MarkLogic Server on Kubernetes

MarkLogic 11

Publication date 2024-02-28

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 the 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 the **kubectl -h** command.
   - 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, secretes 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. 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.

3.5. 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.

3.6. Configuration options for Helm

This section describes Helm configuration options.

3.6.1. 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>

3.6.2. 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.

3.6.3. 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
```

3.6.4. 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.

3.6.5. 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
```

3.6.6. 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.

3.7. 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.

3.8. 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, HAProxy Load Balancer, 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 Load Balancer

HAProxy is provided as a load balancer configured to support cookie-based session affinity and multi-statement transactions. These configurations are needed by some MarkLogic client applications, like mlcp. HAProxy is recommended for production workloads.

4.2.1. Enable the HAProxy Load Balancer

The HAProxy Load Balancer is disabled by default. To enable it, provide the following configuration in the `values.yaml` file for the chart installation:

```
haproxy:
    enabled: true
```

NOTE
For multigroup deployment, HAProxy should only be enabled for enode groups.

4.2.2. Configuration

This section includes information on configurations.

ConfigMap

The HAProxy configuration is dynamically generated in ConfigMap with the name "marklogic-haproxy". A custom configuration can be used by creating a new ConfigMap and setting the *existingConfigmap* parameter in the `values.yaml` file to the name of the new ConfigMap.

Modify port configuration

By default, ports 8000, 8001, and 8002 are configured to handle HTTP traffic. The default ports can be modified in the `values.yaml` file. For each entry, *name*, *type*, and *port* must be specified. *Type* can be one of these values:

- HTTP - The backend is configured as an HTTP proxy that handles HTTPC traffic. It can also be configured to handle cookie-based session affinity and multi-statement transactions from the MarkLogic client.
- TCP - The backend is configured as a TCP proxy that handles TCP traffic.

This is the default configuration:

```
haproxy:
  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
```

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

```
    - name: my-app-1
      type: HTTP
      port: 8010
      targetPort: 8010
```

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

```
    - name: odbc
      type: TCP
      port: 5432
      targetPort: 5432
```

Automatic reload configuration

When a change occurs in the current deployment (such as a change to the backend ports, or the number of running MarkLogic nodes), the HAProxy is restarted by default. During the restart process, the new configuration is loaded. However, the default behavior can be changed with this setting:

```
haproxy:
    restartWhenUpgrade:
        enabled: false
```

NOTE
if you update the number of MarkLogic nodes and *restartWhenUpgrade.enabled* is set to `false`, you must manually delete the HAProxy deployment to get the latest configuration.

Access HA Proxy

In Kubernetes, each pod is assigned its own IP address and namespace. Containers within a pod share the same namespace, IP, and MAC address as the pod. Therefore, nodes and pods can communicate with one another without using Network Address Translation (NAT). However, if a pod is restarted for any reason, it will be assigned a new IP address. Because a pod does not have a fixed IP or DNS, a service is used to connect to an application.

The HAProxy can be accessed from a service with the name of `<RELEASE_NAME>-haproxy`. For example, if the release name is `marklogic`, then the name of the service is `marklogic-haproxy`.

External access

By default, HAProxy is configured to provide access within the Kubernetes cluster. However, HAProxy can provide external access by setting the service type in the `values.yaml` file:

```
haproxy:
  service:
    type: LoadBalancer
```

WARNING
By setting the haproxy service type to `LoadBalancer`, the MarkLogic endpoint is exposed to the public internet. Because of this, `networkPolicy` should be set to limit the sources that can visit MarkLogic.

4.3. HTTP connection through Ingress on an EKS cluster

This section discusses three approaches to exposing a MarkLogic cluster on an EKS cluster via HTTP.

IMPORTANT
For all the approaches, manage the Ingresses that expose the MarkLogic Load Balancer outside of the Helm charts. Create and deploy a dedicated Ingress file using this command:

kubectl apply -f <ingress-file> -n <marklogic-release-namespace>

4.3.1. Application Load Balancer (ALB) Ingress only

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

NOTE

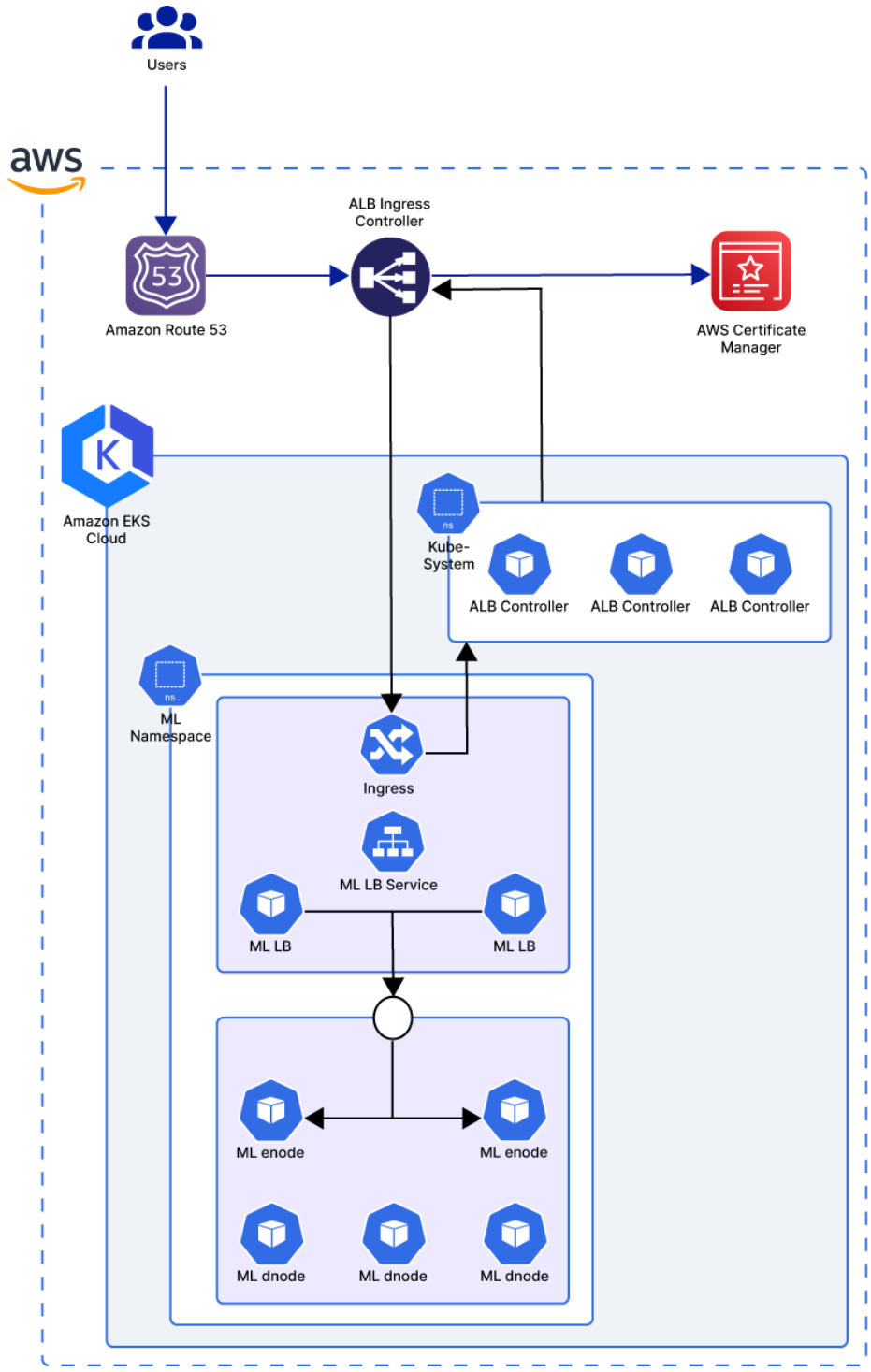
This approach will not address ODBC exposition, as ALB Ingress only supports HTTP/HTTPS connections. See ODBC connection through Ingress in EKS for further information.

For each Ingress created, a new ALB will be created. The `group.name` feature allows several Ingresses to be grouped into the same ALB.

Limitations of ALB Ingress only

- 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.

Architecture



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 Ingresses definition will automatically create an ALB. To expose a MarkLogic cluster, deploy one Ingress per application server:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '8002'
    alb.ingress.kubernetes.io/healthcheck-path: /
    alb.ingress.kubernetes.io/success-codes: '200-401'
    alb.ingress.kubernetes.io/load-balancer-name: ml-lb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS": 8001}]'
    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:XXXXXXXXXX:
    certificate/ XXXXX-xxxx-XXXX-XXXX-XXXXxxxXXX
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-lb-group
  labels:
    app.kubernetes.io/instance: ml-enode
    app.kubernetes.io/name: ml-lb
  name: ml-enode-ml-lb-admin
  namespace: ml-lb
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-enode-ml-lb
            port:
              number: 8001
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '8002'
    alb.ingress.kubernetes.io/healthcheck-path: /
    alb.ingress.kubernetes.io/success-codes: '200-401'
    alb.ingress.kubernetes.io/load-balancer-name: ml-lb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS": 8000}]'
    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:XXXXXXXXXX:
    certificate/XXXXX-xxxx-XXXX-XXXX-XXXXxxxXXX
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-lb-group
  labels:
    app.kubernetes.io/instance: ml-enode
    app.kubernetes.io/name: ml-lb
  name: ml-enode-ml-lb-console
  namespace: ml-lb
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-enode-ml-lb
            port:
              number: 8000
```

