernetes platform provided by Amazon Web Services. The eksctl tool is a simple way to bring up a Kubernetes cluster.

Install eksctl

To install eksctl, follow the installation instructions at Installing or updating eksctl.

Use eksctl to provision a Kubernetes cluster on EKS

The following eksctl code can be used to create a Kubernetes cluster in EKS. Replace the items in capital letters with the correct values for your configuration. For an explanation of the parameters, see Helm chart parameters.

```
eksctl create cluster \
  --name CLUSTER_NAME \
  --version KUBERNETES_VERSION \
  --region REGION \
  --nodegroup-name NODEGROUP_NAME \
  --node-type NODE_TYPE \
  --nodes NUMBER_OF_NODES
```

2.3.3. Parameters

Value	Description
CLUSTER_NAME	A unique (distinctive) name for the cluster.
KUBERNETES_VERSION	The version of Kubernetes in use.
NODEGROUP_NAME	A unique (distinctive) name for the node group.
NODE_TYPE	The type of node. It is recommended to set this to **`r5.large`**.
NUMBER_OF_NODES	Total number of nodes running a MarkLogic database + nodes running other applications.

3. Create a MarkLogic cluster

This section describes how to add the MarkLogic Kubernetes repository. It includes the steps to create a three-node MarkLogic cluster with resource allocation using a Helm Chart.

3.1. Add the MarkLogic repository

To add the MarkLogic repository to Helm, follow these steps:

1. Enter this command:

   ```
   helm repo add marklogic https://marklogic.github.io/marklogic-kubernetes/
   ```

   The message `"marklogic" has been added to your repositories` appears.
2. Verify that the repository was added to Helm by entering this command:

   ```
   helm repo list
   ```

   An entry like `marklogic https://marklogic.github.io/marklogic-kubernetes/` appears.
3. To ensure the Helm repository is up to date, enter this command:

   ```
   helm repo update
   ```

3.2. Install the chart

NOTE
It is recommended to deploy the chart in an exclusive namespace.

To install the chart, follow these steps:

1. To create a three-node MarkLogic cluster with a resource allocation of 16 vCPUs, 128 GB RAM, and storage of 500 GB, update the settings in the `values.yaml` file as shown:

   NOTE
   Use the latest MarkLogic Docker image for the new implementation as specified in the `values.yaml` file below. Refer to dockerhub for the latest image available.

```
# Number of Marklogic nodes
replicaCount: 3

# Marklogic image parameters
# using the latest image 11.0.3-centos-1.0.2
image:
  repository: marklogicdb/marklogic-db;
  tag: 11.0.3-centos-1.0.2
  pullPolicy: IfNotPresent

# Configure persistence using persistent Volume Claim
persistence:
  storageClass:“<storageClass-name>”
  enabled: true
  size: 500Gi

# Compute Resources
resources:
  requests:
    cpu: 16000m
    memory: 128Gi
```

NOTE
storageClass-name is used for gp2, gp3 (for EKS), or custom.

2. Create a Kubernetes Secret for the MarkLogic admin credentials. The secret should include the username, password, and wallet password. The credentials should be inserted between the '' marks when using this command:

```
kubectl create secret generic ml-admin-secrets \
--from-literal=username='' \
--from-literal=password='' \
--from-literal=wallet-password=''
```

3. Set the parameter *auth.secretName* in the `values.yaml` file:

```
# If no secret is specified and the admin credentials are not provided, a secret will
be automatically# generated with random admin and wallet passwords.
auth:
  secretName: "ml-admin-secrets"
```

4. Create a Kubernetes Secret for the credentials of the private image repository. Use the `kubectl create secret` command with the credentials needed to access the repository. In this example, the username and password are set:

```
image-repo-secrets
```

5. Once the secret is created, set the value for *imagePullSecrets.name* in the `values.yaml` file:

```
# Configure the imagePullSecrets to pull the image from private repository that
requires credential
imagePullSecrets:
  - name: "image-repo-secrets"
```

6. Next, install the chart to the current namespace using the settings in the `values.yaml` file by entering this command:

```
helm install my-release marklogic/marklogic --version <version> --values values.yaml
-n <release-namespace>
```

Once the installation is successful, this output appears:

```
NAME: my-release
LAST DEPLOYED:
NAMESPACE: <release-namespace>
STATUS: deployed
REVISION: 1
```

7. Verify the deployment by entering this command:

```
helm list -n <release-namespace>
```

3.3. Deploy Helm with HTTPS enabled

The MarkLogic Helm Chart supports installing MarkLogic with HTTPS enabled on the default app servers. The default app servers are App-Services (8000), Admin (8001), and Manage (8002)

Choose the type of certificate

Two types of certificates are supported: standard certificates and temporary certificates.

- Temporary Certificates - A temporary certificate is ideal for development purposes. When using a temporary certificate for MarkLogic App Servers, a signed certificate does not need to be supplied. The certificate will be generated automatically.
- Standard Certificates - A standard certificate is issued by a trusted Certificate Authority (CA) for a specific domain (host name for MarkLogic server). A standard certificate is strongly recommended for production environments. Support is provided for both named certificates and wildcard certificates.
  - Named Certificate - Each host must possess a designated certificate with a matching common name (CN).
  - Wildcard Certificate - A single wildcard certificate can be used for all hosts within a cluster.

3.3.1. Configure a MarkLogic cluster with a standard certificate

To configure a MarkLogic cluster with a standard certificate, follow these steps:

1. Obtain a certificate with a common name matching the hostname of the MarkLogic host. The certificate must be signed by a trusted Certificate Authority (CA). Either a publicly rooted CA or a private CA can be used. This example uses a private CA and a 2-node cluster.
2. Use this script to generate a self-signed CA certificate with openSSL. The script will create `ca-private-key.pem` as the CA key and `cacert.pem` as the private CA certificate:

```
# Generate private key for CA
openssl genrsa -out ca-private-key.pem 2048

# Generate the self-signed CA certificate
openssl req -new -x509 -days 3650 -key ca-private-key.pem -out cacert.pem
```

3. Use the script below to generate a private key and CSR for the marklogic-0 pod. After running the script, `tls.key` is generated as a private key and a host certificate for the marklogic-0 pod.

NOTE
The filename for the private key must be `tls.key` and the filename for host certificate must be `tls.crt`.

   - If the release name is "marklogic", then the host name for the marklogic-0 pod will be "marklogic-0.marklogic.default.svc.cluster.local".
   - The host name for the marklogic-1 pod will be "marklogic-1.marklogic.default.svc.cluster.local".

```
# Create private key
openssl genpkey -algorithm RSA -out tls.key

# Create CSR for marklogic-0
# Use marklogic-0.marklogic.default.svc.cluster.local as Common Name(CN) for CSR
openssl req -new -key tls.key -out tls.csr

# Sign CSR with private CA
openssl x509 -req -CA cacert.pem -CAkey ca-private-key.pem -in tls.csr -out tls.crt
-days 365
```

4. Use this script below to generate secrets for the host certificate and the CA certificate. Repeat these steps to generate the certificate for the marklogic-1 host and create the secret `marklogic-1-cert` . After running the script, secrets are created for marklogic-0 and marklogic-1. One secret is also created for the private CA certificate.

```
# Generate Secret for marklogic-0 host certificate
kubectl create secret generic marklogic-0-cert --from-file=tls.crt --from-file=tls.key

# Generate Secret for private CA certificate
kubectl create secret generic ca-cert --from-file=cacert.pem
```

5. Once the certificate is created within Kubernetes secrets, add the following section to the `values.yaml` file and follow the instructions outlined in Install the chart.

```
tls:
  enableOnDefaultAppServers: true
  certSecretNames:
    - "marklogic-0-cert"
    - "marklogic-1-cert"
  caSecretName: "ca-cert"
```

3.3.2. Configure a MarkLogic cluster with a temporary certificate

To configure a temporary certificate, simply add the following option to the `values.yaml` file and then follow the instructions outlined in Install the chart.

```
tls:
  enableOnDefaultAppServers: true
```

Access an SSL-enabled server with a temporary certificate

Accessing an SSL-Enabled Server with a temporary certificate requires retrieval of the certificate in order for clients to trust it. Refer to the Accessing an SSL-Enabled Server from a Browser or WebDAV Client of the MarkLogic Security Guide for details.

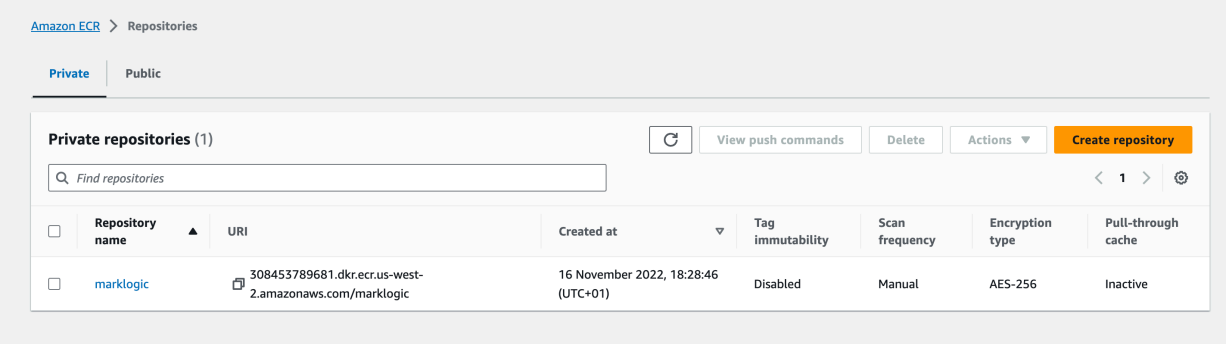
3.4. Test MarkLogic Helm Chart and Docker image from ECR

This section describes how to use MarkLogic Helm Chart and the Docker image from ECR. ECR is an AWS managed container image registry service. It can host any OCI compatible artifact like a Docker image or Helm Chart. For additional information, see the Amazon Elastic Container Registry Documentation.

3.4.1. Create an ECR repository

To create an ECR repository:

1. Navigate to the AWS portal.
2. In the ECR Section, create a private repository.



3.4.2. Push the Docker image and Helm Chart

To push the Docker image and Helm Chart, refer to Pushing a Docker image and Pushing a Helm Chart (AWS documentation).

3.4.3. Deploy the MarkLogic cluster

To deploy the MarkLogic cluster, add the Helm Chart repository and install the MarkLogic cluster.

Add the Helm Chart repository

Login by using:

```
aws ecr get-login-password \
     --region us-west-2 | helm registry login \
     --username AWS \
     --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

Install the MarkLogic cluster

To install the MarkLogic cluster:

1. Add information about the Docker image from ECR to the `values.yaml` file:

```
## Marklogic image parameters
image:
  repository: 308453789681.dkr.ecr.us-west-2.amazonaws.com/marklogic
  tag: 11.1.0-centos-1.1.0
  pullPolicy: IfNotPresent

....

## Configure the imagePullSecrets to pull the image from private repository that
requires credential
imagePullSecrets:
- name: "docker-creds"
# - name: "your-secret-name-2"
```

2. Install the MarkLogic cluster using the helm install command. In this example, a specific `values.yaml` file is used:

```
helm install ml-cluster oci://308453789681.dkr.ecr.us-west-2.amazonaws.com/marklogic
-f values.yaml -n ml
```

3. The command will return a message similar to this:

```
Pulled: 308453789681.dkr.ecr.us-west-2.amazonaws.com/marklogic:1.0.1
Digest: sha256:c3902a1330b0928b7aec1075f16c38c865b9395e5efb0e0eb5314c903fbc40bd
NAME: ml-cluster
LAST DEPLOYED: Thu Oct 12 14:28:21 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
Thank you for installing marklogic.