```
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '8002'
    alb.ingress.kubernetes.io/healthcheck-path: /
    alb.ingress.kubernetes.io/success-codes: '200-401'
    alb.ingress.kubernetes.io/load-balancer-name: ml-lb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS": 8002}]'
    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/XXXXX-xxxx-XXXX-XXXX-XXXXxxxXXX
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-lb-group
  labels:
    app.kubernetes.io/instance: ml-enode
    app.kubernetes.io/name: ml-lb
  name: ml-enode-ml-lb-manage
  namespace: ml-lb
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-enode-ml-lb
            port:
              number: 8002
        path: /
        pathType: Prefix
```

Annotation details

Code	Description
`alb.ingress.kubernetes.io/healthcheck-port`	Specifies the port on which to perform the health check. See https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/guide/ingress/annotations/#healthcheck-port.
`alb.ingress.kubernetes.io/healthcheck-path`	Specifies the path used for the health check. See https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/guide/ingress/annotations/#healthcheck-path.
`alb.ingress.kubernetes.io/success-codes`	Specifies the HTTP status code that is expected during health checks against the specified health check path. The 401 code is included because authentication is needed on the port. See https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.2/guide/ingress/annotations/#success-codes.
`alb.ingress.kubernetes.io/load-balancer-name: ml-lb`	Specifies the prefix for the name of the ALB. Note that name impacts the entire IngressGroup. See https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/guide/ingress/annotations/#load-balancer-name.
`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 https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/guide/ingress/annotations/#scheme.

Code	Description
`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 Application Load Balancer (ALB) Ingress only for the limitations. See https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/guide/ingress/annotations/#listen-ports.
`alb.ingress.kubernetes.io/target-group-attributes`	Specifies the target group attributes as the load balancing algorithm.
`alb.ingress.kubernetes.io/certificate-arn`	Specifes the certificates to be used (HTTPS termination on the ALB should be enabled). See https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/guide/ingress/annotations/#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 https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/guide/ingress/annotations/#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 https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/guide/ingress/annotations/#group.name.

For additional annotations, see the complete list.

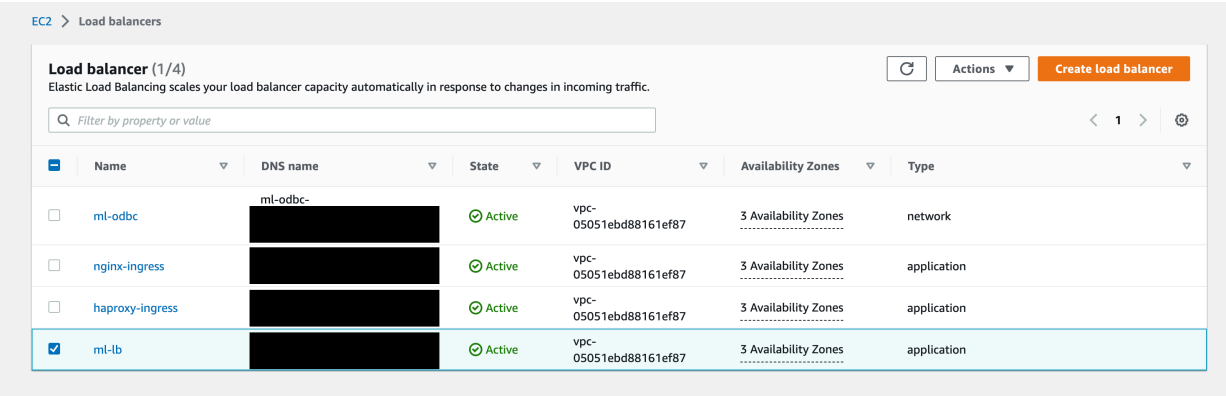
Ingress class name

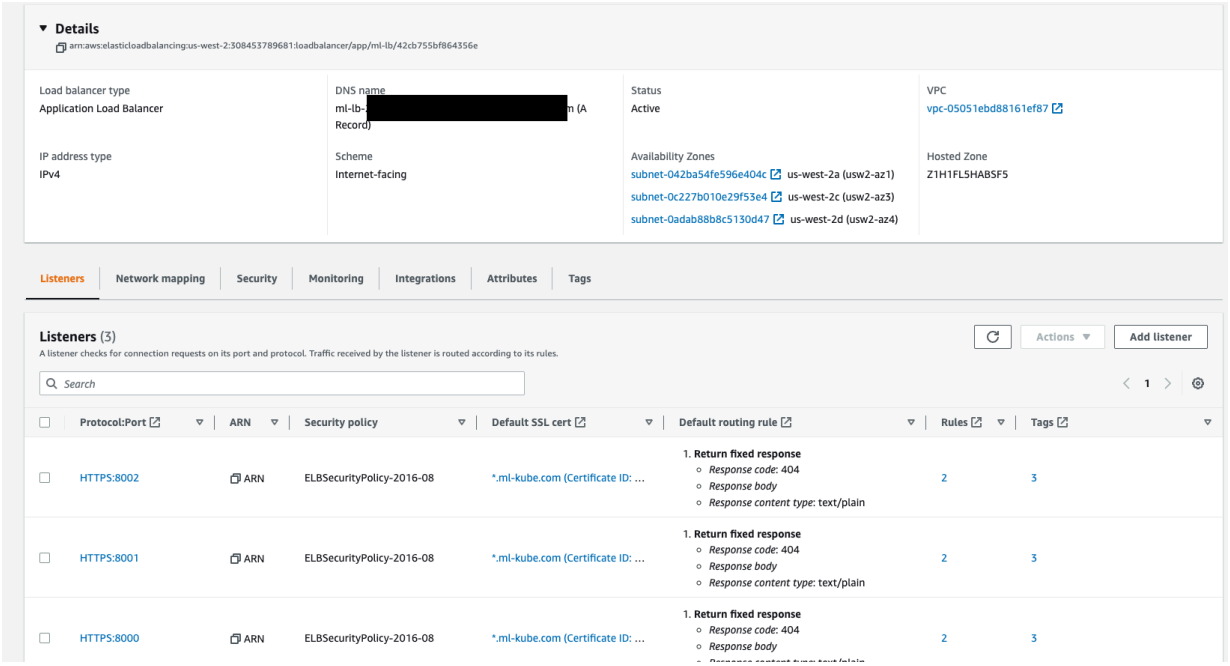
Specifying the ingressClass is recommended, especially with multiple Ingress controllers. Adding *alb* ensures that the Ingress will be managed by the ALB Ingress Controller:

```
spec:
    ingressClassName: alb
```

Checking the ALB

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

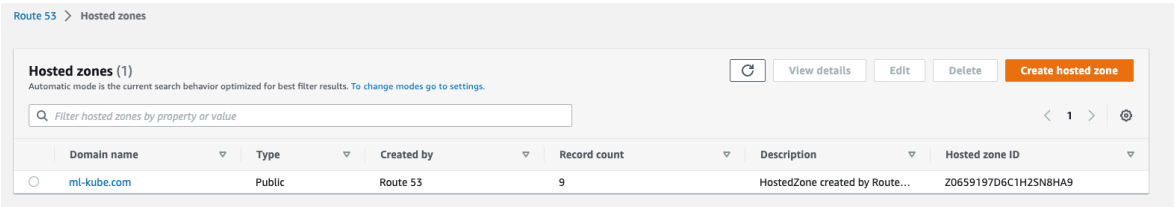




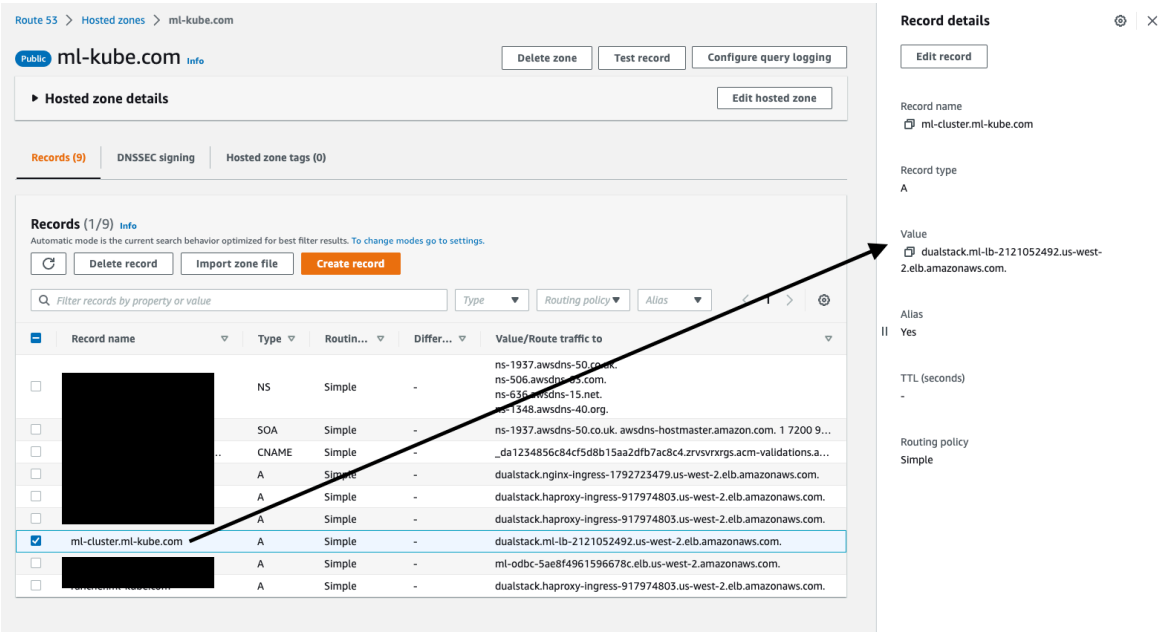
Route53

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:

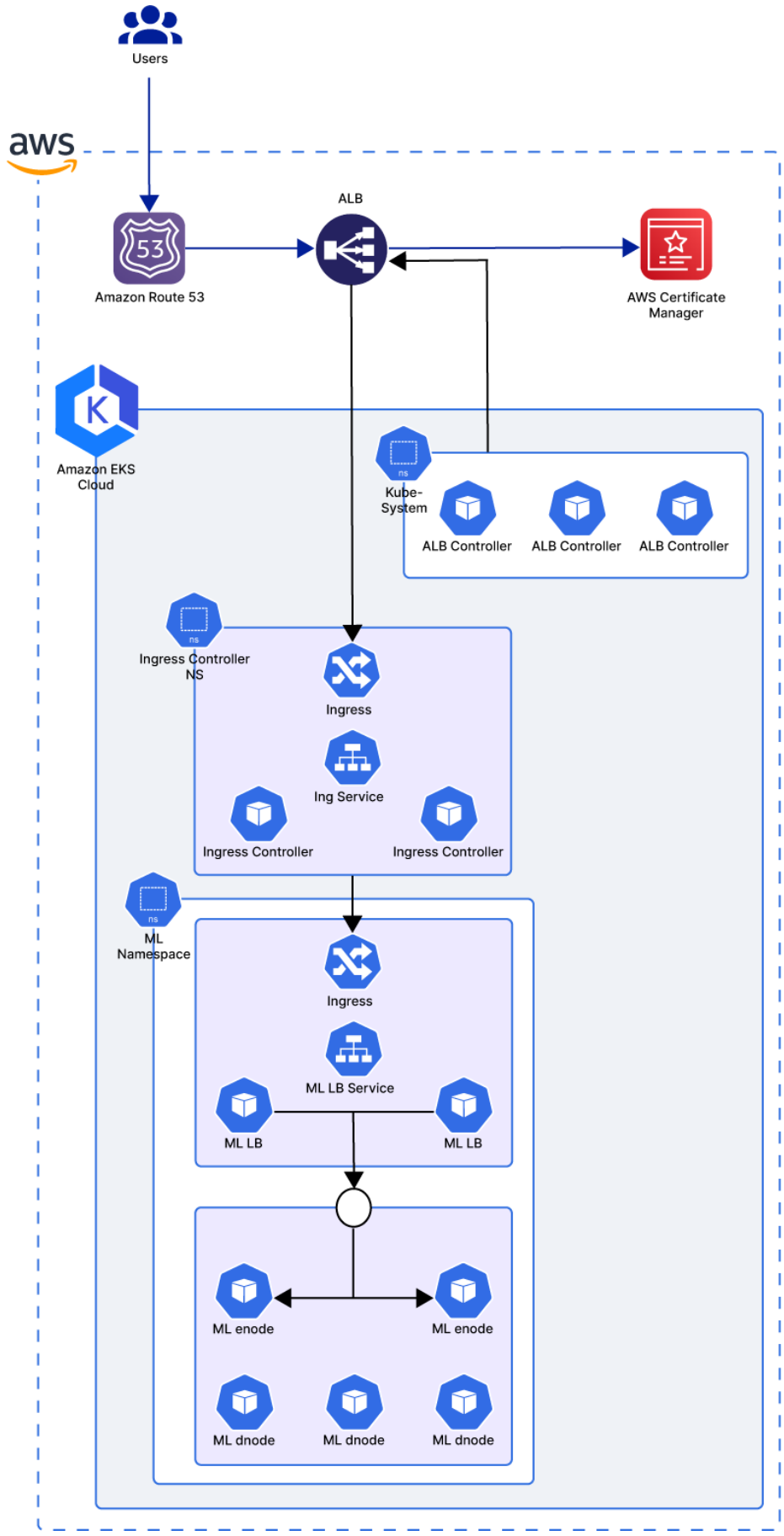


4.3.2. ALB + HAProxy as Ingress controller

In this approach, the ALB points to an HAProxy Ingress Controller. The HAProxy Ingress Controller is exposed using ALB Ingress. This allows management of the HTTP + ODBC connection (See ODBC connection through Ingress in EKS).

The internal MarkLogic Load Balancer is exposed using Ingress and is managed by the HAProxy Ingress Controller. `IngressClass` is used to split the Ingress management.

Architecture



ALB Ingress Controller installation

See Install ALB Ingress for information.

HAProxy Ingress Controller installation

The HAProxy Ingress Controller is installed using the Official Helm Chart with the configuration below. In the example, the Ingress controller is identified as `type: NodePort` (this allows the ODBC exposition). The HA can be achieved in 2 ways: either by using `DaemonSet` or by using three replicas for deployment with node anti-affinity to ensure that the pods will be deployed on separate Kubernetes nodes.

```
## 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'
```

In the next example, `ingressClass` is specified as `haproxy`. This allows the MarkLogic Ingress to be managed using the HAProxy Ingress Controller.

```
## IngressClass:
  ## Ref: https://github.com/haproxytech/kubernetes-ingress/blob/master/
  documentation/ingressclass.md

  # k8s >= 1.18: IngressClass resource used, in multi-Ingress environments, to
    select Ingress resources to implement.
  # ref: https://kubernetes.io/docs/concepts/services-networking/ingress/
  #ingress-class
  # ref: https://kubernetes.io/docs/concepts/services-networking/ingress/
  #default-ingress-class
  # Note: Uses ingressClass as name for the Ingress Class object if enabled
  ingressClassResource:
    name: haproxy
    default: true
    parameters: {}

  # k8s < 1.18: Ingress Class used, in multi-Ingress environments,
  for ingress.class annotation to select Ingress resources to implement.
  # k8s >= 1.18: Ingress Class used to target specific HAProxy Ingress Controller
  in multi-Ingress envionments
  # ref: https://kubernetes.io/docs/concepts/services-networking/
  ingress-controllers/#using-multiple-ingress-controllers
  ingressClass: "haproxy"   # typically "haproxy" or null to receive all
  eventscode needs to be inserted here
```

In the next example, the `config:stats-config-snippet` configuration allows basic authentication for the statistic page exposed. `request-capture` is used to capture the host and user agent from the user request.

```
config:
 #   timeout-connect: "250ms"
 #   servers-increment: "10"
 #   servers-increment-max-disabled: "10"
 #   rate-limit: "ON"
 #   rate-limit-expire: "1m"
 #   rate-limit-interval: "10s"
 #   rate-limit-size: "100k"
   stats-config-snippet: |
     stats auth ml-admin:ml-admin2022
   request-capture: |
     hdr(Host)
     hdr(User-Agent)Code needs to be inserted here
```

Ingress

The HAProxy Ingress Controller is exposed using ALB Ingress. To do this, an Ingress needs to be created:

```
apiVersion: networking.k8s.io/v1
  kind: Ingress
  metadata:
    name: haproxy-ingress
    namespace: ingress
    annotations:
      alb.ingress.kubernetes.io/healthcheck-port: '1042'
      alb.ingress.kubernetes.io/healthcheck-path: /healthz
      alb.ingress.kubernetes.io/group.name: haproxy
      alb.ingress.kubernetes.io/load-balancer-name: haproxy-ingress
      alb.ingress.kubernetes.io/scheme: internet-facing
      alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":443}, {"HTTP":80},
      {"HTTPS":8000}, {"HTTPS":8001}, {"HTTPS":8002} ]'
      alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-west-2:XXXXXXXXXX:
      certificate/XXXXX-xxxx-XXXX-XXXX-XXXXxxxXXX
      alb.ingress.kubernetes.io/target-group-attributes: stickiness.enabled=true,
      stickiness.lb_cookie.duration_seconds=60
      alb.ingress.kubernetes.io/target-type: ip
  spec:
    ingressClassName: alb
    rules:
    - http:
        paths:
        - path: /
          backend:
            service:
              name: haproxy-kubernetes-ingress
              port:
                number: 80
          pathType: Prefix
        - path: /stats
          backend:
            service:
              name: haproxy-kubernetes-ingress
              port:
                number: 1024
          pathType: Prefix
```

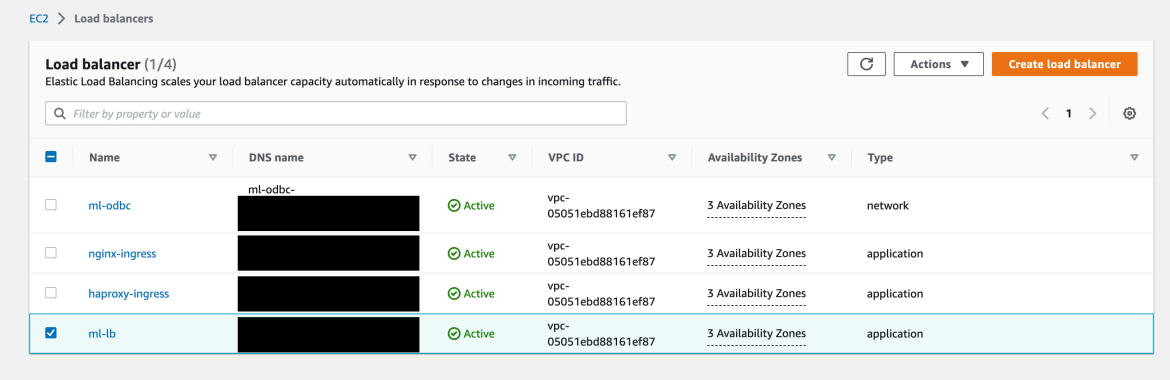
The example above has two rules:

- `path : /` - This is used by all the Ingresses controlled by HAProxy.
- `path: /stats` - This is used to expose the statistics pages of the HAProxy (protected with basic authentication as seen in the previous section).

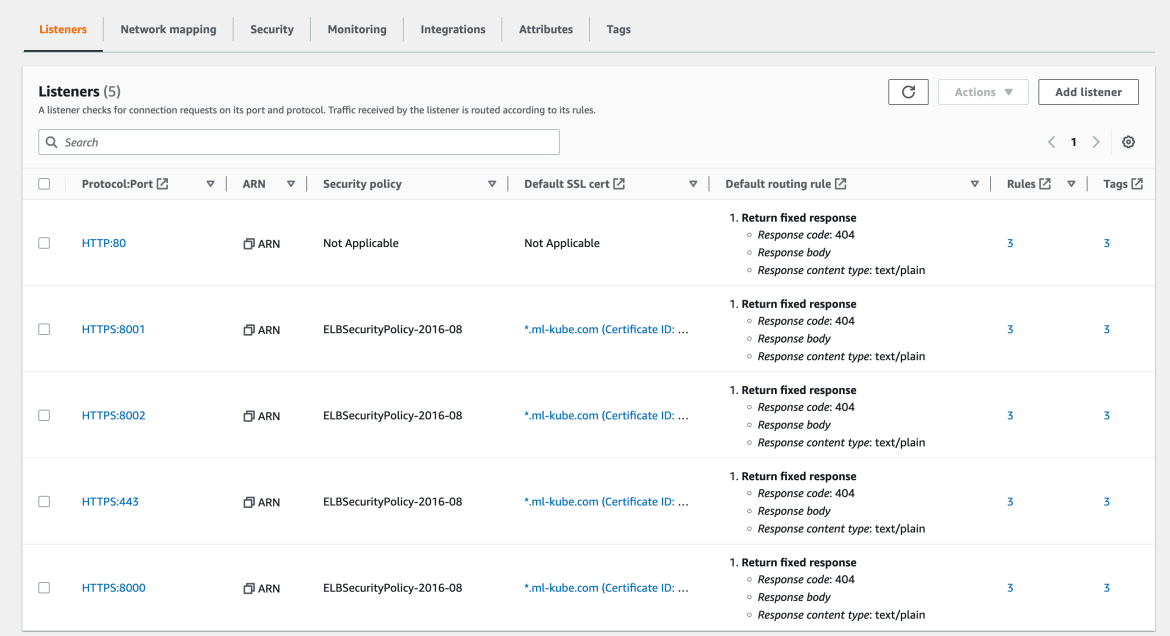
ALB and HAProxy check

ALB check

1. Go to the AWS console to check what the created ALB looks like:



2. Check the Listeners:



HAProxy Ingress Controller check

Pod details

```
$ kubectl get po -n ingress -o wide
NAME                                                READY   STATUS    RESTARTS   AGE   IP              NODE                                              NOMINATED NODE
READINESS GATES
haproxy-kubernetes-ingress-ccfbd9997-2d79d          1/1     Running   0          11d   192.168.31.17   ip-192-168-3-102.us-west-2.compute.internal       <none>
<none>
haproxy-kubernetes-ingress-ccfbd9997-9tbrn          1/1     Running   0          11d   192.168.69.82   ip-192-168-80-151.us-west-2.compute.internal      <none>
<none>
haproxy-kubernetes-ingress-ccfbd9997-brb9k          1/1     Running   0          11d   192.168.48.161  ip-192-168-50-204.us-west-2.compute.internal      <none>
<none>
haproxy-kubernetes-ingress-default-backend-59dc4b55d9-bh6fv   1/1   Running   0          14d   192.168.42.101  ip-192-168-50-204.us-west-2.compute.internal      <none>
```

Ingress details