Your release is named ml-cluster.
```

3.5. Topology spread constraints

Topology spread constraints and the `actualSkew` and `maxSkew` parameters control the spread of pods among worker nodes and zones in a cluster.

- `actualSkew` is the difference between the number of pods in the most populated worker nodes or availability zones, and the number of pods in the least populated worker nodes or availability zones.
- `maxSkew` is the maximum degree to which pods may be unevenly distributed.

For additional information and examples, see GitHub.

The MarkLogic Helm Chart defaults to this configuration:

```
 - maxSkew: 1
   topologyKey: kubernetes.io/hostname
   whenUnsatisfiable: DoNotSchedule
   labelSelector:
     matchLabels:
       app.kubernetes.io/name: marklogic
 - maxSkew: 1
topologyKey: topology.kubernetes.io/zone
   whenUnsatisfiable: ScheduleAnyway
   labelSelector:
     matchLabels:
       app.kubernetes.io/name: marklogic
```

In the first rule, `topologyKey` is set to the hostname. This ensures that MarkLogic pods are scheduled to all the available worker nodes evenly and that `maxSkew` is not exceeded.

In the second rule, the `topologyKey` is set to the zone. This setting attempts to schedule the pods onto worker nodes located in different availability zones. If the `topologyKey` zone has an even distribution, the rule only applies to nodes with the label `zone: <any value>`. Nodes without a zone label are skipped.

When the `actualSkew` of all the nodes exceeds `maxSkew`, the rules are unsatisfied. When the rules are unsatisfied, `whenUnsatisfiable` controls what happen next:

- if `whenUnsatisfiable` is set to `DoNotSchedule`, pods are not scheduled to the worker nodes.
- If `whenUnsatisfiable` is set to `ScheduleAnyway`, pods are scheduled to the worker nodes. Pods are scheduled even if the rule is unsatisfied.

3.6. Retrieve the MarkLogic admin credentials

If credentials were not provided for the admin user when installing the MarkLogic Chart, a randomly generated alphanumeric value was used. This value is stored in Kubernetes Secrets.

NOTE
Custom admin credentials can also be set using the *auth* parameter during installation.

To retrieve the randomly generated admin credentials from Kubernetes Secrets, follow these steps:

1. List the secrets for a MarkLogic deployment by entering this command:

```
kubectl get secrets -n <release-namespace>
```

2. Find the appropriate secret. The secret generated by the Helm Chart has the format `<release-name>-admin`. For example, if `release-name` = `marklogic`, the secret that contains the admin username, password, and wallet password is `marklogic-admin`.
3. Retrieve the encoded credentials by entering this command:

```
kubectl get secret marklogic-admin -n <release-namespace> SECRET_NAME -o
jsonpath='{.data}'
```

4. Use the output to decode the credentials. For example, if the encoded password is `UyFCXCpkJHpEc2I9`, enter this command to decode the password:

```
echo 'UyFCXCpkJHpEc2I9' | base64 --decode
```

5. Repeat the process described in step 4 for the username and wallet password.

3.7. Configuration options for Helm

This section describes Helm configuration options.

3.7.1. values flag

The values flag points to a YAML file. The values in this file will override the default Helm values.

To view the default configuration variables, enter this command:

```
helm show values marklogic/marklogic --version <version>
```

To set different values with a YAML file, follow these steps:

1. Create a `values.yaml` file with custom values as needed. See Helm chart parameters for a list of parameters.
2. After creating the `values.yaml` file, install MarkLogic by entering this command:

```
helm install my-release marklogic/marklogic --version <version> --values values.yaml
-n <release-namespace>
```

3.7.2. set flag

The set flag can be used to make one or more configuration changes directly as shown in this example:

```
helm install my-release marklogic/marklogic --version <version> \

--set imagePullSecret.registry="https://index.docker.io/v1/" \

--set imagePullSecret.username=YOUR_USERNAME \

--set imagePullSecret.password=YOUR_PASSWORD \ -n <release-namespace>
```

NOTE
It is recommended to use the `values.yaml` file for configuring an installation.

3.7.3. High availability and pod anti-affinity

To attempt to provide the highest availability deployment, the MarkLogic Helm Chart provides a default affinity configuration that prefers to schedule one MarkLogic pod per worker node using the preferred rule. However, if a one-MarkLogic-pod-per-worker node configuration must be strictly enforced, the required rule is recommended.

Preferred rule

The preferred rule, `podAntiAffinity: preferredDuringSchedulingIgnoredDuringExecution`, is a softly enforced rule that prefers scheduling MarkLogic pods on different worker nodes:

```
affinity:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 100
      podAffinityTerm:
        labelSelector:
          matchExpressions:
          - key: app.kubernetes.io/name
            operator: In
            values:
            - marklogic
        topologyKey: kubernetes.io/hostname
```

However, the rule will still co-locate the MarkLogic pods if the worker nodes are limited.

Required rule

The strict rule, `podAntiAffinity: requiredDuringSchedulingIgnoredDuringExecution`, is a rigidly enforced rule that requires scheduling MarkLogic pods on different worker nodes:

```
affinity:
  podAntiAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    - labelSelector:
        matchExpressions:
        - key: app.kubernetes.io/name
          operator: In
          values:
          - marklogic
      topologyKey: kubernetes.io/hostname
```

Use this rule, for example, if there is only one worker node but you want to create two MarkLogic pods. In this case, the rule will cause the second pod to remain in pending status until a second worker node with adequate resources is created.

Pods running on different worker nodes and in separate zones

Spreading resources across availability zones is part of the availability equation in the cloud. However, because the MarkLogic Helm Chart may be used in non-cloud environments, there is no default affinity setting that includes zones. To deploy to the cloud and to deploy across zones, include a pod affinity for `topologyKey: topology.kubernetes.io/zone`. This affinity rule prefers scheduling pods to run on different worker nodes and in separate zones:

```
affinity:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 100
      podAffinityTerm:
        labelSelector:
          matchExpressions:
          - key: app.kubernetes.io/name
            operator: In
            values:
            - marklogic
        topologyKey: kubernetes.io/hostname
    - weight: 100
      podAffinityTerm:
        labelSelector:
          matchExpressions:
          - key: app.kubernetes.io/name
            operator: In
            values:
            - marklogic
        topologyKey: topology.kubernetes.io/zone
```

3.7.4. Security context

Security context defines privilege and access control settings for a pod or container. By default, security context for containers is enabled with the `runAsUser`, `runAsNonRoot`, and `allowPrivilegeEscalation` settings. To configure these values for containers, set the `containerSecurityContext` in the `values.yaml` file or use the `--set` flag. You can also add security context settings to the `containerSecurityContext` configuration. See Configure a Security Context for a Pod or Container for additional information.

This is the current configuration:

```
containerSecurityContext:
  enabled: true
  runAsUser: 1000
  runAsNonRoot: true
  allowPrivilegeEscalation: true
```

WARNING

This security context should not be modified. See Known issues and limitations.

3.7.5. Network policy

NOTE

To use network policies, the networking solution used must support NetworkPolicy. Creating a NetworkPolicy resource without a controller that implements it will have no effect. See Prerequisites for further information.

NetworkPolicy can be used to control network traffic flow for applications and to specify how pods should communicate. By default, network policy is disabled in the `values.yaml` file. To enable it, set

the `networkPolicy.enabled` parameter to `true`. Default ports are provided in the settings. Custom rules for the sources of the traffic to the desired ports can also be defined.

The default configuration is that ports 8000-8020 are open.

```
ports:
  - port: 8000
    endPort: 8020
    protocol: TCP
```

3.7.6. Assign pod priority

Pod priority can be used to indicate the significance of a pod compared to other pods. Assigning priority to pods is important to ensure that high-priority pods are not preempted and can use required resources. For example, if a pod cannot be scheduled, the scheduler will attempt to free up resources by evicting lower-priority pods. When enough resources are available, the higher-priority pods can be scheduled.

IMPORTANT

To ensure the availability of the database, it is highly recommended that a `PriorityClass` object with the highest possible value is set for MarkLogic pods. For more details on pod priority and PriorityClass, see Pod Priority and Preemption.

To assign priority for pods, follow these steps:

1. Add a `PriorityClass`. This example shows a `PriorityClass` with a value of 1 million:

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
globalDefault: false
description: "This high priority class should be used for MarkLogic pods only."
```

2. Set `priorityClassName` to one of the added PriorityClassNames through the `values.yaml` file or by using `--set` flag while installing the chart.

3.8. Enable log collection

To enable collection for MarkLogic logs, follow these steps:

1. Set the `LogCollection.enabled` parameter to `true`.
2. Set each parameter in the `logCollection.files` to `true` if you want to track that type of log file or to `false` if you do not. See Helm chart parameters for parameter descriptions.
3. Define an output in the `values.yaml` file.
4. Use Fluent Bit to parse and output all the log files from each pod to the outputs specified in the `values.yaml` file. See Fluent Bit's output documentation for more information on configuring Fluent Bit output with a logging backend.

3.9. Deploy a MarkLogic cluster with multiple groups

To deploy a MarkLogic cluster with multiple groups (for example, separate E and D nodes), either the `bootstrapHostName` and `group.name` must be configured in the `values.yaml` file, or the values provided for these configurations must be set using the `--set` flag while installing Helm Charts. For example, if you want to create a MarkLogic cluster with three nodes in a `dnode` group and two nodes in an `enode` group, start with this Helm command:

```
helm install dnode-group marklogic/marklogic --set group.name=dnode --set replicaCount=3
-n <release-namespace>
```

Once the deployment is complete, a MarkLogic cluster with three hosts will be running. To add the `enode` group and nodes to the cluster, the `bootstrapHostName` must be set to join the existing MarkLogic cluster. The first host in the other group can be used. For this example, set `bootstrapHostName` to `dnode-group-marklogic-0.dnode-group-marklogic-headless.default.svc.cluster.local` with this command:

```
helm install enode-group marklogic/marklogic --set group.name=enode --set
replicaCount=2 --set bootstrapHostName=dnode-group-marklogic-0.dnode-group-marklogic-
headless.default.svc.cluster.local-n <release-namespace>
```

Once the deployment is complete, there will be a new "enode" group with two hosts in the MarkLogic cluster. Each MarkLogic group will have its own chart release. In the example, both dnode groups and enode groups have a chart release. Each group can be handled separately.

4. Access MarkLogic Server in a Kubernetes cluster

You can access MarkLogic Server using native Kubernetes, MarkLogic HAProxy Load Balancer Configuration, an HTTP Connection Through Ingress on an EKS cluster, or an ODBC connection through Ingress in EKS.

4.1. Native Kubernetes

In a native Kubernetes environment, access MarkLogic using the ClusterIP service, DNS record, or port forward.

4.1.1. Use the ClusterIP service

You can use the ClusterIP service to access MarkLogic within a Kubernetes cluster. The ClusterIP service includes Helm Chart installation.

WARNING

The Kubernetes service does not support HTTP-level load balancing and cookie-based session affinity. To support cookie-based session affinity, use HAProxy as the load balancer.

To use the ClusterIP service, follow these steps:

1. Use the command `kubectl get services` to get a list of Kubernetes services. The output will look like this (the actual names may be different):

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	1d
marklogic	**ClusterIP**	**10.109.182.205**	**<none>**	**8000/TCP, 8001/TCP, 8002/TCP**	**1d**
marklogic-headless	ClusterIP	None	<none>	7997/TCP,7998/TCP,7999/TCP,8000/ TCP,8001/TCP,8002/TCP	1d

2. The service you are looking for ends with "marklogic" and CLUSTER-IP <> None. In the example above, marklogic is the service name for the ClusterIP service. The row is shown in bold.

Additional ports

When you create a new application server on MarkLogic, you must add the new server port to `additionalPorts` in the service configuration:

```
## @param service.additionalPorts. Additional ports exposed at the service level.
  ## Example:
  ## - name: app1
  ##   port: 8010
  ##   targetPort: 8010
  ##   protocol: TCP
  additionalPorts:
    - name: app-server1
      port: 8010
      targetPort: 8010
      protocol: TCP
```

4.1.2. Use the DNS record

For each Kubernetes ClusterIP service, a DNS with this format is created:

```
<service-name>.<namespace-name>.svc.cluster.local
```

For example, if the service-name is `marklogic` and the namespace-name is `default`, the DNS URL to access the MarkLogic cluster is `marklogic.default.svc.cluster.local`

Because StatefulSet is used for the MarkLogic deployment, the DNS for individual pods is created based on the headless service:

```
<pod-name>.<headless-service-name>.<namespace-name>.svc.cluster.local
```

For example, if the pod name is `marklogic-0`, then the headless service name is `marklogic-headless` and the namespace-name is `default`. The DNS URL to access the marklogic-0 pod is `marklogic-0.marklogic.default.svc.cluster.local`.

The DNS name can be used to access a MarkLogic cluster or an individual pod if your applications are deployed in the same Kubernetes cluster.

4.1.3. Use the port-forward command

Use the `kubectl port-forward` command to access MarkLogic outside of the Kubernetes cluster. Use it to access either a specific pod or the whole cluster.

Forward to pod

To access each pod directly, use the `kubectl port-forward` command with this format:

```
kubectl port-forward <POD-NAME> <LOCAL-PORT>: <CONTAINER-PORT> -n <release-namespace>
```

For example, enter this command to forward port 8000 from the MarkLogic service to localhost:

```
kubectl port-forward svc/marklogic 8000:8001 -n <release-namespace>
```

This pod can now be accessed from http://localhost:8001.

Forward to service

To access the whole cluster, use the `kubectl port-forward` command with this format:

```
kubectl port-forward svc/<SERVICE-NAME><LOCAL-PORT>:<CONTAINER-PORT> -n <release-
namespace>
```

For example, enter this command to forward ports 8000 from the MarkLogic service to localhost:

```
kubectl port-forward svc/marklogic 8000:8000 -n <release-namespace>
```

This pod can now be accessed via http://localhost:8001.

4.2. HAProxy

MarkLogic clusters need a load balancer to handle load balancing activity. The MarkLogic Helm Chart comes with an internal load balancer based on HAProxy. This section aims to provide clear documentation about the configuration of the HAProxy.

4.2.1. Configuration

The HAProxy configuration is designed to be as dynamic as possible. The configuration works efficiently with MarkLogic and should not require any modification. The configuration is separated into five sections:

- Global
- Default
- Resolver DNS
- Front-ends
- Back-ends

Global section

The default HAProxy is configured to handle 1024 parallel connections. Log output is done through the container `stdout` and can be checked using the `kubectl` log command.

Default section

`timeout` is set at 600s to fit with the default timeout on MarkLogic. By default, the HAProxy will forward the client IP to MarkLogic.

To modify `timeout`, change the configuration in the `values.yaml` file:

```
timeout:
  timeoutClient: 600s
  timeoutConnect: 600s
  timeoutServer: 600s
```

Resolver DNS

This section mainly handles the DNS configuration between HAProxy and MarkLogic server.

Front-end section

HAProxy statistics pages

The statistics page is disabled by default. It can be exposed by providing this configuration in the `values.yaml` file of the chart:

```
stats:
  enabled: false
  port: 1024
  auth:
    enabled: false
    username: ''
    password: ''
```

Then, a dedicated front-end section can be configured.

MarkLogic front-end

Two different front-end types can be configured:

- HTTP
- TCP

The HTTP front-end can be configured using path-based routing or port routing. See Set up and use path-based routing with MarkLogic Helm Chart for configuration information.

NOTE
Path-based routing is only supported in MarkLogic 11.1 and higher.

TCP front-end

By default, HAProxy uses the leastconn algorithm. HAProxy selects the server with the fewest active sessions. The logs are standard TCP output which is the recommended value for a pure TCP connection. The TCP front-end allows the exposure of ODBC servers from MarkLogic.

To add port 5432 for TCP load balancing, add this configuration to the `values.yaml` file:

```
tcpports:
# TCP port has to be explicitely enabled
  enabled: true
  ports:
    - name: odbc
      type: TCP
      port: 5432
```

HTTP front-end

The HTTP front-ends are all configured in the same way. When path-based routing is enabled, only one front-end section is created. Each request is handled by the relevant MarkLogic server based on the path used. When path-based routing is not enabled, then a dedicated front-end section is created for each port exposed.

This is the default configuration:

```
# Path and port used on HAProxy
# The same path will be used on Ingress for Default AppServers

  defaultAppServers:
    appservices:
      path: /console
      port: 8000
    admin:
      path: /adminUI
      port: 8001
    manage:
      path: /manage
      port: 8002
```

To add port 8010 for HTTP load balancing, add this configuration to the `values.yaml` file:

```
additionalAppServers:
  - name: myapp1
    type: HTTP
    port: 8010
    targetPort: 8010
    path: /myapp1
```

Back-end section

HTTP back-ends are all configured in the same way. By default, HAProxy uses the leastconn algorithm. HAProxy selects the server with the fewest active sessions.

Specific cookies are managed to bring session stickiness capability.

- The `HostId` cookie is used for XCC connections.
- The `SessionID` cookie is used for the Java client.
- A dedicated cookie is used to manage requests made through the internet browser.

Cookies are set to expired all 4 hours.

4.3. HTTP connection through Ingress on an EKS cluster

With MarkLogic 11.1 and Helm Chart release 2.0.0, it is possible to expose a MarkLogic cluster using Ingress and path-based routing. This section describes how to do this with the Application Load Balancer (ALB) Ingress Controller.

4.3.1. ALB Ingress

This approach uses the ALB Ingress Controller functionality provided by EKS.

NOTE

This approach will not address ODBC exposition, as ALB Ingress only supports HTTP/HTTPS connections. See ODBC connection through Ingress in EKS for further information.

ALB Ingress limitations

- 100 total rules per application load balancer.
- Typically, only 100 Ingresses per ALB.
  - 5 condition values per rule.
  - 5 wildcards per rule.
  - 5 weighted target groups per rule.
  - Only HTTP/HTTPS protocol.

Install ALB Ingress

To install ALB Ingress, see AWS Load Balancer Controller installation.

NOTE

To use External DNS, see Setup External DNS.

This feature is still in alpha release and should not be used in production.

Ingress definition

When you install ALB Ingress, the Ingress definition automatically creates an ALB.

Configure the paths

To configure the paths, use these values in the `values.yaml` file in the HAProxy section of the Helm Chart:

```
# Used if MarkLogic Default APP-Servers are meant to be exposed under subpath different
from /

#########################################################################
#  IMPORTANT NOTE:                                                      #
#  This feature is only supported with MarkLogic 11.1 and higher.       #
#  See Limitations and known Issues in the README file.                 #
#########################################################################

  pathbased:
    enabled: true

# This the default listening port in the Front-End section of the HAProxy when using Path
based routing
  frontendPort: 443


# Path and port used on HAProxy
# The same path will be used on Ingress for Default AppServers

  defaultAppServers:
    appservices:
      path: /console
      port: 8000
    admin:
      path: /adminUI
      port: 8001
    manage:
      path: /manage
      port: 8002
```

Configure the Ingress definition

The Ingress definition can be configured in the `values.yaml` file:

```
## Configure Ingress

###################################################################################
#  IMPORTANT NOTE:                                                                #
#  Ingress is only supported with MarkLogic 11.1 and higher.                      #
#  See Limitations and known Issues in the README file.                           #
###################################################################################

## ref: https://kubernetes.io/docs/concepts/services-networking/ingress/
ingress:
  enabled: true

  ## Ingress class
  ## ref: https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-class
  className: "alb"

  ## Ingress labels
  ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml

  ## Ingress annotations
  ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '443'
    alb.ingress.kubernetes.io/healthcheck-path: /adminUI
    alb.ingress.kubernetes.io/success-codes: '200-401'
    alb.ingress.kubernetes.io/load-balancer-name: ml
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":443}]'
    alb.ingress.kubernetes.io/target-group-attributes:
load_balancing.algorithm.type=least_outstanding_requests
    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-
west-2:XXXXXXXXXXX:certificate/XxXXXXXX-xxxx-XXXX-XXXX-XXXXXXXxxxxXXX
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
    alb.ingress.kubernetes.io/load-balancer-attributes:
idle_timeout.timeout_seconds=600,routing.http.xff_header_processing.mode=append
```

NOTE
The path definition of the Ingress will be the same as defined in the HAProxy section.

Configuration details

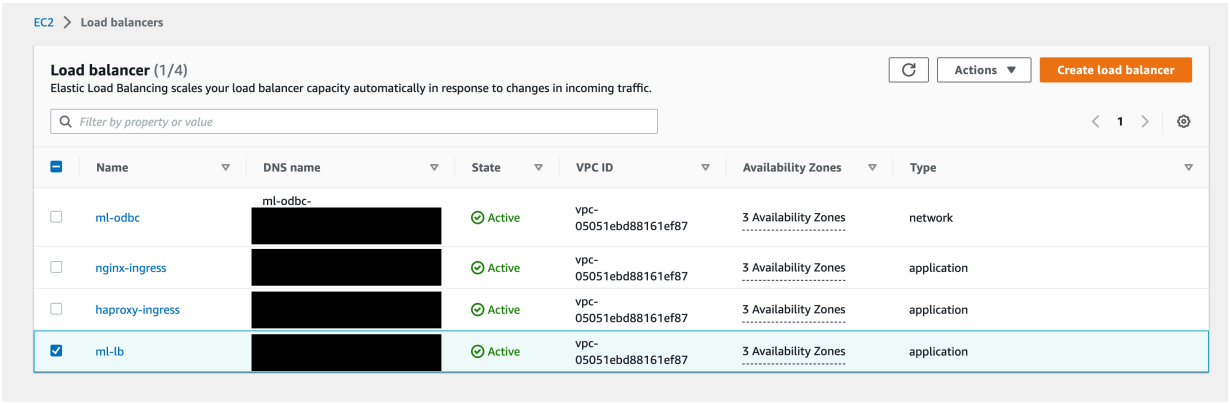
Code	Description
alb.ingress.kubernetes.io/ healthcheck-port	Specifies the port on which to perform the health check. See alb.ingress.kubernetes.io/healthcheck-port.
alb.ingress.kubernetes.io/ healthcheck-path	Specifies the path used for the health check. See alb.ingress.kubernetes.io/healthcheck-path.
alb.ingress.kubernetes.io/load-balancer-name: ml	Specifies the prefix for the name of the ALB. Note that name impacts the entire IngressGroup. See alb.ingress.kubernetes.io/load-balancer-name.

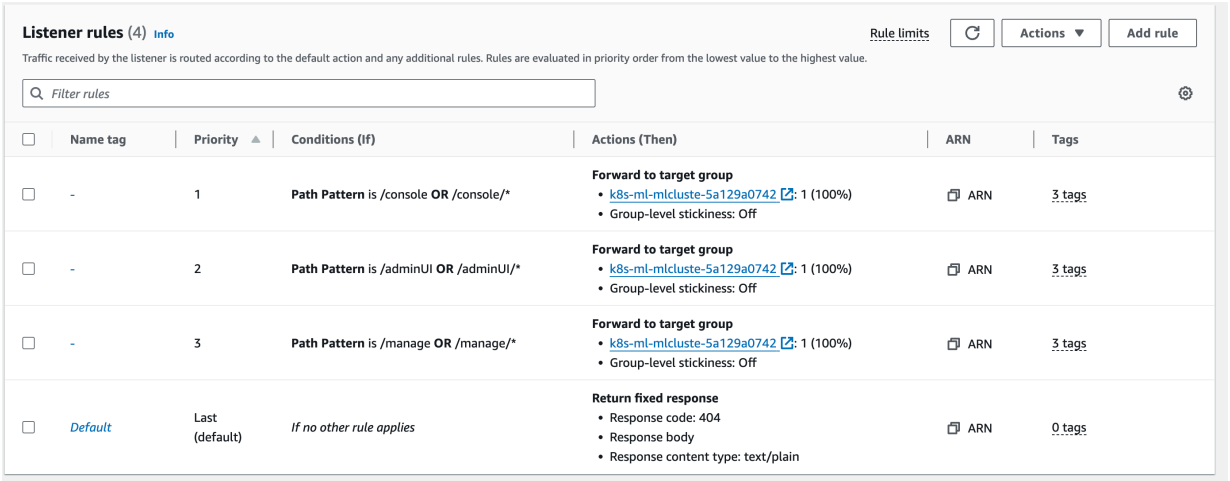
Code	Description
alb.ingress.kubernetes.io/scheme: internet-facing	Specifies the scheme. Because MarkLogic app-servers should be exposed from outside of the cluster and outside of the cloud, this is set it to internet-facing. See alb.ingress.kubernetes.io/scheme.
alb.ingress.kubernetes.io/listen-ports	Specifies the listening port for each Ingress. The syntax is : *<protocol>:<port>*. Protocol can only be HTTP or HTTPS. See alb.ingress.kubernetes.io/listen-ports.
alb.ingress.kubernetes.io/target-group-attributes	Specifies the target group attributes as the load balancing algorithm. See alb.ingress.kubernetes.io/target-group-attributes.
alb.ingress.kubernetes.io/certificate-arn	Specifies the certificates to be used (HTTPS termination on the ALB should be enabled). See alb.ingress.kubernetes.io/certificate-arn.
alb.ingress.kubernetes.io/target-type	Specifies the target type. The target type can be *instance* or *ip*. instance type is only available if the target service is type *NodePort*. See alb.ingress.kubernetes.io/target-type.
alb.ingress.kubernetes.io/group.name	Specifies the group name for the ALB. This allows several Ingresses to use the same ALB. *group.name* and *load.balancer.name* have to be the same in the same IngressGroup. See alb.ingress.kubernetes.io/group.name.

For additional annotations, see the complete list.