```
$ kubectl get ing -n ingress -o wide
NAME              CLASS   HOSTS   ADDRESS                                                  PORTS   AGE
haproxy-ingress   alb     *       haproxy-ingress-xxxxxxxxx.us-west-2.elb.amazonaws.com    80      12d
```

MarkLogic Load Balancer Ingress configuration

An Ingress can be created to expose the MarkLogic Load Balancer via the HAProxy Ingress Controller.

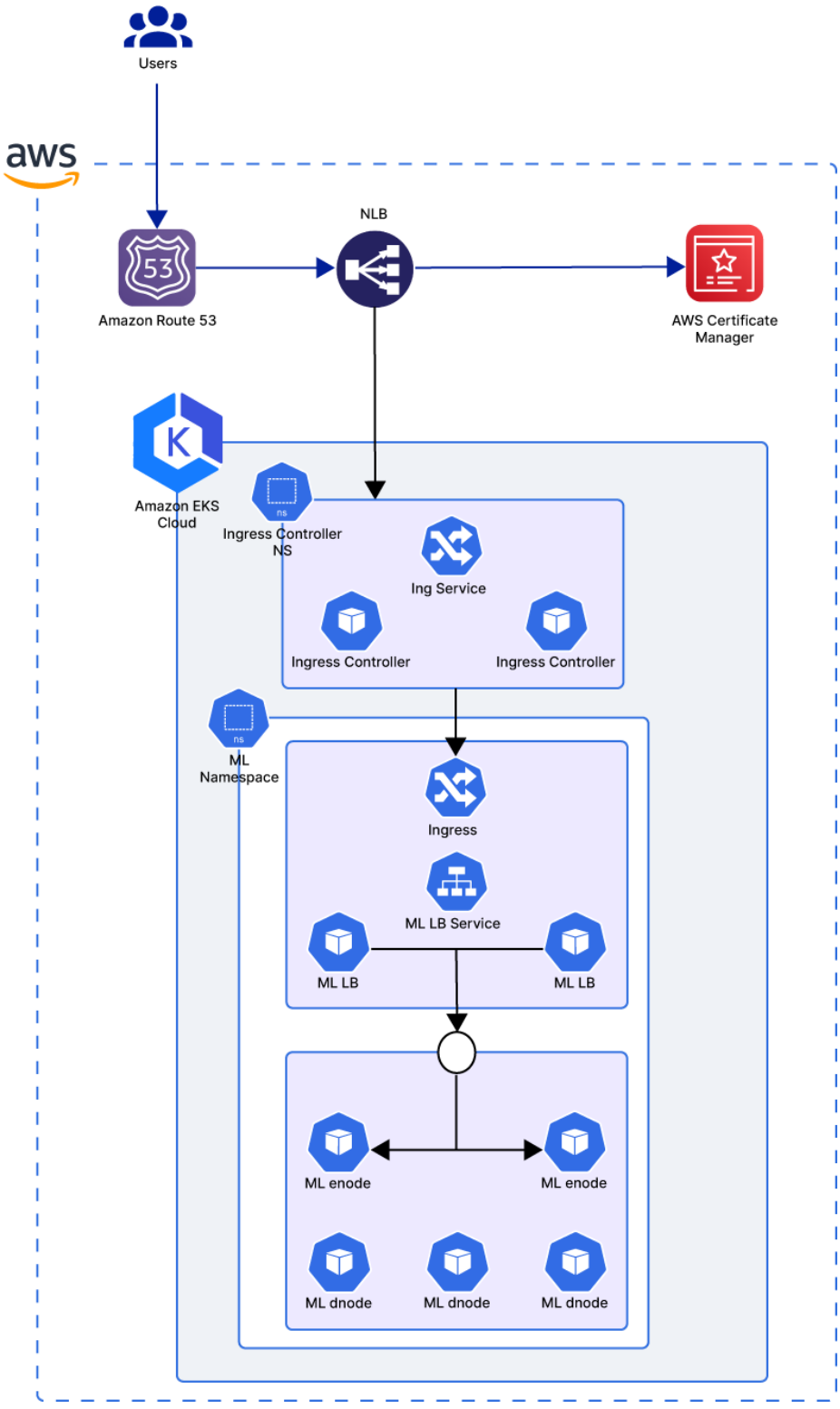
To expose a MarkLogic cluster, deploy one Ingress per application server:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  labels:
    app.kubernetes.io/instance: ml-enode
    app.kubernetes.io/name: ml-lb
  name: ml-enode-ml-lb-admin
  namespace: ml-lb
spec:
  ingressClassName: haproxy
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-enode-ml-lb
            port:
              number: 8001
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  labels:
    app.kubernetes.io/instance: ml-enode
    app.kubernetes.io/name: ml-lb
  name: ml-enode-ml-lb-console
  namespace: ml-lb
spec:
  ingressClassName: haproxy
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-enode-ml-lb
            port:
              number: 8000
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  labels:
    app.kubernetes.io/instance: ml-enode
    app.kubernetes.io/name: ml-lb
  name: ml-enode-ml-lb-manage
  namespace: ml-lb
spec:
  ingressClassName: haproxy
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-enode-ml-lb
            port:
              number: 8002
        path: /
        pathType: Prefix
```

Items specified

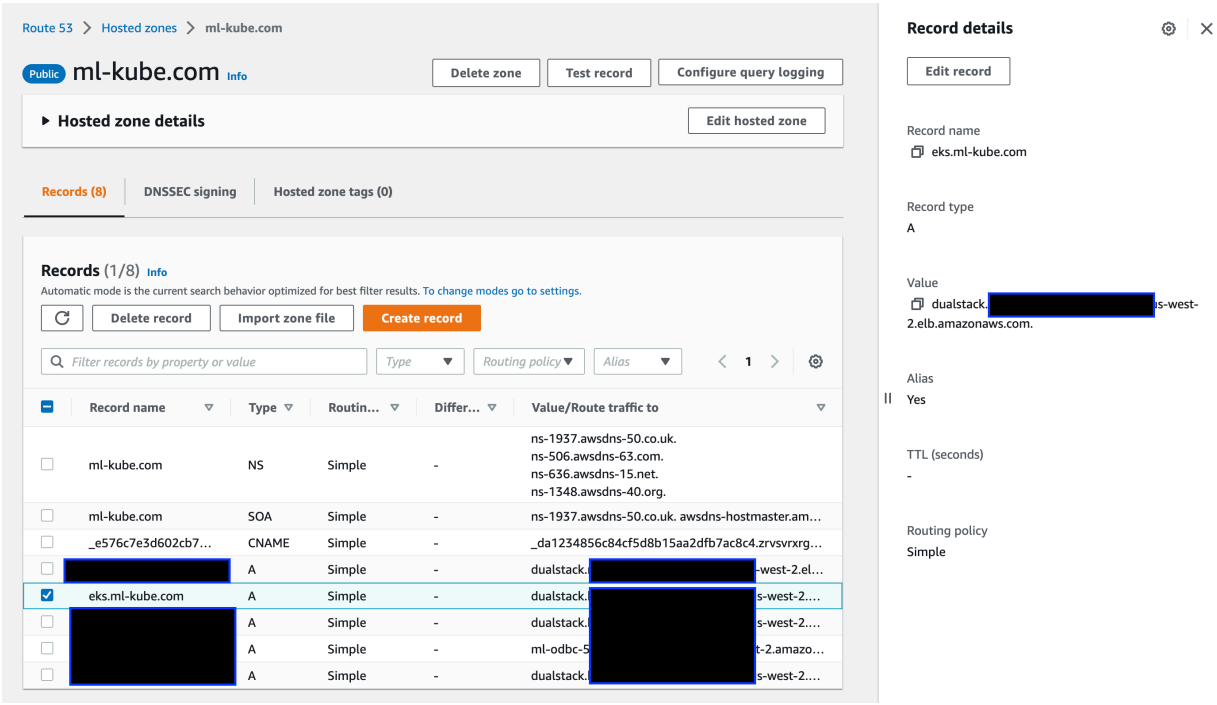
Code	Description
IngressClass	Set to `haproxy` in order to use the HAProxy Ingress Controller.

Macro architecture



Route53

With the ALB, HAProxy Ingress Controller, and the Ingress deployed to expose the MarkLogic cluster, Route53 can be managed to create a dedicated record for the MarkLogic cluster.

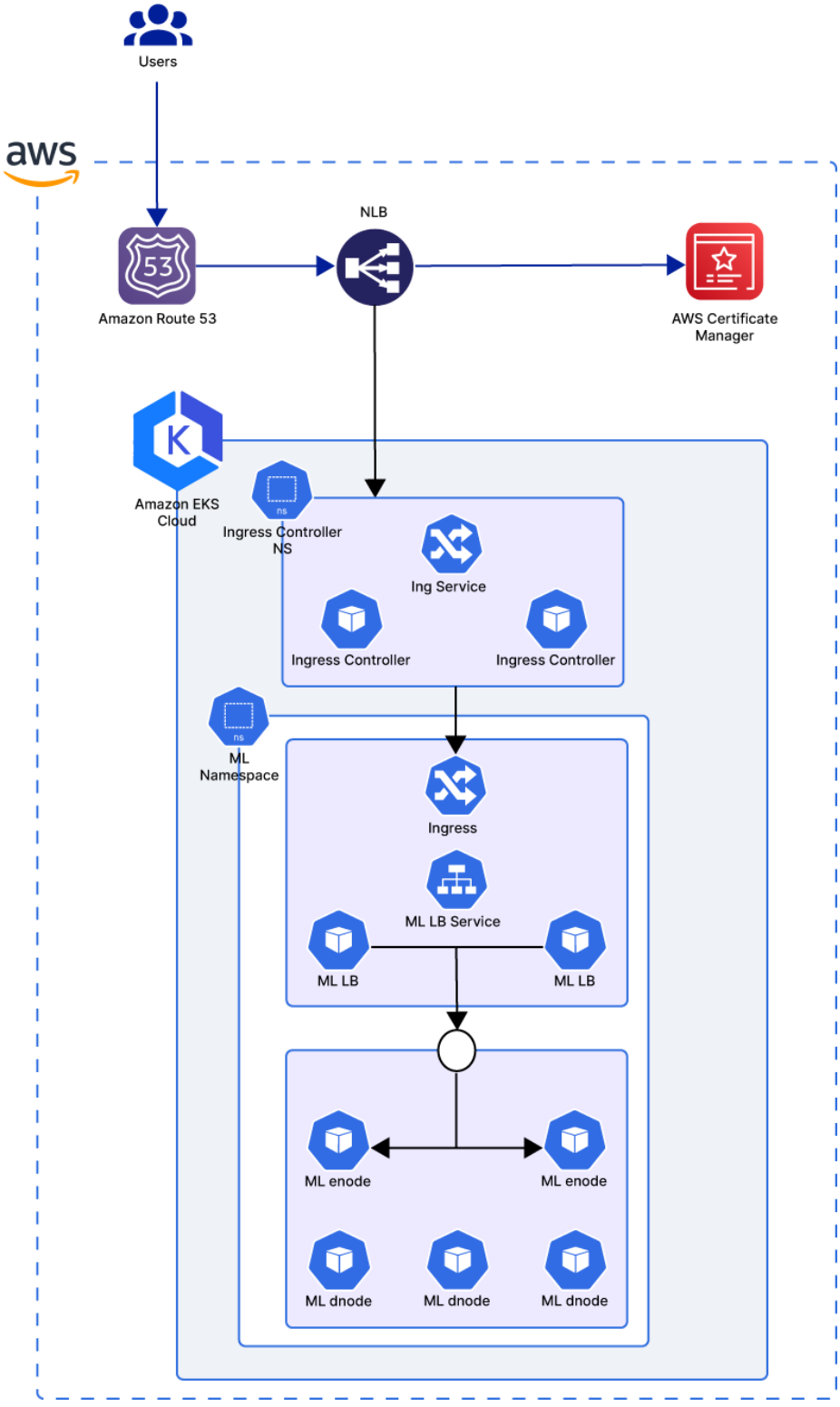


After creating the dedicated record, the Admin UI can be accessed at https://eks.ml-kube.com:8001/.

4.3.3. HAProxy Ingress controller using service load balancer

Using this approach, the Ingress Controller is exposed using a service type of LoadBalancer. This can be done only on cloud providers that support an external load balancer. The service type `LoadBalancer` only provides L4 load balancing, but it can be used for ODBC connections.