Check the ALB

Go to the AWS Console and check what the created ALB looks like:

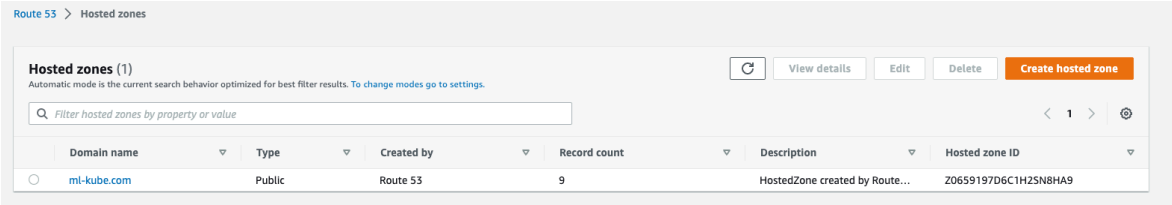




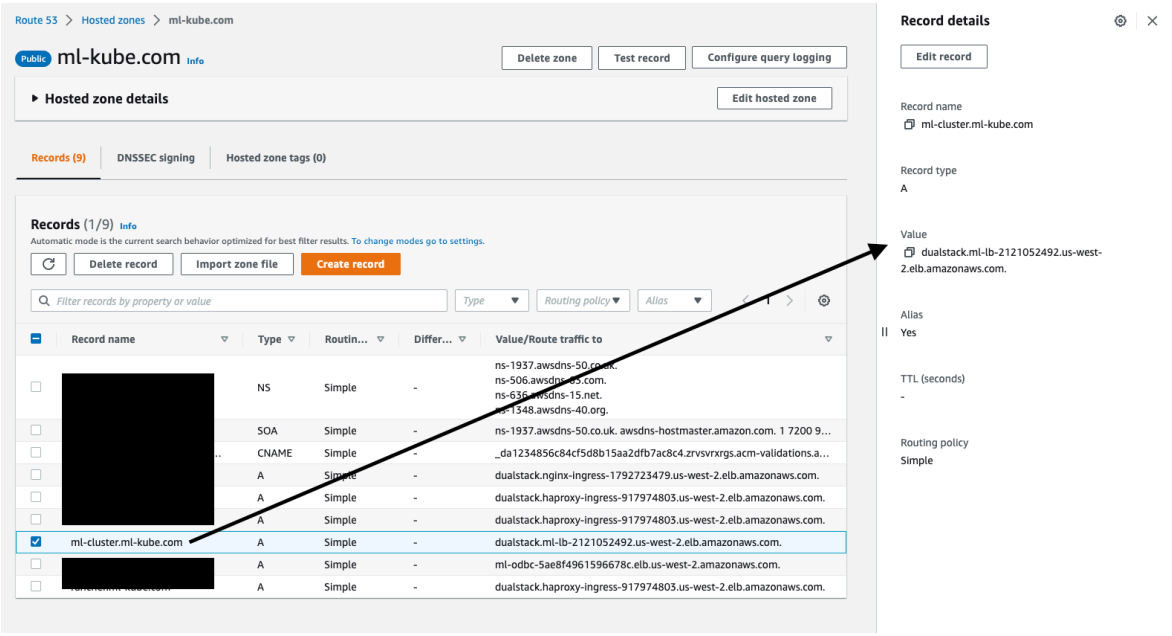
Route53

Because the ALB scheme is specified as internet-facing, the automatically generated DNS name can be used. However, it is more convenient to use a proper DNS name. This is done using Route53:

1. Configure Route53 with one hosted zone:



2. Create a dedicated record to point to the ALB:



4.3.2. Set up and use path-based routing with MarkLogic Helm Chart

Helm Chart release 2.0.0 includes the ability to setup and use path-based routing to access a MarkLogic cluster. This include the capability to use Ingress.

Limitations

Path-based routing and Ingress features are only supported on MarkLogic 11.1 and higher.

Prerequisites

The HAProxy LoadBalancer needs to be enabled to use path base routing and Ingress
See Section HAProxy LoadBalancer.

Path bath routing configuration

Enable path-based routing

Path- based routing is disabled by default. To enable it, use this configuration in the `values.yaml` file for the chart installation:

```
pathbased:
  enabled: true
```

Front-end port configuration

Configure the front-end port to expose the HAProxy in the `values.yaml` file using this code:

```
frontendPort: 443
```

Default App Server path and back-end port

Configure the path and back-end port for the default App Servers in the chart installation `values.yaml` file:

```
defaultAppServers:
  appservices:
    path: /console
    port: 8000
  admin:
    path: /adminUI
    port: 8001
  manage:
    path: /manage
    port: 8002
```

Additional App Server

Include additional App Servers in the HAProxy configuration using the `values.yaml` file:

```
additionalAppServers:
  - name: dhf-jobs
    type: HTTP
    port: 8010
    targetPort: 8010
    path: /DHF-jobs
  - name: dhf-final
    type: HTTP
    port: 8011
    targetPort: 8011
    path: /DHF-final
```

Ingress configuration

Enable Ingress

Ingress routing is disabled by default. To enable it, use this configuration in the `values.yaml` file:

```
ingress:
  enabled: false
```

Paths and ports configuration

The paths and ports only need to be configured in the HAProxy. The Ingress will automatically adapt its configuration to use what is defined there.

Ingress Class

Ingresses can be implemented by different controllers with different configurations. Each Ingress should specify a class in the `values.yaml` file. In this example, the AWS ALB Ingress Controller is specified.

```
className: "alb"
```

Annotations

Some Ingress controllers require specific annotations. This example shows a configuration for an ALB Ingress Controller on AWS. See HTTP connection through Ingress on an EKS cluster for additional information.

```
annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '443'
    alb.ingress.kubernetes.io/healthcheck-path: /adminUI
    alb.ingress.kubernetes.io/success-codes: '200-401'
    alb.ingress.kubernetes.io/load-balancer-name: ml
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":443}]'
    alb.ingress.kubernetes.io/target-group-attributes:
load_balancing.algorithm.type=least_outstanding_requests
    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-
west-2:XXXXXXXXXXX:certificate/XxXXXXXX-xxxx-XXXX-XXXX-XXXXXXXxxxxXXX
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
    alb.ingress.kubernetes.io/load-balancer-attributes:
idle_timeout.timeout_seconds=600,routing.http.xff_header_processing.mode=append
```

Access MarkLogic Cluster

After path-based routing is configured, access the UI using these addresses:

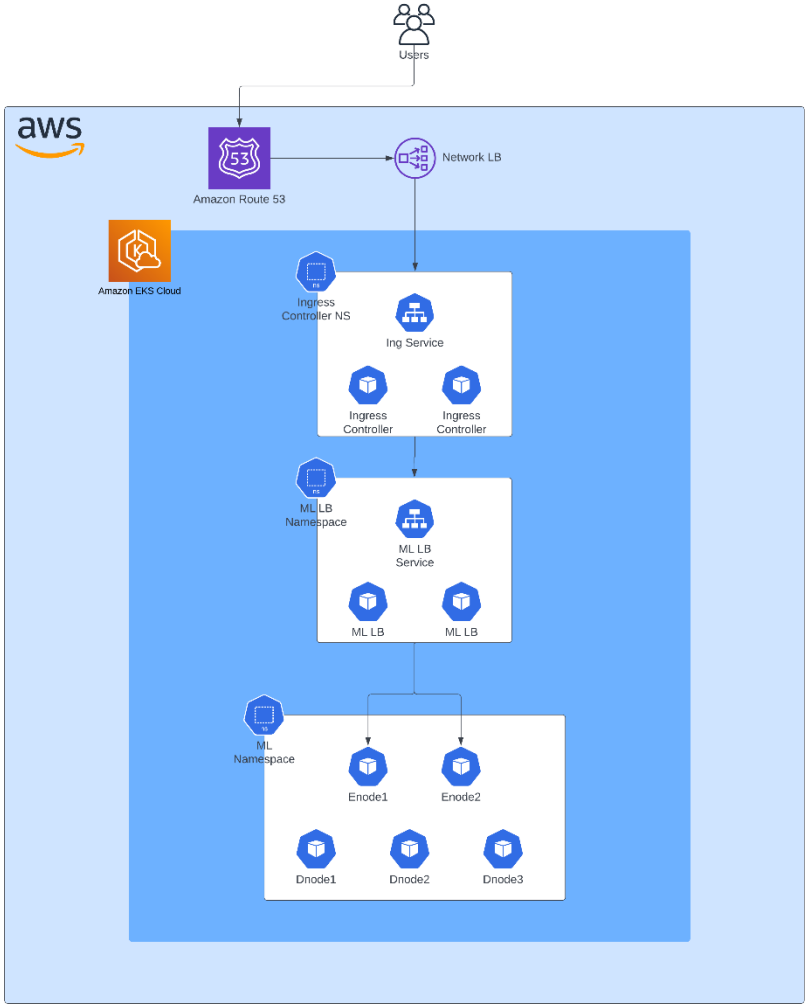
Component	URL
QConsole	https://example.com/qconsole/console
Admin UI	https://example.com/adminUI/
Manage UI	https://example.com/manage/dashboard

4.4. ODBC connection through Ingress in EKS

Ingress does not support TCP or UDP services. However, most Ingress controllers can point to an existing config map where the key is the external port and the value indicates the service to expose. This automatically adds frontend/backend entries to the Ingress controller configuration. To do this, use the `--configmap-tcp-services` argument. Nginx and taefix manage TCP connections using a Kubernetes object called IngressRouteTCP.

This section contains information on exposing an ML ODBC App-server on an EKS cluster using HAProxy as an Ingress controller.

4.4.1. Macro Architecture



4.4.2. MarkLogic ODBC config

The ODBC App server is configured using SQL Quick Start.

4.4.3. MarkLogic HAProxy Load Balancer Configuration

This section describes how to configure the HAProxy Load Balancer.

Configure the ODBC App Server

First, configure the ODBC App Server by providing these values in the `values.yaml` file:

```
## Ports and load balancing type configuration for HAproxy
## There are three types of backends supported:
## 1. HTTP: HTTP(Layer 7) proxy mode. This works for most of the App Servers handling
HTTP connections.
## 2. TCP: TCP(Layer 4) proxy mode. This works for the MarkLogic App Servers handling TCP
connections like ODBC.
ports:
  - name: app-service
    type: HTTP
    port: 8000
    targetPort: 8000
  - name: admin
    type: HTTP
    port: 8001
    targetPort: 8001
  - name: manage
    type: HTTP
    port: 8002
    targetPort: 8002
  - name: odbc
    type: TCP
    port: 5432
```

Auto-generated HAProxy configuration file

After adding the values to the `values.yaml` file, this appears in the auto-generated HAProxy configuration file:

```
## ODBC



frontend ml-cluster-odbc

    description "ml-cluster-odbc"

    mode tcp

    option tcplog

    bind :5432

    use_backend ml-cluster-odbc


backend ml-cluster-odbc

    description "ml-cluster-manage"

    mode tcp

    balance leastconn

    server ml-enode-mlenode-0 ml-enode-mlenode-0.ml-enode-mlenode-
headless.ml.svc.cluster.local:5432 check resolvers dns init-addr none

    server ml-enode-mlenode-1 ml-enode-mlenode-1.ml-enode-mlenode-
headless.ml.svc.cluster.local:5432 check resolvers dns init-addr none
```

Code explanation

In the Auto-generated HAProxy configuration file:

- `mode tcp` - This mode is used because Layer4 LB is only required for ODBC connections.

- `option tcplog` - This allows the log output to be enriched with data from the connection timers, the session status, the connection numbers, the frontend, backend, and server name. The source address and ports can also be included.
- `balance leastconn` - With this configuration, HAProxy selects the servers with the fewest active sessions.

Service

After the HAProxy configuration file is generated, this service is deployed for the HAProxy load balancer:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    meta.helm.sh/release-name: ml-enode
    meta.helm.sh/release-namespace: ml
  labels:
    app.kubernetes.io/instance: ml-enode
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: ml-lb
    app.kubernetes.io/version: "2.7"
    helm.sh/chart: ml-lb-2.7
  name: ml-enode-ml-lb
  namespace: ml-lb
spec:
  clusterIP: 10.100.14.59
  clusterIPs:
  - 10.100.14.59
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: https
  - name: ml-admin
    port: 8001
    protocol: TCP
    targetPort: ml-admin
  - name: ml-manage
    port: 8002
    protocol: TCP
    targetPort: ml-manage
  - name: ml-odbc
    port: 5432
    protocol: TCP
    targetPort: ml-odbc
  - name: ml-query
    port: 8000
    protocol: TCP
    targetPort: ml-query
  - name: stat
    port: 1024
    protocol: TCP
    targetPort: stat
  selector:
    app.kubernetes.io/instance: ml-enode
    app.kubernetes.io/name: ml-lb
  sessionAffinity: None
  type: ClusterIP
```

Code explanation

In the Service code, this section is dedicated to ODBC:

```
 - name: ml-odbc
port: 5432
protocol: TCP
targetPort: ml-odbc
```

NOTE
The service is a standard ClusterIP service.

4.4.4. HAProxy Ingress Controller configuration

The Ingress controller is exposed using NodePort and the `--configmap-tcp-services` functionality.

Service configuration

The HAProxy Ingress controller service is configured using this code:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    meta.helm.sh/release-name: haproxy
    meta.helm.sh/release-namespace: ingress
  labels:
    app.kubernetes.io/instance: haproxy
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: kubernetes-ingress
    app.kubernetes.io/version: 1.8.3
    helm.sh/chart: kubernetes-ingress-1.22.4
  name: haproxy-kubernetes-ingress
  namespace: ingress
spec:
  clusterIP: 10.100.226.75
  clusterIPs:
  - 10.100.226.75
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http
    nodePort: 31080
    port: 80
    protocol: TCP
    targetPort: http
  - name: https
    nodePort: 31443
    port: 443
    protocol: TCP
    targetPort: https
  - name: stat
    nodePort: 31024
    port: 1024
    protocol: TCP
    targetPort: stat
  - name: ml-odbc-tcp
    nodePort: 31032
    port: 5432
    protocol: TCP
    targetPort: 5432
  - name: healthz-tcp
    nodePort: 31042
    port: 1042
    protocol: TCP
    targetPort: 1042
  selector:
    app.kubernetes.io/instance: haproxy
    app.kubernetes.io/name: kubernetes-ingress
  sessionAffinity: None
  type: NodePort
```

Code explanation

In the Service configuration, the ODBC code is:

```
 - name: ml-odbc-tcp
    nodePort: 31032
    port: 5432
    protocol: TCP
    targetPort: 5432
```

The port is exposed using `nodePort: 31032`.

configmap-tcp-services

To create a dedicated configmap:

- Use --configmap-tcp-services with this code:

```
apiVersion: v1
data:
  "5432": ml-lb/ml-enode-ml-lb:5432
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"5432":"ml-lb/ml-enode-
ml-lb:5432"},"kind":"ConfigMap","metadata":{"annotations":{},"labels":{"app.kubernetes.io/
instance":"haproxy","app.kubernetes.io/name":"kubernetes-ingress"},"name":"ml-odbc-
config","namespace":"ingress"}}
  labels:
    app.kubernetes.io/instance: haproxy
    app.kubernetes.io/name: kubernetes-ingress
  name: ml-odbc-config
  namespace: ingress
```

Code explanation

In configmap-tcp-services:

- The configuration is typically done with this code:

```
  data:
  "5432": ml-lb/ml-enode-ml-lb:5432

  <ingress-tcp-sevice-port> : <tcp-service-to-be-exposed-namespace>/<tcp-service-name-to-
  be-exposed>
```

- Port 5432 is exposed at the Ingress controller service level and the ML LB ODBC port is bound.

Ingress controller Helm Chart level

At the Ingress controller Helm Chart level, this was specified in the `values.yaml` file:

```
## Additional command line arguments to pass to Controller
## ref: https://github.com/haproxytech/kubernetes-ingress/blob/master/documentation/
controller.md
extraArgs:
#  - --namespace-whitelist=default
#  - --namespace-whitelist=namespace1
#  - --namespace-blacklist=namespace2
  - --configmap-tcp-services=ingress/ml-odbc-config
  ## Additional tcp ports to expose
  ## This is especially useful for TCP services:
  ## https://github.com/haproxytech/kubernetes-ingress/blob/master/documentation/
controller.md
  tcpPorts:
    - name: ml-odbc
      port: 5432
      targetPort: 5432
      nodePort: 31032
## Controller Service configuration
## ref: https://kubernetes.io/docs/concepts/services-networking/service/
service:
  enabled: true     # set to false when controller.kind is 'DaemonSet' and
controller.daemonset.useHostPorts is true

  type: NodePort    # can be 'ClusterIP', 'NodePort' or 'LoadBalancer'
```

Code explanation

In the Ingress controller Helm Chart level:

- `extraArgs` specifies which configmap is used. The tcp service syntax is:

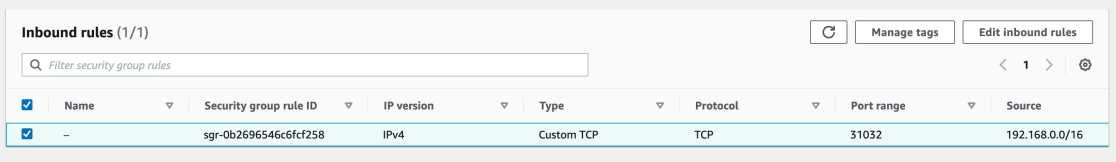
```
- --configmap-tcp-services=<configmap-namespace>/<configmap-name>
```

- `tcpPorts` specifies the additional tcp ports exposed by the Ingress controller service.
- The Ingress controller service is specified as NodePort.

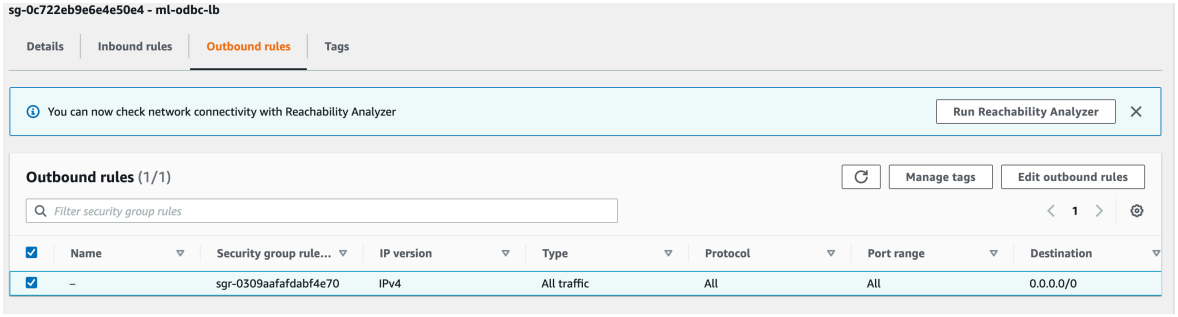
4.4.5. Network load balancer security group

To allow the network Load Balancer to communicate with the worker node on the NodePort, create a dedicated security group:

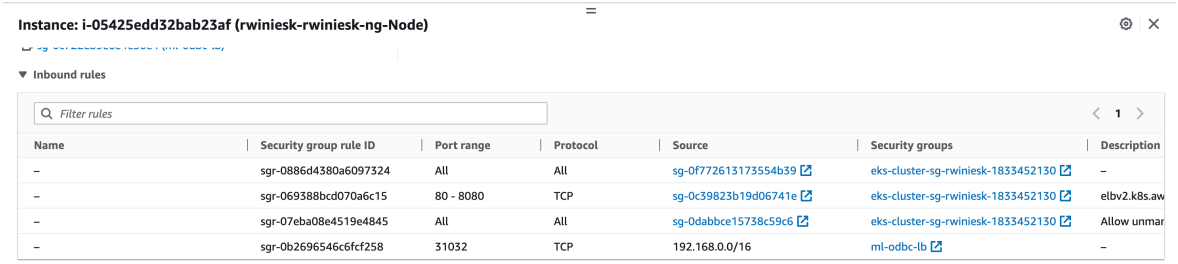
1. First, configure the inbound rules:
   a. Set the port range on the NodePort to 31032
   b. Restrict the source as the CIDR related to the private VPC of the EKS cluster.



2. There are no restrictions on outbound rules:



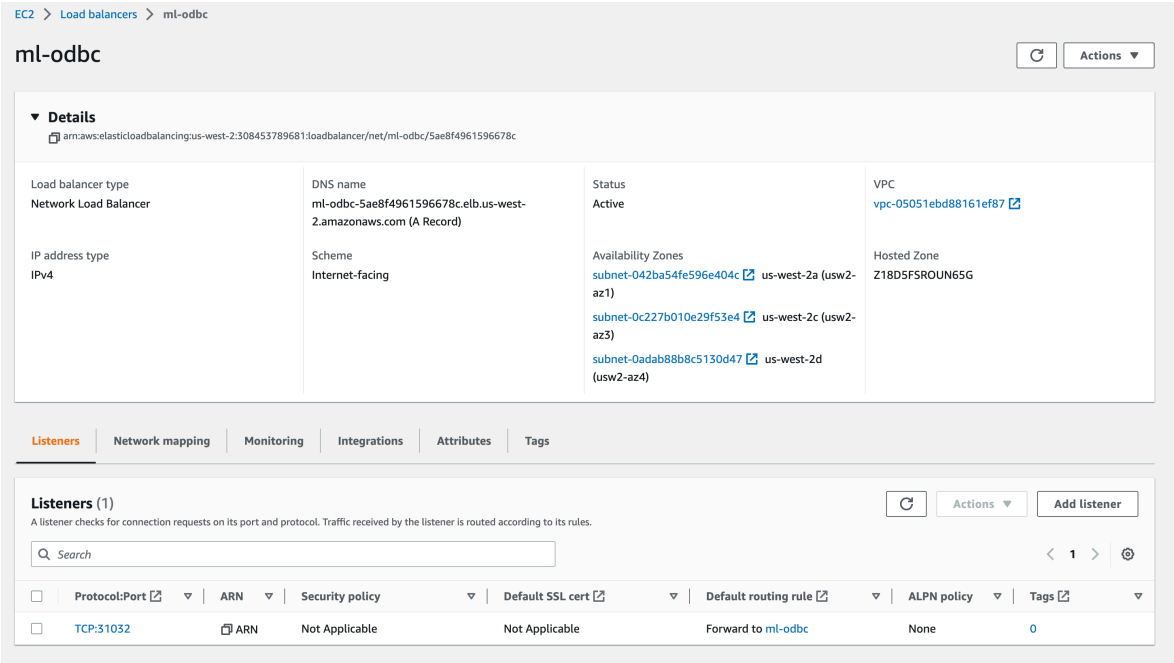
3. Attach the security group to the worker nodes. This must be done for all the worker nodes:



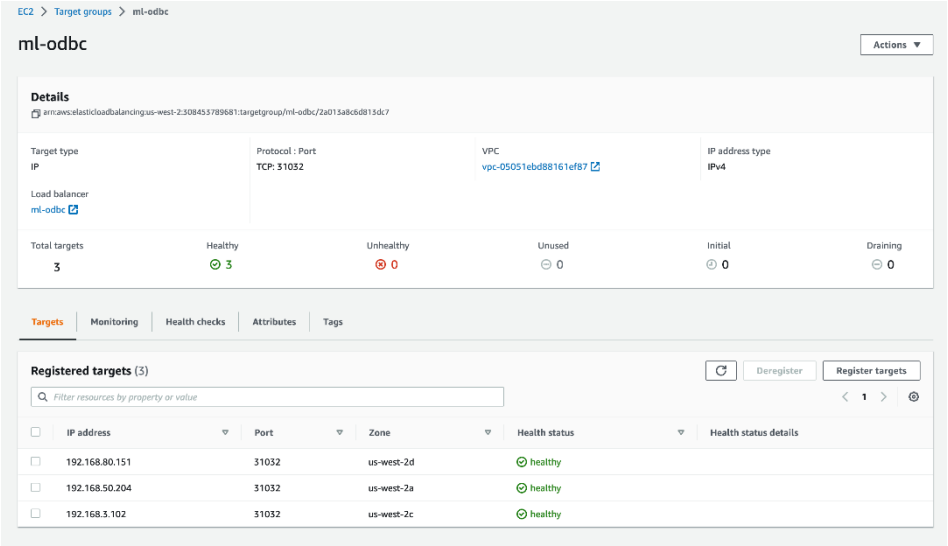
4.4.6. AWS Network Load balancer

To configure the network load balancer:

1. Configure the load balancer to listen on port 31032 (or another tcp port).



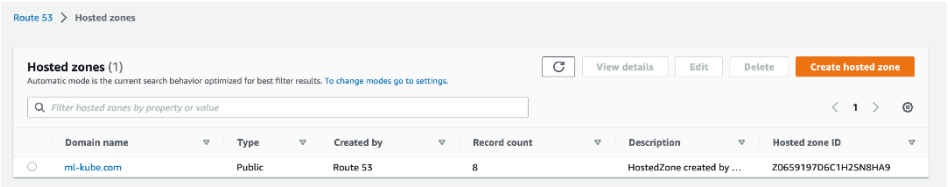
2. Configure the listeners to forward on a specific target group. In this example, the target group has 3 worker nodes registered bind on the NodePort 31032 specific to the Ingress Controller service port for ODBC.
3. Perform a health check by pinging the tcp port specified. In this example, it is protocol port 31302:



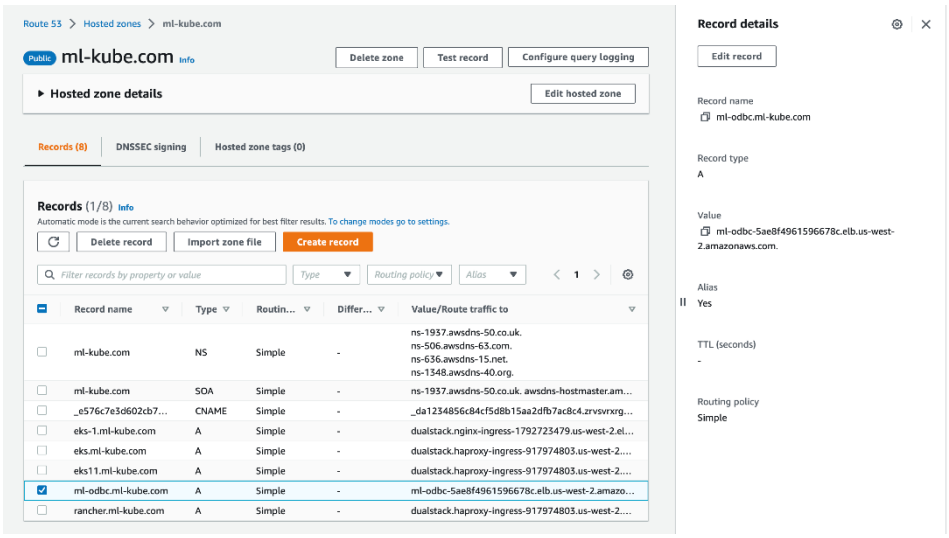
4.4.7. Route53 configuration

To configure Route53:

1. Configure Route53 with `ml-kube.com` as a hosted zone.



2. Specify a dedicated record for ODBC:



3. Use the endpoint `ml-odbc.ml-kube.com:3103` to connect to the ML ODBC.

4.4.8. Connection Test

To configure a connection test:

1. Follow the steps in the ML Knowledge article for setting up ODBC on a Linux environment. Use this configuration:

```
[MarkLogicSQL]
Description=MarkLogicSQL
Driver=MarkLogicSQL
Trace=No
TraceFile=
Database=ml-odbc
Servername=ml-odbc.ml-kube.com
Username=<username>
Password=<password>
Port=31032
Protocol=7.4
ReadOnly=No
SSLMode=disable
UseServerSidePrepare =Yes
ShowSystemTables=No
ConnSettings=
```

2. Connect to the odbc and test a query:

```
[azureuser@marklogic1083 ~]$ isql -v MarkLogicSQL
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL> SELECT employees.FirstName, employees.LastName, SUM(expenses.Amount) AS
ExpensesPerEmployee FROM employees JOIN expenses ON employees.EmployeeID =
expenses.EmployeeID GROUP BY employees.FirstName, employees.LastName ORDER BY
ExpensesPerEmployee;
+-------------+-----------+-------------------------+
| FirstName   | LastName  | ExpensesPerEmployee     |
+-------------+-----------+-------------------------+
| Jane        | Lead         | 155.22                |
| John          | Widget      | 190.97            |
| Debbie      | Goodall   | 259.84                  |
| Steve          | Manager   | 282.95                  |
+-------------+-----------+-------------------------+
SQLRowCount returns -1
4 rows fetched
SQL>
```

5. Maintain a cluster

This section includes information on maintaining a cluster.