Macro architecture



AWS Load Balancer Controller installation

The load balancer controller provisions the AWS Network Load Balancer (NLB) and Application Load Balancer (ALB) resources. The load balancer controller watches for new services or Ingress Kubernetes resources and configures AWS resources.

See the AWS Load Balancer Controller Installation.

HAProxy Ingress controller installation

To expose the HAProxy Ingress Controller, set the service type to `LoadBalancer`:

```
service:
    enabled: true     # set to false when controller.kind is 'DaemonSet' and
    controller.daemonset.useHostPorts is true

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

    ## Service annotations
    ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/
       annotations/
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "external"
      service.beta.kubernetes.io/aws-load-balancer-connection-idle-timeout: "60"
      service.beta.kubernetes.io/aws-load-balancer-name: haproxy-nlb
      service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
      service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http
      service.beta.kubernetes.io/aws-load-balancer-ssl-cert: arn:aws:acm:
      us-west-2:XXXXXXXXXX:certificate/XXXXX-xxxx-XXXX-XXXX-XXXXxxxXXX
      service.beta.kubernetes.io/aws-load-balancer-ssl-ports: "443,8000,8001,8002,
      1024"
      service.beta.kubernetes.io/aws-load-balancer-healthcheck-path: /healthz
      service.beta.kubernetes.io/aws-load-balancer-healthcheck-port: "31042"
      service.beta.kubernetes.io/aws-load-balancer-healthcheck-protocol: http
      service.beta.kubernetes.io/aws-load-balancer-healthcheck-healthy-threshold:
      "2"
      service.beta.kubernetes.io/aws-load-balancer-healthcheck-unhealthy-threshold:
      "3"
      service.beta.kubernetes.io/aws-load-balancer-healthcheck-interval: "15"
      service.beta.kubernetes.io/aws-load-balancer-healthcheck-timeout: "5"
```

Items specified

Code	Description
`type : LoadBalancer`	Indicates that an external load balancer will be created for the Ingress Controller.

Annotations

Additional annotations are included below. See the full documentation for additional information.

Annotation	Description
`service.beta.kubernetes.io/aws-load-balancer-name: haproxy-nlb`	Defines the name of the NLB.
`service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http`	Indicates the SLL offload will be done at the NLB level.
`service.beta.kubernetes.io/aws-load-balancer-ssl-cert: arn:aws:acm:us-west-2:XXXXXXXXXX:certificate/XXXX-xxxx-XXXX-XXXX-XXXXxxxXXX`	Defines the ARN of the certificate used for SSL configuration.
`service.beta.kubernetes.io/aws-load-balancer-ssl-ports: "443,8000,8001,8002,1024"`	Sets up the ports exposed at the NLB level: • Standard HTTP and HTTPS port + MarkLogic standard ports. • Port 1024 exposes the HAProxy statistic page.
`service.beta.kubernetes.io/aws-load-balancer-healthcheck-port: "31042"`	NodePort 31042 exposes the health check port for HAProxy. See the example.

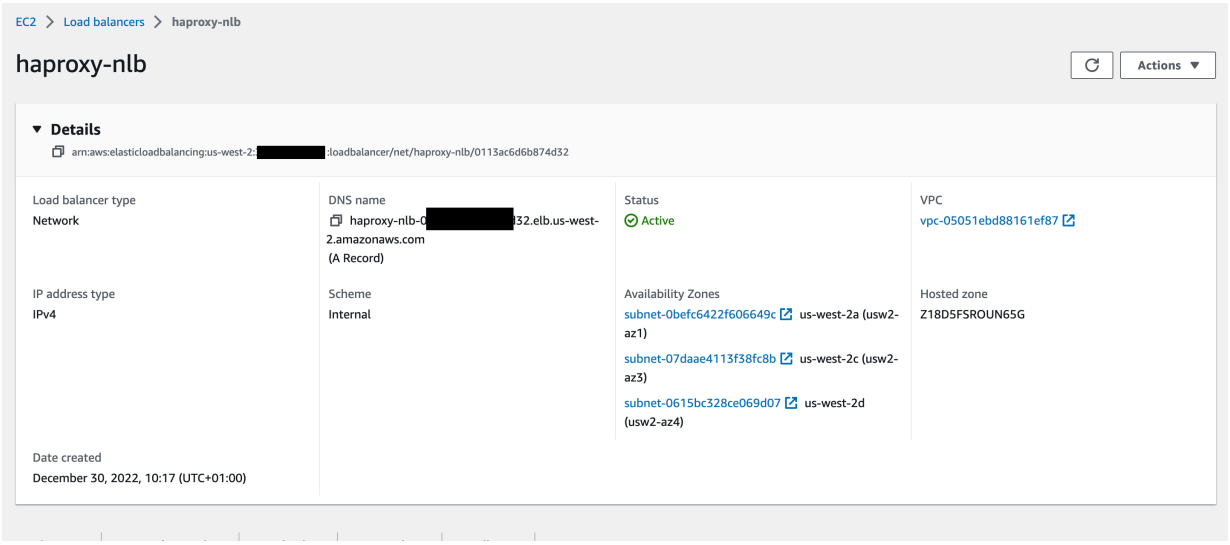
Example

```
## Controller Container listener port configuration
  ## ref: https://kubernetes.io/docs/concepts/services-networking/connect-
     applications-service/
  containerPort:
    http: 80
    https: 443
    stat: 1024
    ml-odbc: 5432
    healthz: 1042
    ml-query: 8000
    ml-admin: 8001
    ml-manage: 8002
...
   ## 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
      - name: healthz
        port: 1042
        targetPort: 1042
        nodePort: 31042
      - name: ml-query
        port: 8000
        targetPort: 8000
        nodePort: 31800
      - name: ml-admin
        port: 8001
        targetPort: 8001
        nodePort: 31801
      - name: ml-manage
        port: 8002
        targetPort: 8002
        nodePort: 31802
```

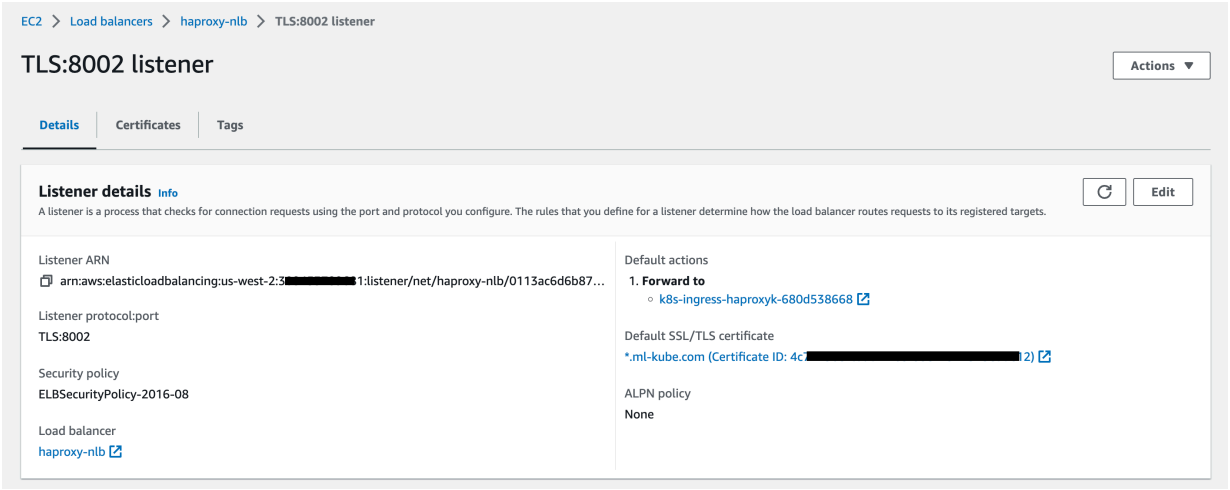
The health check port 1042 at HAProxy is being exposed using NodePort 31042. The health check is the default port used for the readiness/liveness and startup probe.

Check the status of the network load balancer

Check the status of the the network load balancer on the AWS portal:



This example shows port 8082:



Separate the Application Plane and the Administration Plane

For security purposes, it is often useful to separate access to services into different planes:

- Application plane - used by standard users to access the different application servers through the internal load balancer.
- Administration plane - used by MarkLogic administrators and allows access to each individual MarkLogic node.

This is implemented by using different Ingresses and using a custom configuration file for the HAProxy (internal load balancer).

Configure the HA Proxy

The MarkLogic Helm Chart includes a dedicated HAProxy used as an internal load balancer. The Helm chart allows a custom ConfigMap to configure the HAProxy.

Configure the Application Plane

An extract of the ConfigMap responsible for the application plane is shown in this section:

```
## Application Plane

frontend marklogic-8000
  mode http
  bind :8000
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor
  default_backend marklogic-8000

backend marklogic-8000
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-marklogic-8000-0 ml-cluster-marklogic-0.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8000 resolvers dns init-addr none cookie ml-cluster-
marklogic-8000-0
  server ml-cluster-marklogic-8000-1 ml-cluster-marklogic-1.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8000 resolvers dns init-addr none cookie ml-cluster-
marklogic-8000-1
  server ml-cluster-marklogic-8000-2 ml-cluster-marklogic-2.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8000 resolvers dns init-addr none cookie ml-cluster-
marklogic-8000-2

frontend marklogic-8001
  mode http
  bind :8001
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor
  default_backend marklogic-8001

backend marklogic-8001
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-marklogic-8001-0 ml-cluster-marklogic-0.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8001 resolvers dns init-addr none cookie ml-cluster-
marklogic-8001-0
  server ml-cluster-marklogic-8001-1 ml-cluster-marklogic-1.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8001 resolvers dns init-addr none cookie ml-cluster-
marklogic-8001-1
  server ml-cluster-marklogic-8001-2 ml-cluster-marklogic-2.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8001 resolvers dns init-addr none cookie ml-cluster-
marklogic-8001-2

frontend marklogic-8002
  mode http
  bind :8002
```

```
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor
  default_backend marklogic-8002

backend marklogic-8002
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-marklogic-8002-0 ml-cluster-marklogic-0.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8002 resolvers dns init-addr none cookie ml-cluster-
marklogic-8002-0
  server ml-cluster-marklogic-8002-1 ml-cluster-marklogic-1.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8002 resolvers dns init-addr none cookie ml-cluster-
marklogic-8002-1
  server ml-cluster-marklogic-8002-2 ml-cluster-marklogic-2.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8002 resolvers dns init-addr none cookie ml-cluster-
marklogic-8002-2

frontend marklogic-8010
  mode http
  bind :8010
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor

  default_backend marklogic-8010

backend marklogic-8010
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-marklogic-8010-0 ml-cluster-marklogic-0.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8010 resolvers dns init-addr none cookie ml-cluster-
marklogic-8010-0
  server ml-cluster-marklogic-8010-1 ml-cluster-marklogic-1.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8010 resolvers dns init-addr none cookie ml-cluster-
marklogic-8010-1
  server ml-cluster-marklogic-8010-2 ml-cluster-marklogic-2.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8010 resolvers dns init-addr none cookie ml-cluster-
marklogic-8010-2

frontend marklogic-8011
  mode http
  bind :8011
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor
```

```
  default_backend marklogic-8011

backend marklogic-8011
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-marklogic-8011-0 ml-cluster-marklogic-0.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8011 resolvers dns init-addr none cookie ml-cluster-
marklogic-8011-0
  server ml-cluster-marklogic-8011-1 ml-cluster-marklogic-1.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8011 resolvers dns init-addr none cookie ml-cluster-
marklogic-8011-1
  server ml-cluster-marklogic-8011-2 ml-cluster-marklogic-2.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8011 resolvers dns init-addr none cookie ml-cluster-
marklogic-8011-2

frontend marklogic-8013
  mode http
  bind :8013
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor

  default_backend marklogic-8013

backend marklogic-8013
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-marklogic-8013-0 ml-cluster-marklogic-0.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8013 resolvers dns init-addr none cookie ml-cluster-
marklogic-8013-0
  server ml-cluster-marklogic-8013-1 ml-cluster-marklogic-1.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8013 resolvers dns init-addr none cookie ml-cluster-
marklogic-8013-1
  server ml-cluster-marklogic-8013-2 ml-cluster-marklogic-2.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8013 resolvers dns init-addr none cookie ml-cluster-
marklogic-8013-2
```

- A dedicated frontend/backend is included for each port exposed by MarkLogic.
- Each server in the backend section is declared using the FQDN of each pod:
  `<pod-name>.<headless-service-name>.<release-namespace>.svc.cluster.local:<marklogic-port>`
  For example:
  `ml-cluster-marklogic-0.ml-cluster-marklogic-headless.ml.svc.cluster.local:8000`
- Each MarkLogic node is declared on each backed section.

Configure the Administration Plane

In this example, HAProxy is configured to choose the MarkLogic node to connect to. Port routing is used, however, hostname routing can also be used.

```
### Administration Plane

## Access ML-0 node
## Access query console

frontend marklogic-0-8000
  mode http
  bind :81
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor

  default_backend marklogic-0-8000

backend marklogic-0-8000
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-8000-0 ml-cluster-marklogic-0.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8000 resolvers dns init-addr none cookie ml-cluster-
marklogic-0-8000

## Access admin UI

frontend marklogic-0-8001
  mode http
  bind :82
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor

  default_backend marklogic-0-8001

backend marklogic-0-8001
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-8001-0 ml-cluster-marklogic-0.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8001 resolvers dns init-addr none cookie ml-cluster-
marklogic-0-8001

## Access manage

frontend marklogic-0-8002
  mode http
  bind :83
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
```

```
  option forwardfor

  default_backend marklogic-0-8002

backend marklogic-0-8002
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-8002-0 ml-cluster-marklogic-0.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8002 resolvers dns init-addr none cookie ml-cluster-
marklogic-0-8002

# Access ML-1 node

## Access query console

frontend marklogic-1-8000
  mode http
  bind :84
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor

  default_backend marklogic-1-8000

backend marklogic-1-8000
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-8000-1 ml-cluster-marklogic-1.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8000 resolvers dns init-addr none cookie ml-cluster-
marklogic-1-8000

## Access admin UI

frontend marklogic-1-8001
  mode http
  bind :85
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor

  default_backend marklogic-1-8001

backend marklogic-1-8001
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
```

```
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-8001-1 ml-cluster-marklogic-1.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8001 resolvers dns init-addr none cookie ml-cluster-
marklogic-1-8001

## Access manage

frontend marklogic-1-8002
  mode http
  bind :86
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor

  default_backend marklogic-1-8002

backend marklogic-1-8002
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-8002-1 ml-cluster-marklogic-1.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8002 resolvers dns init-addr none cookie ml-cluster-
marklogic-1-8002

# Access ML-2 node

## Access query console

frontend marklogic-2-8000
  mode http
  bind :87
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor

  default_backend marklogic-2-8000

backend marklogic-2-8000
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-8000-2 ml-cluster-marklogic-2.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8000 resolvers dns init-addr none cookie ml-cluster-
marklogic-2-8000
```

```
## Access admin UI

frontend marklogic-2-8001
  mode http
  bind :88
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor

  default_backend marklogic-2-8001

backend marklogic-2-8001
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-8001-2 ml-cluster-marklogic-2.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8001 resolvers dns init-addr none cookie ml-cluster-
marklogic-2-8001

## Access manage

frontend marklogic-2-8002
  mode http
  bind :89
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor

  default_backend marklogic-2-8002

backend marklogic-2-8002
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-8002-2 ml-cluster-marklogic-2.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8002 resolvers dns init-addr none cookie ml-cluster-
marklogic-2-8002
```

NOTE

There is a frontend/backend section for each node + MarkLogic port to expose. For example, for the node `ml-cluster-marklogic-2` on port 8002, the frontend section `frontend marklogic-2-8002` is using the backend section `backend marklogic-2-8002`.

The code below shows the frontend bound on port 89 at the HAProxy level. Only the pod `ml-cluster-marklogic-2` is used in the backend section. There is one Ingress per HAProxy as the Administration Plane will be exposed:

```
## Access manage

frontend marklogic-2-8002
  mode http
  bind :89
  log-format "%ci:%cp [%tr] %ft %b/%s %TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r"
  option httplog
  option forwardfor

  default_backend marklogic-2-8002

backend marklogic-2-8002
  mode http
  balance leastconn
  option forwardfor
  cookie haproxy insert indirect nocache maxidle 30m maxlife 4h
  stick-table type string len 32 size 10k expire 4h
  stick store-response res.cook(HostId)
  stick store-response res.cook(SessionId)
  stick match req.cook(HostId)
  stick match req.cook(SessionId)
  default-server check
  server ml-cluster-8002-2 ml-cluster-marklogic-2.ml-cluster-marklogic-
headless.ml.svc.cluster.local:8002 resolvers dns init-addr none cookie ml-cluster-
marklogic-2-8002
We bind the frontend on the port 89 at HAProxy level and only the pod ml-cluster-
marklogic-2 is used in the backend section.
```

Configure the Ingress

The Ingresses should be managed outside of the Helm chart in a dedicated yaml file. In this example, there is one Ingress per HAProxy to expose (application + administration plane):

```
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '8001'
    alb.ingress.kubernetes.io/healthcheck-path: /
    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": 8001}]'
    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/XxXXXXX-XXxxxXXX-XXxx-XXXxxx
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml
  name: ml-cluster-haproxy-admin
  namespace: ml
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-cluster-haproxy
            port:
              number: 8001
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '8000'
    alb.ingress.kubernetes.io/healthcheck-path: /
    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": 8000}]'
    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/XxXXXXX-XXxxxXXX-XXxx-XXXxxx
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml
  name: ml-cluster-haproxy-console
  namespace: ml
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-cluster-haproxy
            port:
```

```
              number: 8000
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '8002'
    alb.ingress.kubernetes.io/healthcheck-path: /
    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": 8002}]'
    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:XXXXXXXXXXXX:certificate/XxXXXXX-XXxxxXXX-XXxx-XXXxxx
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml
  name: ml-cluster-haproxy-manage
  namespace: ml
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-cluster-haproxy
            port:
              number: 8002
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '81'
    alb.ingress.kubernetes.io/healthcheck-path: /
    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": 81}]'
    alb.ingress.kubernetes.io/ssl-redirect: '81'
    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:XXXXXXXXXXXX:certificate/XxXXXXX-XXxxxXXX-XXxx-XXXxxx
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml
  name: ml-cluster-haproxy-console-0
  namespace: ml
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
```

```
      - backend:
          service:
            name: ml-cluster-haproxy
            port:
              number: 81
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '82'
    alb.ingress.kubernetes.io/healthcheck-path: /
    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": 82}]'
    alb.ingress.kubernetes.io/ssl-redirect: '82'
    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:XXXXXXXXXXXX:certificate/XxXXXXX-XXxxxXXX-XXxx-XXXxxx
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml
  name: ml-cluster-haproxy-admin-0
  namespace: ml
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-cluster-haproxy
            port:
              number: 82
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '83'
    alb.ingress.kubernetes.io/healthcheck-path: /
    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": 83}]'
    alb.ingress.kubernetes.io/ssl-redirect: '83'
    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:XXXXXXXXXXXX:certificate/XxXXXXX-XXxxxXXX-XXxx-XXXxxx
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml
  name: ml-cluster-haproxy-manage-0
  namespace: ml
```

```
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-cluster-haproxy
            port:
              number: 83
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '84'
    alb.ingress.kubernetes.io/healthcheck-path: /
    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": 84}]'
    alb.ingress.kubernetes.io/ssl-redirect: '84'
    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:XXXXXXXXXXXX:certificate/XxXXXXX-XXxxxXXX-XXxx-XXXxxx
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml
  name: ml-cluster-haproxy-console-1
  namespace: ml
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-cluster-haproxy
            port:
              number: 84
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '85'
    alb.ingress.kubernetes.io/healthcheck-path: /
    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": 85}]'
    alb.ingress.kubernetes.io/ssl-redirect: '85'
    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:XXXXXXXXXXXX:certificate/XxXXXXX-XXxxxXXX-XXxx-XXXxxx
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
```

```
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml
  name: ml-cluster-haproxy-admin-1
  namespace: ml
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-cluster-haproxy
            port:
              number: 85
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '86'
    alb.ingress.kubernetes.io/healthcheck-path: /
    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": 86}]'
    alb.ingress.kubernetes.io/ssl-redirect: '86'
    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:XXXXXXXXXXXX:certificate/XxXXXXX-XXxxxXXX-XXxx-XXXxxx
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml
  name: ml-cluster-haproxy-manage-1
  namespace: ml
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-cluster-haproxy
            port:
              number: 86
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '87'
    alb.ingress.kubernetes.io/healthcheck-path: /
    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": 87}]'
    alb.ingress.kubernetes.io/ssl-redirect: '87'
    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/XxXXXXX-XXxxxXXX-XXxx-XXXxxx
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml
  name: ml-cluster-haproxy-console-2
  namespace: ml
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-cluster-haproxy
            port:
              number: 87
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '88'
    alb.ingress.kubernetes.io/healthcheck-path: /
    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": 88}]'
    alb.ingress.kubernetes.io/ssl-redirect: '88'
    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/XxXXXXX-XXxxxXXX-XXxx-XXXxxx
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml
  name: ml-cluster-haproxy-admin-2
  namespace: ml
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-cluster-haproxy
            port:
              number: 88
        path: /
        pathType: Prefix
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    alb.ingress.kubernetes.io/healthcheck-port: '89'
    alb.ingress.kubernetes.io/healthcheck-path: /
    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": 89}]'
    alb.ingress.kubernetes.io/ssl-redirect: '89'
    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/XxXXXXX-XXxxxXXX-XXxx-XXXxxx
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/group.name: ml-group
  labels:
    app.kubernetes.io/instance: marklogic
    app.kubernetes.io/name: ml
  name: ml-cluster-haproxy-manage-2
  namespace: ml
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: ml-cluster-haproxy
            port:
              number: 89
        path: /
        pathType: Prefix
```

NOTE

The ALB Ingress Controller is managing the Ingresses. All Ingresses can point to the same hostname becasue port routing is used.