5.1. Upgrades

MarkLogic Kubernetes Helm Chart is released in major, minor, and patch releases:

- Major releases may include breaking changes and new features that require configuration changes to the `values.yaml` file. Because of this, review the changes in a release and test the upgrade in a non-production environment.
- Minor and patch releases include bug fixes and other smaller changes.

5.1.1. Recommendations before upgrading

Before you start the upgrade process:

- Read the MarkLogic documentation for details on upgrading MarkLogic.
- Have the latest version of Helm installed.
- Avoid using the `--reuse-values` option with the Helm upgrade to ensure the changes in the new `values.yaml` are merged into your release.
- Always use the `values.yaml` file using the `-f` option and avoid using the `--set` option while installing and upgrading the chart. This ensures your release has all the new values.
- Upgrade the bootstrap host in the MarkLogic StatefulSet before any other node in the cluster. Because of this, the OnDelete upgrade strategy is recommended over the RollingUpgrade strategy. See Update Strategies for more information on upgrade strategies.
- It is important to have a database backup in case of upgrade failure. See Backing Up and Restoring a Database.

5.1.2. Upgrade procedures

This section describes three upgrade procedures.

Upgrade MarkLogic Helm Chart version

When a new version of the MarkLogic Helm Chart is released, upgrade to the new version by following these steps:

1. Update the chart repository to get the new version of the chart:

   ```
   helm repo update
   ```
2. Check the upgrades available for MarkLogic Kubernetes Helm Chart:

   ```
   helm search repo marklogic
   ```
3. Set the upgrade strategy in the `values.yaml` file to `OnDelete`:

   ```
   updateStrategy:
       type: OnDelete
   ```
4. Update the `values.yaml` file with the new values from the updated chart version.
5. Run the Helm upgrade command. Specify the name of your release and the new chart version using the `--version` option. Specify the `values.yaml` file using the `-f` option:

   ```
   helm upgrade <your release> marklogic/marklogic -f values.yaml --version <new
   version> -n <release-namespace>
   ```
6. To start the upgrade, terminate the pod with the smallest ordinal that is running a bootstrap node:

   ```
   kubectl delete pod <pod-name>-n <release-namespace>
   ```

   For example:

```
kubectl delete pod dnode-group-marklogic-0 -n marklogic
```

Once the pod is terminated, a new pod will be created with the updated Helm Chart version.

7. Repeat step 6 for all pods in your release.
8. Complete the upgrade process.

Upgrade Marklogic version in your release

NOTE

- If a cluster is a multi-group MarkLogic cluster, each release corresponding to a group should be upgraded using the following procedure. If all the nodes in the groups are not updated to the same MarkLogic version, then differences in the version and effective version of the MarkLogic cluster will exist.
- MarkLogic Kubernetes Helm Chart releases are independent of MarkLogic Server releases. An upgrade may be required when there is a new MarkLogic Server version available.
- MarkLogic Server also uses the major, minor, and patch release classification. For further information, see MarkLogic Upgrade Support.

To upgrade the MarkLogic version in your release, follow these steps:

1. Update the `image.repository` and `image.tag` in the `values.yaml` file to the version of MarkLogic to upgrade to:

```
image:
  repository: marklogicdb/marklogic-db
  tag: <new tag>
```

2. Set `upgradeStrategy` in the `values.yaml` file to `OnDelete`:

```
updateStrategy:
    type: OnDelete
```

3. Upgrade the Helm Chart using the `helm upgrade` command with the release name, chart name, and `values.yaml`:

```
helm upgrade <release-name> <chart-name> -f <values.yaml> --version <chart-version>
-n <release-namespace>
```

4. Use this command to start the upgrade by deleting the pod with the smallest ordinal that is a MarkLogic bootstrap host:

```
kubectl delete pod <pod-name> -n <release-namespace>.
```

For example,:

```
kubectl delete pod dnode-group-marklogic-0.
```

5. Once the pod is terminated, a new pod will be created with an updated MarkLogic version. New values will also be updated in the `values.yaml` file.
6. To complete the upgrade, repeat the termination process for all the pods in your release. After all the pods are upgraded, access the Admin UI on the bootstrap host and check that there is a configuration and/or a security database upgrade and/or an effective version change. If there is, a prompt to click OK to upgrade appears. If the prompt does not appear, the process is finished.
7. Verify the upgrade by checking the version of MarkLogic on the Admin UI or by accessing the server logs. Required tests can now be run.

Upgrade MarkLogic and the Helm Chart at the same time

1. To upgrade the MarkLogic and Helm Chart versions at the same time, follow steps 1-5 in Upgrade MarkLogic Helm Chart version and steps 1-3 in Upgrade Marklogic version in your release.
2. Next, initiate terminating the pods. First, delete pod-0 (the pod running the MarkLogic bootstrap host). Then delete the other pods using a command like this one:

   ```
   kubectl delete pod <pod-name> -n <release-namespace>.
   ```

   For example:

   ```
   kubectl delete pod dnode-group-marklogic-0 -n marklogic.
   ```
3. Monitor the pod status with this command:

   ```
   kubectl get pods --nampespace=<your-namespace> -w
   ```
4. As soon as all pods are back running, verify the upgrade by checking the version or by running the required tests.

5.1.3. Upgrading the MarkLogic root image to rootless

To upgrade the MarkLogic image from root to rootless, follow these steps:

1. Set the `rootToRootlessUpgrade` flag in the `values.yaml` to true:

   ```
   rootToRootlessUpgrade: true
   ```
2. Update the image.tag in the `values.yaml` file to the rootless MarkLogic version to upgrade to:

   ```
   repository: marklogicdb/marklogic-db
   tag: <11.2.0-ubi-rootless>
   ```
3. Upgrade the Helm Chart using the `helm upgrade` command with the release name, chart name, and values.yaml:

   ```
   helm upgrade <release-name> <chart-name>  -n <release-namespace>
   ```
4. Use this command to start the upgrade. The command will delete the pod with the smallest ordinal that is a MarkLogic bootstrap host:

   ```
   kubectl delete pod <pod-name> -n <release-namespace>
   ```

   For example:

   ```
   kubectl delete pod dnode-group-marklogic-0
   ```
5. Once the pod is terminated, a new pod is created with a rootless MarkLogic version. Monitor the pod status with this command:

   ```
   kubectl get pods --nampespace=<your-namespace> -w
   ```
6. As soon as all pods are back running, verify the upgrade by checking the permissions on volume mounts or by running the required tests.

NOTE

If `rootToRootlessUpgrade` is set to `true` and the image tag is not rootless, then this error message is displayed:

```
ERROR: Root to Rootless Upgrade is supported only if
rootToRootlessUpgrade flag is true and image type is
rootless
```

5.2. Add and remove hosts

This section describes how to add and remove hosts from clusters.

5.2.1. Add and remove hosts

The MarkLogic Helm Chart creates one MarkLogic "host" per Kubernetes pod in a StatefulSet. To add a new MarkLogic host to an existing cluster, simply increase the number of pods in your StatefulSet.

For example, to change the host count of an existing MarkLogic cluster from 2 to 3, follow these steps:

1. Enter this Helm command:

```
helm upgrade release-name marklogic/marklogic --version <version> --namespace
<release-namespace> -set replicaCount=3
```

2. Once this deployment is complete, the new MarkLogic host joins the existing cluster.
3. To track deployment status, use the `kubectl get pods` command.

NOTE

This procedure does not automatically create forests on the new host. If the host will be managing forests for a database, create the forests using MarkLogic's Admin UI or APIs once the pod is up and running.

5.2.2. Remove hosts

When scaling down a StatefulSet, Kubernetes attempts to stop one or more pods in the set to achieve the desired number of pods. However, the storage attached to the pod remains until the persistent volume claims are deleted.

Shutting down a pod from Kubernetes does not modify the MarkLogic cluster configuration; it merely stops the pod. Stopping the pod causes the MarkLogic host to go offline. If there are forests assigned to the stopped hosts, the associated forests will go offline.

5.2.3. Scale down the MarkLogic hosts

The procedure to scale down the number of MarkLogic hosts in a cluster varies depending on whether forests are assigned to the hosts and whether the hosts will be permanently removed from the MarkLogic cluster.

For example, after migrating forest data from the third MarkLogic host, change the host count on an existing MarkLogic cluster from 3 to 2 by running the following Helm command:

```
helm upgrade release-name marklogic/marklogic --version <version> --namespace <release-
namespace> --set replicaCount=2
```

Before Kubernetes stops the pod, it makes a call to the MarkLogic host to shut down with the `fastFailOver` flag set to `true`. This tells the remaining hosts in the cluster that this host is shutting down. It also triggers failover for any replica forests available on this host. There is a two-minute grace period to allow MarkLogic to shut down cleanly before Kubernetes kills the pod.

Track shutdown progress

To track the host shutdown progress, run this command:

```
kubectl logs pod/terminated-host-pod-name -n <release-namespace>
```

Permanently remove the host

If the host should be permanently removed from the MarkLogic cluster, once the pod is terminated, follow the procedure in "Recovery - Step 3: Remove dead host configuration" in the MarkLogic Knowledgebase article Replacing a failed MarkLogic node in a cluster: a step by step walkthrough.

WARNING
Before attempting to scale the hosts in the StatefulSet back up, persistent volume claims and persistent volumes must be manually deleted using the Kubernetes API.

To delete the persistent volumes and persistent volume claims of the terminated host, follow these steps:

1. Get the persistent volume claims:

```
kubectl get pvc datadir-<terminated-host-pod-name> -n <release-namespace>
```

2. Delete the persistent volume:

```
kubectl delete pv <volume name from get pvc command>
```

3. Delete the persistent volume claims:

```
kubectl delete pvc datadir-<terminated-host-pod-name> -n <release-namespace>
```

5.2.4. Enable SSL over XDQP

To enable SSL over XDQP, set *enableXdqpSsl* to `true` either in the `values.yaml` file or by using the `--set` flag. All communications to and from hosts in the cluster will be secured. With this setting enabled, default SSL certificates will be used for XDQP encryption. By default, SSL over XDQP is activated in the Helm Chart.

NOTE
To enable other XDQP/SSL settings, like `xdqp ssl allow sslv3`, `xdqp ssl allow tls`, and `xdqp ssl ciphers`, use the MarkLogic REST Management API.

5.3. Backup and restore a database

When backing up MarkLogic to the file system, a dedicated volume should be allocated for each MarkLogic host. This can be done by adding `additionalVolumes` in the `values.yaml` file:

```
## Specify additional list of persistent volume claims
additionalVolumeClaimTemplates:
   - metadata:
      name: "backup-dir"
    spec:
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 10Gi
## specify additional list of volumes
additionalVolumes:
   - name: "backup-dir"
    emptyDir: {}
## specify additional list of volumeMounts
additionalVolumeMounts:
  - name: "backup-dir"
    mountPath: "/space"
```

Once the `values.yaml` file is modified, `/space` can be used as the backup directory for backing up and restoring a database using the procedures described in the MarkLogic documentation.

5.4. Extend the data volumes

Volume expansion is only available if the underlying StorageClass has the option `allowVolumeExpansion` set to `true`. See Expanding Persistent Volumes Claims for more information, including a list of volume types supported.