It is recommended to set path as / to avoid problems when using mlcp.

4.4. ODBC connection through Ingress in EKS

Ingress does not support TCP or UDP services. However, a MarkLogic ODBC application server can be exposed using HAProxy as an Ingress controller.

HAProxy Ingress Controller provides a feature to expose the TCP port through Ingress.

4.4.1. Components

This section contains information on the EKS cluster, MarkLogic cluster, MarkLogic Load Balancer, the Ingress controller, and the AWS Network LoadBalancer.

EKS cluster

The EKS cluster is composed of three worker nodes, each hosted on a different availability zone (AZ).

MarkLogic cluster

The MarkLogic cluster is deployed on a dedicated namespace and is composed of two groups, a Dnode group and an Enode group.

Dnode group

- Managed by its own StatefulSet

- Each pod hosted on a different worker node
- Dedicated to Dnodes
- Not exposed through the MarkLogic Load Balancer

Enode group

- Managed by its own StatefulSet
- Exposed through the MarkLogic Load Balancer
- Dedicated to Enodes

MarkLogic Load Balancer (HAProxy Load Balancer)

- Deployed into a dedicated namespace
- Deployment composed of two replicas
- Each pod deployed on a different worker node

Ingress Controller (HAProxy Ingress Controller)

- Deployed on a dedicated namespace
- Configured with two replicas
- Each pod deployed on a different worker node
- Exposed via NodePort

AWS Network LoadBalancer

- Listens on all Ingress NodePorts via the TCP service
- Exposed using Route53

4.4.2. Configuring components

This section contains information on configuring componets.

ODBC configuration

Configure the ODBC application server at the Enode level using the SQL Quick Start.

MarkLogic Load Balancer configuration

The MarkLogic Load Balancer in the Helm chart can manage TCP connections:

```
    {{- if eq $portType "TCP"  -}}
      listen odbc
        bind :{{ $portNumber }}
        mode tcp
        balance leastconn
        {{- range $i := until $replicas }}
        server {{ printf "ml-%s-%s-%v" $releaseName $portNumber $i }}
        {{ $releaseName }}-{{ $i }}.{{ $releaseName }}-headless.
        {{ $namespace }}.svc.{{ $clusterDomain }}:{{ $portNumber }}
        check resolvers dns init-addr none
        {{- end }}
    {{- else if eq $portType "HTTP" }}
```

These connections are managed via 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
```

Code Explanation

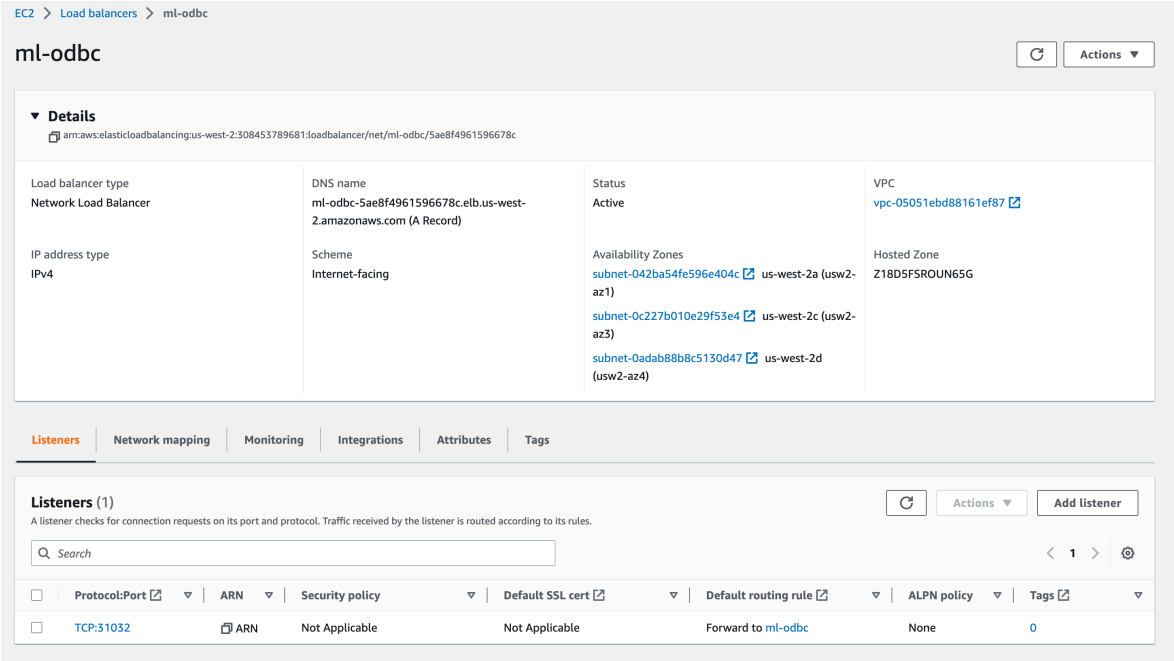
Code	Description
`mode tcp`	TCP is set because a layer 4 load balancer is required for ODBC connections.
`balance leastconn`	This setting selects the servers with the fewest active sessions.

Security group

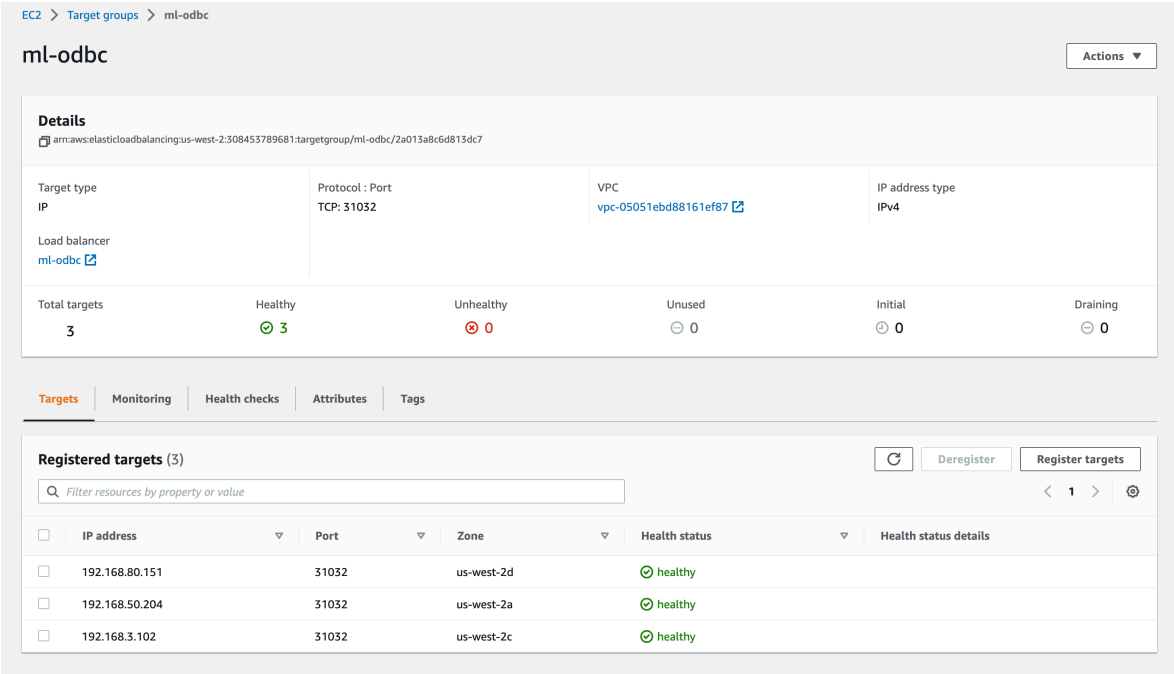
Create a dedicated security group to allow the NLB to communicate with the worker node on the NodePort.

AWS Network Load Balancer

1. Configure the AWS Network Load Balancer like this:



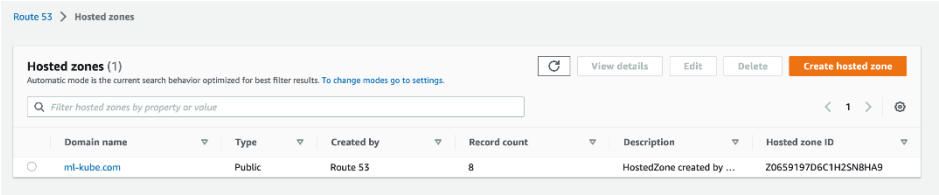
2. The load balancer listens on port 31032 (but it could be another TCP port).



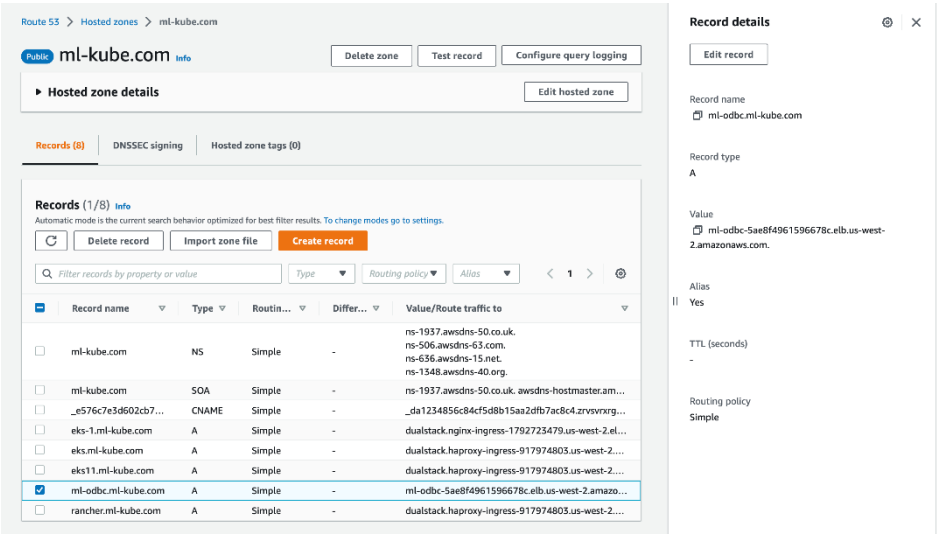
The listeners are configured to forward on to a specific target group. The target group has three worker nodes registered to bind on NodePort 31032. This is specific to the Ingress Controller service port for ODBC. For a basic health check, ping the specified TCP port.

Configure Route 53

1. Route53 is configured with `ml-kube.com` as the hosted zone:



2. For this hosted zone, a dedicated record for ODBC is specified:



3. The endpoint `ml-odbc.ml-kube.com:31032` is used to connect to the MarkLogic ODBC.

Perform a connection test

1. Follow the MarkLogic Knowledge article for setting up ODBC on a Linux environment:
   odbc.ini