After StatefulSet objects are created, the only items that can be modified are the number of replicas, the update strategy, and the object template. Attempting to modify any other specifications returns this error:

```
# * spec: Forbidden: updates to statefulset spec for fields other than
‘replicas’, ‘template’, and ‘updateStrategy’ are forbidden.
```

5.4.1. Expand the volume without downtime

To expand the volume without downtime, follow these steps:

1. Delete the StatefulSet set without deleting the pods by entering this command:

   ```
   kubectl delete sts <statefulset-name>--cascade=orphan -n <release-namespace>
   ```

   NOTE
   This will cause orphan pods. However, there will not be any downtime.

2. Modify each PVC with the desired size by entering this command:

   ```
   kubectl edit pvc <pvc-name> -n <release-namespace>
   ```

   This output appears:

   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     annotations:
   ...
     labels:
       app.kubernetes.io/instance: huge-ml
       app.kubernetes.io/name: marklogic
     name: datadir-huge-ml-marklogic-0
     namespace: ml
   spec:
     accessModes:
     - ReadWriteOnce
     resources:
       requests:
         storage: 80Gi (old size 20Gi)
     storageClassName: gp3
   ```

3. Recreate the StatefulSet with the new storage request. First, modify the `values.yaml` used to deploy the ML-cluster:

```
# Configure persistence using persistent Volume Claim
# ref: https://kubernetes.io/docs/concepts/storage/persistent-volumes
  /#persistentvolumeclaims
# The "" storageClass will use the default storage class for your cluster.
  (gp2 for EKS, standard for Minikube)
# If set the enabled to false, it will use EmptyDir
 volumepersistence:
  enabled: true
  storageClass: "gp3"
  size: 80Gi<---New size
  annotations: {}
  accessModes:
    - ReadWriteOnce
  mountPath: /var/opt/MarkLogic
```

4. Next, upgrade the Helm Chart by entering this command:

```
helm upgrade <release name> -n <release-namespace> marklogic --version <version> - f
<path-to-values-file>
```

5.5. Huge pages

This section explains setting up and using huge pages. For additional information, see the Kubernetes documentation.

WARNING

To increase performance and efficiency, disable transparent huge pages before following the steps in this section. Especially, on nodes with high memory utilization.

5.5.1. Set huge pages at the node level

Huge pages are configured by setting the kernel parameter `vm.nr_hugepages`. This parameter can be set using DaemonSet:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: sysctl-hugepages
  namespace: kube-system
  labels:
    k8s-app: sysctl-hugepages
spec:
  selector:
    matchLabels:
      name: sysctl-hugepages
  template:
    metadata:
      labels:
        name: sysctl-hugepages
    spec:
      tolerations:
      # these tolerations are to have the daemonset runnable on control plane nodes
      # remove them if your control plane nodes should not run pods
      - key: node-role.kubernetes.io/control-plane
        operator: Exists
        effect: NoSchedule
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
      containers:
      - name: sysctl
        image: busybox
        command: ["/bin/sh"]
        args: ["-c", "sysctl -w vm.hugetlb_shm_group=100; sysctl -w vm.nr_hugepages=1280;
tail -f /dev/null"]
        securityContext:
          privileged: true
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
      terminationGracePeriodSeconds: 30
      # these nodeSelector is to have the daemonset only running on specific nodes
hosting ML pods
      # adapt value if necessary
      nodeSelector:
        role: ml-worker
```

The docker image used is the standard busybox.

5.5.2. Arguments

Huge pages are set with these arguments:

- args: ["-c", "sysctl -w vm.hugetlb_shm_group=100; sysctl -w vm.nr_hugepages=1280; tail -f /dev/null"]
- vm.hugetlb_shm_group=100 (gid of default ml user)
- vm.nr_hugepages=1280

NOTE

Linux huge pages should be set at 3/8 the size of physical memory.

5.5.3. Set privileged to true

Set the `securityContext: privileged` setting to true as shown below (and in Set huge pages at the node level).

```
containers:
      - name: sysctl
        image: busybox
        command: ["/bin/sh"]
        args: ["-c", "sysctl -w vm.hugetlb_shm_group=100; sysctl -w vm.nr_hugepages=1280;
tail -f /dev/null"]
        securityContext:
          privileged: true
```

5.5.4. Kubelet restart

In order for Kubernetes to account for the huge pages, the kubelet on each involved node has to be restarted. After restarting and applying the DaemonSet, the `HugePages_Total` = 1280.

```
# cat /proc/meminfo | grep -i hug
AnonHugePages:    124928 kB
ShmemHugePages:        0 kB
FileHugePages:         0 kB
HugePages_Total:    1280
HugePages_Free:     1280
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:         2621440 kB
```

When the kubelet is not restarted, the new configuration is not taken into account. The `kubectl describe node command` indicates that hugepages-1Gi and 2Mi = 0

```
Capacity:
  attachable-volumes-aws-ebs:  25
  cpu:                         8
  ephemeral-storage:           104845292Ki
  hugepages-1Gi:               0
  hugepages-2Mi:               0
  memory:                      32408692Ki
  pods:                        58
Allocatable:
  attachable-volumes-aws-ebs:  25
  cpu:                         7910m
  ephemeral-storage:           95551679124
  hugepages-1Gi:               0
  hugepages-2Mi:               0
  memory:                      31391860Ki
  pods:                        58
```

After the kubelet has been restarted, `hugepages-2Mi` has a value.

```
Capacity:
  attachable-volumes-aws-ebs:  25
  cpu:                         8
  ephemeral-storage:           104845292Ki
  hugepages-1Gi:               0
  hugepages-2Mi:               2560Mi
  memory:                      32408692Ki
  pods:                        58
Allocatable:
  attachable-volumes-aws-ebs:  25
  cpu:                         7910m
  ephemeral-storage:           104845292Ki
  hugepages-1Gi:               0
  hugepages-2Mi:               2560Mi
  memory:                      28770420Ki
  pods:                        58
```

Huge pages can now be allocated at the pod level.

5.5.5. Set huge pages for MarkLogic StatefulSet

The use of huge pages in a namespace is controlled with ResourceQuota similar to other compute resources like `cpu` or `memory` using the `hugepages-<size>` token.

NOTE
Huge pages do not support overcommit

In the `values.yaml` file, huge pages can be set:

```
## Manage HugePages
## ref: https://v1-23.docs.kubernetes.io/docs/tasks/manage-hugepages/scheduling-hugepages/
hugepages:
  enabled: true
  mountPath: /dev/hugepages


resources:
# Marklogic pods' resource requests and limits
# ref: https://kubernetes.io/docs/user-guide/compute-resources/
  limits:
    hugepages-2Mi: 1Gi
    memory: 8Gi
  requests:
    memory: 8Gi
```

Check the error log

After setting the huge pages, check the error log to verify that the huge pages are detected. You should see an entry in the log indicating the number of huge pages detected:

```
2023-02-06 16:01:40.190 Info: Linux Huge Pages: detected 1280, using 1280, recommend 1280
to 1820
```

Check resource usage at the node level

To verify huge pages are working as expected, can check the resource usage,

```
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource                    Requests      Limits
  --------                    --------      ------
  cpu                         915m (11%)    2100m (26%)
  memory                      9506Mi (33%)  11770Mi (41%)
  ephemeral-storage           0 (0%)        0 (0%)
  hugepages-1Gi               0 (0%)        0 (0%)
  hugepages-2Mi               1Gi (40%)     1Gi (40%)
  attachable-volumes-aws-ebs  0             0
```

5.6. Uninstall the chart

To uninstall the Helm Chart, follow these steps:

1. Enter this command:

   ```
   helm uninstall my-release -n <namespace-release>
   ```

   `release "my-release" uninstalled` appears.
2. Verify the uninstall was successful with this command:

   ```
   helm list --all-namespaces
   ```

   An entry named "my-release" (or the release name you chose) should no longer appear.
3. Manually delete the persistent volume claims:

   ```
   kubectl delete pvc -n <namespace-release> -l app.kubernetes.io/name=marklogic
   ```

6. MarkLogic Content Pump (mlcp) in Kubernetes

MarkLogic Content Pump (mlcp) is a powerful tool used for ingesting data into a MarkLogic database. This section describes how to run mlcp in Kubernetes.

NOTE
Before following the steps in this section, configure a Kubernetes cluster and ensure that it is accessible. mlcp can either be run within or outside of the cluster.

6.1. mlcp inside a Kubernetes cluster

To run mlcp inside a Kubernetes cluster:

- Generate a Docker image with mlcp
- Deploy the mlcp pod to the Kubernetes cluster

6.2. Generate a Docker image with mlcp

To build an image containing mlcp, prepare a Docker file or use the image generated by our development team: `mdweller5/theswamp:mlcp`.

6.3. Deploy the mlcp pod to the Kubernetes cluster

The next step is to create a Kubernetes deployment YAML file named `mlcp.yaml` and `persistentVolumeClaim` for storage. A sample file is included below:

```
apiVersion: v1
kind: Pod
metadata:
  name: mlcp
spec:
  volumes:
    - name: mlcp-volume
      persistentVolumeClaim:
        claimName: mlcp-pvc
  containers:
  - name: mlcp
    image: mdweller5/theswamp:mlcp
    command:
      - bash
      - '-c'
      - |
        tail -f /dev/null
    volumeMounts:
      - mountPath: "/data"
        name: mlcp-volume
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mlcp-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
```

6.4. Kubectl Apply

Once `mlcp.yaml` is created, use the `kubectl apply` command to deploy `mlcp.yaml` to the Kubernetes cluster alongside MarkLogic. Verify that the mlcp pod is successfully deployed using the `kubectl get pods` command.

6.5. Access the mlcp pod

Once the mlcp pod has been deployed, use this command to access the pod within the cluster:

```
kubectl exec mlcp -- /bin/bash
```

mlcp is located under the path `/mlcp`. Use `export PATH=${PATH}:/mlcp/bin` to run the `mlcp.sh` command from any location.

NOTE

The `/data` directory is mounted with a persistent volume which provides a storage location for the ingested data. Additional volumes can also be mounted in the `volumeMounts` mlcp pod.

6.6. mlcp outside a Kubernetes cluster

Because the fully qualified domain name for MarkLogic hosts is not accessible outside a Kubernetes cluster, use HAProxy with the LoadBalancer service to access MarkLogic hosts. Please refer to MarkLogic HAProxy Load Balancer Configuration for additional information.

After the HAProxy with the LoadBalancer server is configured with MarkLogic, use the `kubectl get services` command to find the external IP of the server with the name ending with `-haproxy` as the host for mlcp.

6.7. Run mlcp to ingest data

Run mlcp to ingest data. The example below shows an options file configured to ingest a CSV file. Once the file is ready, use the command `mlcp.sh -options_file import.txt` to ingest the file. For additional information on data ingestion, see the mlcp User Guide.

```
import
-username
your_username
-password
your_password
-host
marklogic-0.marklogic-headless.default.svc.cluster.local,marklogic-1.marklogic-
headless.default.svc.cluster.local
-port
8000
-document_type
json
-input_file_path
data.txt
-input_file_type
delimited_text
```

6.7.1. -host setting

Provide the fully qualified domain name of all MarkLogic hosts to the `-host` setting.

NOTE

if you use a Load Balancer like HAProxy for `-host` , then `-restrict_hosts` needs to set to `true`.

7. Helm chart parameters

Name	Description	Default Value
replicaCount	Number of MarkLogic nodes	1
updateStrategy.type	Update strategy for MarkLogic pods	OnDelete
terminationGracePeriod	Seconds before the MarkLogic pod terminates gracefully	120
clusterDomain	Domain for the Kubernetes cluster	cluster.local
allowLongHostnames	Indicates whether to allow deployment with hostnames over 64 characters	false
useLegacyHostnames	Use the legacy hostnames used before the 1.1.0 version	false
podAnnotations	Pod annotations	{}
group.name	Group name for joining MarkLogic cluster	Default
group.enableXdqpSsl	SSL encryption for XDQP	true
bootstrapHostName	Host name of MarkLogic bootstrap host (to join a cluster)	""
image.repository	Repository for MarkLogic image	progressofficial/marklogic-db
image.tag	Image tag for MarkLogic image	11.3.0-ubi-rootless
image.pullPolicy	Image pull policy for MarkLogic image	IfNotPresent
initContainers.configureGroup.image	Image for configureGroup InitContainer	curlimages/curl:8.8.0
initContainers.configureGroup.pullPolicy	Pull policy for configureGroup InitContainer	IfNotPresent
initContainers.utilContainer.image	Image for copyCerts and volume permission change for root to rootless upgrade InitContainer	redhat/ubi9:9.4
initContainers.utilContainer.pullPolicy	Pull policy for copyCerts and volume permission change for root to rootless upgrade InitContainer	IfNotPresent
imagePullSecrets	Registry secret names as an array	[]
hugepages.enabled	Parameter to enable Hugepages on MarkLogic	false
hugepages.mountPath	Mountpath for Hugepages	/dev/hugepages
resources	The resource requests and limits for MarkLogic container	{}
nameOverride	String to override the app name	""

Name	Description	Default Value
fullnameOverride	String to completely replace the generated name	""
auth.secretName	Kubernetes Secret name for MarkLogic Admin credentials	""
auth.adminUsername	Username for default MarkLogic Administrator	""
auth.adminPassword	Password for default MarkLogic Administrator	""
auth.walletPassword	Password for wallet	""
tls.enableOnDefaultAppServers	Parameter to enalbe TLS on Default App Servers (8000, 8001, 8002)	false
tls.certSecretNames	Names of the secret that contains the named certificate	[]
tls.caSecretName	Name of the secret that contains the CA certificate	""
enableConverters	Parameter to Install converters for the client if they are not already installed	false
license.key	Used to set the MarkLogic license key installed	""
license.licensee	Used to set the MarkLogic licensee information	""
affinity	Affinity for MarkLogic pods assignment	{}
topologySpreadConstraints	POD topology spread constraints to spread pods across the cluster	[]
nodeSelector	Node labels for MarkLogic pods assignment	{}
persistence.enabled	Parameter to enable MarkLogic data persistence using Persistence Volume Claim (PVC). If set to false, EmptyDir will be used.	true
persistence.storageClass	Storage class for MarkLogic data volume. Leave this parameter empty to use the default storage class	""
persistence.size	Size of storage request for MarkLogic data volume	10Gi
persistence.annotations	Annotations for Persistence Volume Claim (PVC)	{}
persistence.accessModes	Access mode for persistence volume	["ReadWriteOnce"]
additionalVolumeClaimTemplates	List of additional volumeClaimTemplates to each MarkLogic container	[]

Name	Description	Default Value
additionalVolumes	List of additional volumes to add to the MarkLogic containers	[]
additionalVolumeMounts	List of mount points for the additional volumes to add to the MarkLogic containers	[]
additionalContainerPorts	List of ports in addition to the defaults exposed at the container level This does not typically need to be updated. Use `service.additionalPorts` to expose app server ports	[]
service.annotations	Annotations for MarkLogic service	{}
service.type	Default service type	ClusterIP
service.additionalPorts	List of ports, in addition to the defaults exposed at the service level	[]
serviceAccount.create	Parameter to enable creating a service account for a MarkLogic Pod	true
serviceAccount.annotations	Annotations for MarkLogic service account	{}
serviceAccount.name	Name of the serviceAccount	""
priorityClassName	Name of a PriortyClass defined to set pod priority	""
networkPolicy.enabled	Parameter to enable network policy	false
networkPolicy.customRules	Placeholder to specify selectors	{}
networkPolicy.ports	Parameter to specify the ports where traffic is allowed	[{port:8000, endPort: 8020, protocol: TCP}]
podSecurityContext.enabled	Parameter to enable security context for a pod running MarkLogic containers	true
podSecurityContext.fsGroup	Parameter to specify the group id for a mounted data volume	2
podSecurityContext.fsGroupChangePolicy	Parameter to specify how the volume ownership should be changed when a pod's volumes needs to be updated with an fsGroup	OnRootMismatch
containerSecurityContext.enabled	Parameter to enable security context for MarkLogic containers	true
containerSecurityContext.runAsUser	User ID to run the entrypoint of the container process	1000
containerSecurityContext.runAsNonRoot	Indicates that the container must run as a non-root user	true

Name	Description	Default Value
containerSecurityContext.allowPrivilegeEscalation	Controls whether a process can gain more privileges than its parent process	false
livenessProbe.enabled	Parameter to enable the liveness probe	true
livenessProbe.initialDelaySeconds	Initial delay (in seconds) for liveness probe	300
livenessProbe.periodSeconds	Period (in seconds) for liveness probe	10
livenessProbe.timeoutSeconds	Timeout (in seconds) for liveness probe	5
livenessProbe.failureThreshold	Failure threshold for liveness probe	15
livenessProbe.successThreshold	Success threshold for liveness probe	1
readinessProbe.enabled	Parameter to enable the readiness probe	true
readinessProbe.initialDelaySeconds	Initial delay (in seconds) for readiness probe	10
readinessProbe.periodSeconds	Period seconds for readiness probe	10
readinessProbe.timeoutSeconds	Timeout seconds for readiness probe	5
readinessProbe.failureThreshold	Failure threshold for readiness probe	3
readinessProbe.successThreshold	Success threshold for readiness probe	1
logCollection.enabled	Parameter to enable cluster wide log collection of Marklogic server logs	false
logCollection.image	Image repository and tag for fluent-bit container	fluent/fluent-bit:3.1.1
logCollection.resources.requests.cpu	The requested cpu resource for the fluent-bit container	100m
logCollection.resources.requests.memory	The requested memory resource for the fluent-bit container	128Mi
logCollection.resources.limits.cpu	The cpu resource limit for the fluent-bit container	100m
logCollection.resources.limits.memory	The memory resource limit for the fluent-bit container	128Mi
logCollection.files.errorLogs	Parameter to enable collection of MarkLogic's error logs when log collection is enabled	true
logCollection.files.accessLogs	Parameter to enable collection of MarkLogic's access logs when log collection is enabled	true
logCollection.files.requestLogs	Parameter to enable collection of MarkLogic's request logs when log collection is enabled	true
logCollection.files.crashLogs	Parameter to enable collection of MarkLogic's crash logs when log collection is enabled	true

Name	Description	Default Value
logCollection.files.auditLogs	Parameter to enable collection of MarkLogic's audit logs when log collection is enabled	true
logCollection.outputs	Used to configure the desired output for fluent-bit	""
haproxy.enabled	Parameter to enable the HAProxy Load Balancer for MarkLogic Server	false
haproxy.existingConfigmap	Name of an existing configmap with configuration for HAProxy	marklogic-haproxy
haproxy.replicaCount	Number of HAProxy deployment	2
haproxy.restartWhenUpgrade.enabled	Indicates whether to automatically roll deployments for every helm upgrade	true
haproxy.stats.enabled	Parameter to enable the stats page for HAProxy	false
haproxy.stats.port	Port for stats page	1024
haproxy.stats.auth.enabled	Parameter to enable the basic auth for stats page	false
haproxy.stats.auth.username	Username for stats page	""
haproxy.stats.auth.password	Password for stats page	""
haproxy.service.type	The service type of the HAproxy	ClusterIP
haproxy.pathbased.enabled	Parameter to enable path based routing on the HAProxy Load Balancer for MarkLogic	false
haproxy.frontendPort	Listening port in the front-end section of the HAProxy when using path-based routing	443
haproxy.defaultAppServers.appservices.path	Path used to expose MarkLogic App-Services App-Server	""
haproxy.defaultAppServers.admin.path	Path used to expose MarkLogic Admin App-Server	""
haproxy.defaultAppServers.manage.path	Path used to expose the MarkLogic Manage App-Server	""
haproxy.additionalAppServers	List of additional HTTP ports configuration for HAproxy	[]
haproxy.tcpports.enabled	Parameter to enable TCP port routing on HAProxy	false
haproxy.tcpports	TCP ports and load balancing type configuration for HAproxy	[]
haproxy.timemout.client	The timeout for inactivity during periods that the client is expected to be speaking	600s

Name	Description	Default Value
haproxy.timeout.connect	This parameter configures the time that HAProxy will wait for a TCP connection to a backend server to be established	600s
haproxy.timeout.server	This parameter measures inactivity when the backend server is expected to be speaking	600s
haproxy.tls.enabled	Parameter that enables TLS for HAProxy	false
haproxy.tls.secretName	Name of the secret that stores the certificate	""
haproxy.tls.certFileName	The name of the certificate file in the secret	""
haproxy.nodeSelector	Node labels for HAProxy pods assignment	{}
haproxy.affinity	Affinity for HAProxy pods assignment	{}
haproxy.resources.requests.cpu	The requested cpu resource for the HAProxy container	250m
haproxy.resources.requests.memory	The requested memory resource for the HAProxy container	128Mi
haproxy.resources.limits.cpu	The cpu resource limit for the HAProxy container	250m
haproxy.resources.limits.memory	The memory resource limit for the HAProxy container	128Mi
ingress.enabled	Enables an ingress resource for the MarkLogic cluster	false
ingress.className	Defines which ingress controller will implement the resource	""
ingress.labels	Additional ingress labels	{}
ingress.annotations	Additional ingress annotations	{}
ingress.hosts	List of ingress hosts	[]
ingress.additionalHost	List of ingress additional hosts	[]

8. Troubleshooting

NOTE
For the commands below, provide the namespace name if the chart is deployed to a different namespace than the current kubectl context. Use `-n <your-namespace>` to apply the command to a specific namespace, or use `--all-namespaces (-A)` to apply the command to all namespaces.

Retrieve the status of deployed resources

To get the status of the Helm deployment, enter this command:

```
helm list
```

To get the status of all the pods in the current namespace, enter this command:

```
kubectl get pods
```

NOTE
The commands above will get all the pods running in the current namespace.

To get the status of all the pods in a MarkLogic deployment, enter this command:

```
kubectl get pods --selector="app.kubernetes.io/name=marklogic" --all-namespaces
```

To list all the pods for a specific release:

```
kubectl get pods --selector="app.kubernetes.io/instance=<RELEASE-NAME>
```

To get detailed information, use the `kubectl describe` command:

```
kubectl describe pods <POD-NAME>
```

NOTE
After entering this command, you can use the Events list at the bottom for debugging.

Statuses for MarkLogic pods

Pending

This status indicates that the pod has been accepted by the Kubernetes system, but the container within the pod has not started yet. If a pod is stuck in this phase, use the `kubectl describe pods <POD-NAME>` command to get more information. Often, a detailed warning is listed in the

Events list at the bottom. For example, if none of the nodes meet the scheduling requirements, a `FailedScheduling` warning event appears in the Events list.

Running

This status indicates that the pod has been scheduled to a node and that all the containers in the pod are running.

Access logs

To access container logs for specific pod, use this command:

```
kubectl logs <pod-name>
```

To access all the logs in MarkLogic server, follow these steps:

1. Use the `kubect exec` command to get access into a specific MarkLogic container:

   ```
   kubectl exec -it <POD-NAME> -- /bin/bash
   ```
2. Go to `/var/opt/MarkLogic/Logs/` to view all the logs.

NOTE
It is recommended that you set up log forwarding in production environments.

Common issues

ImagePullBackOff

- When a pod enters `ImagePullBackOff` status, Kubernetes was unable to download the container image for the pod's container. This could be caused by a network issue or incorrect image tags.
- By default, the image registry is Docker Hub. Test the connection from the node to Docker Hub to make sure that the Kubernetes node has access to the registry.
- If you provide a customized value for the image repository or tag during the installation, use this command to test if the image is valid:

  ```
  kubectl run marklogic --image=marklogicdb/marklogic-db:latest
  ```

CrashLoopBackOff

When a pod enters `CrashLoopBackOff` status, the pod's containers have exited with an error, causing Kubernetes to restart them.

This issue could be caused by several reasons:

- Probe Failure - The MarkLogic container uses a liveness probe to perform a container health check. If the liveness probe fails a certain number of times, the container will restart.
- Insufficient Resources, such as CPU or Memory - Double-check the resource limits and requests specified in the `values.yaml` file.
- Application Failure - Check the container or MarkLogic Server logs to see if there are any errors or messages related to the crashes.

NOTE
To see MarkLogic Server for a crashed container, you need a logs forwarder solution. (FluentBit is enabled in the Helm Chart).

Common debugging practices

1. Get pod statuses by using `kubectl get pods`.
2. Get detailed information by using `kubectl describe pods`.
3. Get container logs and MarkLogic logs.

Recommend guides for debugging in Kubernetes

For more information about how to troubleshoot in Kubernetes, see A visual guide on troubleshooting Kubernetes deployments.

9. Known issues and limitations

1. If the hostname is greater than 64 characters there will be issues with certificates. It is highly recommended to use a hostname shorter than 64 characters or use SANs for hostnames in the certificates. If you still choose to use hostname greater than 64 characters, set `allowLongHostnames` to `true`.
2. The latest released version of fluent/fluent-bit:3.1.1 has known high and critical security vulnerabilities. If you decide to enable the log collection feature, choose and deploy the fluent-bit or an alternate image with no vulnerabilities as per your requirements.
3. The security context `allowPrivilegeEscalation` is set to `false` by default in the `values.yaml` file. This should not be changed when running the MarkLogic container with the default rootless image. If you choose to use an image with root privileges, set `allowPrivilegeEscalation` to `true`.
4. Known Issues and Limitations for the MarkLogic Server Docker image can be viewed at Known Issues and Limitations.
5. Path-based routing and Ingress features are only supported with MarkLogic 11.1 and higher.

10. Technical support

Progress Software provides technical support according to the terms detailed in your Software License Agreement or End User License Agreement.

We invite you to visit our support website at `http://help.marklogic.com` to access information on known and fixed issues, knowledge base articles, and more. For licensed customers with an active maintenance contract, see the *Support Handbook* for instructions on registering support contacts and on working with the MarkLogic Server Technical Support team.

Complete product documentation, the latest product release downloads, and other useful information is available for all developers at `http://developer.marklogic.com`. For technical questions, we encourage you to ask your question on the Progress Community.

11. Copyright

For copyright information, see Product Documentation and Copyright Notice.