```
[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.
3. Test a query:
   ODBC test

```
[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;
+------------------------------------------------------------------------------
FIRST NAME                : LAST NAME                : EXPENSE PER EMPLOYEE
------------------------------------------------------------------------------
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 maintaning 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, it is recommended that you review the changes in a release and test the upgrade in a non-production environment.
- Minor and patch releases will include bug fixes and other smaller changes.

5.1.1. Recommendations before upgrading

Before you start the upgrade process, consider these recommendations:

- Read the MarkLogic documentation for details on upgrading MarkLogic.
- Make sure to 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.

Upgrade procedures

This section describes three upgrade procedures:

- Upgrade MarkLogic Kubernetes Helm Chart version
- Upgrade MarkLogic version in your release
- Upgrade MarkLogic and the Helm Chart versions at the same time

Upgrade MarkLogic Kubernetes 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 Kuberenetes 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 versions at the same time

1. To upgrade the MarkLogic and Helm chart versions at the same time, follow steps 1-5 in Upgrade MarkLogic Kubernetes Helm Chart versionand 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.

5.2. 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 "additional volumes" 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 develop 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 HAProxy Load Balancer 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 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

This table describes the list of available parameters for the Helm chart:

Name	Description	Default Value	As of Version
replicaCount	Number of MarkLogic nodes	1	1.0.0
updateStrategy.type	Update strategy for MarkLogic pods	OnDelete	1.1.0
terminationGracePeriod	The number of seconds until the MarkLogic Pod terminates gracefully	120	1.0.0
clusterDomain	Domain for the Kubernetes cluster	cluster.local	1.0.0
group.name	Group name for joining MarkLogic cluster	Default	1.0.0
group.enableXdqpSsl	Parameter to enable SSL encryption for XDQP	true	1.0.0
bootstrapHostName	Host name of MarkLogic bootstrap host (to join a cluster)	""	1.0.0
image.repository	Repository for MarkLogic image	marklogicdb/ marklogic-db	1.0.0
image.tag	Image tag for MarkLogic image	11.1.0-centos-1.1.2	1.1.0
image.pullPolicy	Image pull policy for MarkLogic image	IfNotPresent	1.0.0
initContainers.configureGroup.image	Image for configureGroup InitContainer	curlimages/curl:8.6.0	1.1.0
initContainers.configureGroup.pullPolicy	Pull policy for configureGroup InitContainer	IfNotPresent	1.1.0
initContainers.copyCerts.image	Image for copyCerts InitContainer	redhat/ubi9:9.3	1.1.0
initContainers.copyCerts.pullPolicy	Pull policy for copyCerts InitContainer	IfNotPresent	1.1.0
imagePullSecrets	Registry secret names (as an array)	[]	1.0.0
hugepages.enabled	Parameter to enable Hugepage on MarkLogic	false	1.1.0
hugepages.mountPath	Mountpath for Hugepages	/dev/hugepages	1.1.0
resources	Resource limits for the MarkLogic container	{}	1.0.0
nameOverride	String to override the app name	""	1.0.0
fullnameOverride	String to completely replace the generated name	""	1.0.0
auth.secretName	Kubernetes secret name for MarkLogic admin credentials	""	1.0.0

Name	Description	Default Value	As of Version
auth.adminUsername	Username for the default MarkLogic administrator	""	1.0.0
auth.adminPassword	Password for the default MarkLogic administrator	""	1.0.0
auth.walletPassword	Password for the wallet	"	1.0.0
tls.enableOnDefaultAppServers	Parameter to enalbe TLS on Default App Servers (8000, 8001, 8002)	false	1.1.0
tls.certSecretNames	Names of the secrets that contain the named certificate	[]	1.1.0
tls.caSecretName	Name of the secret that contain the CA certificate	""	1.1.0
enableConverters	Parameter to install converters for the client (if they are not already installed).	false	1.0.0
license.key	Parameter to indicate the installed MarkLogic license key	""	1.0.0
license.licensee	Parameter to set the MarkLogic licensee information	""	1.0.0
affinity	Affinity for MarkLogic pods assignment	{}	1.0.0
topologySpreadConstraints	POD Topology Spread Constraints to spread Pods across cluster	[]	1.1.0
nodeSelector	Node labels for MarkLogic pods assignment	{}	1.0.0
persistence.enabled	Parameter to enable MarkLogic data persistence using Persistence Volume Claim (PVC). If set to false, EmptyDir will be used.	true	1.0.0
persistence.storageClass	The storage class for the MarkLogic data volume. Leave this parameter empty to use the default storage class.	""	1.0.0
persistence.size	Size of storage request for the MarkLogic data volume	10Gi	1.0.0
persistence.annotations	Annotations for Persistence Volume Claim (PVC)	{}	1.0.0
persistence.accessModes	Access mode for persistence volume	["ReadWriteOnce"]	1.0.0

Name	Description	Default Value	As of Version
additionalVolumeClaimTemplates	List of additional volumeClaimTemplates to each MarkLogic container	[]	1.0.0
additionalVolumes	List of additional volumes to add to the MarkLogic containers	[]	1.0.0
additionalVolumeMounts	List of mount points for the additional volumes to add to the MarkLogic containers	[]	1.0.0
additionalContainerPorts	List of ports exposed at the container level (in addition to the default ports). Typically, this parameter is not updated. Use `service.additionalPorts` to expose the app server ports.	[]	1.0.0
service.annotations	Annotations for MarkLogic service	{}	1.0.0
service.type	Default service type	ClusterIP	1.0.0
service.additionalPorts	List of ports in addition to the defaults exposed at the service level	[]	1.1.0
serviceAccount.create	Parameter to enable creating a service account for a MarkLogic Pod	true	1.0.0
serviceAccount.annotations	Annotations for the MarkLogic service account	{}	1.0.0
serviceAccount.name	Name of the serviceAccount	""	1.0.0
priorityClassName	Name of the PriortyClass defined to set pod priority	""	1.0.0
networkPolicy.enabled	Parameter to enable network policy	false	1.0.0
networkPolicy.customRules	Placeholder to specify selectors	{}	1.0.0
networkPolicy.ports	Parameter to specify the ports where traffic is allowed	[{port:8000, endPort: 8020, protocol: TCP}]	1.0.0
containerSecurityContext.enabled	Parameter to enable security context for MarkLogic containers	true	1.0.0
containerSecurityContext.runAsUser	User ID for running the entrypoint of the container process	1000	1.0.0
containerSecurityContext.runAsNonRoot	When set to true, this parameter indicates that the container must run as a non-root user	true	1.0.0
containerSecurityContext.allowPrivilegeEscalation	Controls whether a process can gain more privileges than its parent process	true	1.0.0

Name	Description	Default Value	As of Version
livenessProbe.enabled	Parameter to enable the liveness probe	true	1.0.0
livenessProbe.initialDelaySeconds	Parameter to indicate the Initial delay for the liveness probe (in seconds)	300	1.1.0
livenessProbe.periodSeconds	Period seconds for the liveness probe	20	1.1.0
livenessProbe.timeoutSeconds	Timeout for the liveness probe (in seconds)	5	1.0.0
livenessProbe.failureThreshold	Failure threshold for the liveness probe	15	1.10
livenessProbe.successThreshold	Success threshold for the liveness probe	1	1.0.0
readinessProbe.enabled	Parameter to enable the readiness probe	false	1.1.0
readinessProbe.initialDelaySeconds	Initial delay for the readiness probe (in seconds)	10	1.0.0
readinessProbe.periodSeconds	Period seconds for the readiness probe	60	1.0.0
readinessProbe.timeoutSeconds	Timeout for the readiness probe (in seconds)	5	1.0.0
readinessProbe.failureThreshold	Failure threshold for the readiness probe	3	1.0.0
readinessProbe.successThreshold	Success threshold for the readiness probe	1	1.0.0
startupProbe.enabled	Parameter to enable the startup probe	false	1.1.0
startupProbe.initialDelaySeconds	Initial delay for the startup probe (in seconds)	10	1.0.0
startupProbe.periodSeconds	Period seconds for the startup probe	20	1.0.0
startupProbe.timeoutSeconds	Timeout for the startup probe (in seconds)	1	1.0.0
startupProbe.failureThreshold	Failure threshold for the startup probe	30	1.0.0
startupProbe.successThreshold	Success threshold for the startup probe	1	1.0.0
logCollection.enabled	Parameter to enable cluster-wide log collection of Marklogic server logs	false	1.0.0
logCollection.image	Image repository and tag for the fluent-bit container	fluent/fluent-bit:2.2.2	1.1.0
logCollection.resources.requests.cpu	The requested cpu resource for the fluent-bit container	100m	1.0.0
logCollection.resources.requests.memory	The requested memory resource for the fluent-bit container	128Mi	1.0.0
logCollection.resources.limits.cpu	The cpu resource limit for the fluent-bit container	100m	1.0.0

Name	Description	Default Value	As of Version
logCollection.resources.limits.memory	The memory resource limit for the fluent-bit container	128Mi	1.0.0
logCollection.files.errorLogs	Parameter to enable the collection of MarkLogic error logs when clog collection is enabled	true	1.0.0
logCollection.files.accessLogs	Parameter to enable the collection of MarkLogic access logs (when log collection is enabled)	true	1.0.0
logCollection.files.requestLogs	Parameter to enable collection of MarkLogic request logs (when log collection is enabled)	true	1.0.0
logCollection.files.crashLog	Parameter to enable collection of MarkLogic crash logs (when log collection is enabled)	true	1.0.0
logCollection.files.auditLogs	Parameter to enable collection of MarkLogic audit logs (when log collection is enabled)	true	1.0.0
logCollection.outputs	Parameter to configure the desired output for fluent-bit	""	1.0.0
haproxy.enabled	Parameter to enable the HAProxy Load Balancer for MarkLogic Server	false	1.0.0
haproxy.existingConfigmap	Name of an existing configmap with configuration for HAProxy	marklogic-haproxy	1.0.0
haproxy.replicaCount	Number of HAProxy Deployments	2	1.0.0
haproxy.restartWhenUpgrade.enabled	Parameter to automatically roll deployments for every helm upgrade	true	1.0.0
haproxy.stats.enabled	Parameter to enable the stats page for HAProxy	false	1.0.0
haproxy.stats.port	Port for the stats page	1024	1.0.0
haproxy.stats.auth.enabled	Parameter to enable the basic auth for the stats page	false	1.0.0
haproxy.stats.auth.username	Username for the stats page	""	1.0.0
haproxy.stats.auth.password	Password for the stats page	""	1.0.0
haproxy.service.type	The service type of the HAproxy	ClusterIP	1.0.0
haproxy.ports	Ports and load balancing type configuration for HAproxy	[]	1.0.0

Name	Description	Default Value	As of Version
haproxy.tls.enabled	Parameter to enable TLS for HAProxy	false	1.0.0
haproxy.tls.secretName	Name of the secret that stores the certificate	""	1.0.0
haproxy.tls.certFileName	The name of the certificate file in the secret	""	1.0.0
haproxy.nodeSelector	Node labels for HAProxy pods assignment	{}	1.0.0
haproxy.affinity	Affinity for HAProxy pods assignment	{}	1.0.0
haproxy.resources.requests.cpu	The requested CPU resource for the HAProxy container	250m	1.0.0
haproxy.resources.requests.memory	The requested memory resource for the HAProxy container	128Mi	1.0.0
haproxy.resources.limits.cpu	The CPU resource limit for the HAProxy container	250m	1.0.0
haproxy.resources.limits.memory	The memory resource limit for the HAProxy container	128Mi	1.0.0

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** 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 recommended to use a hostname shorter than 64 characters or use SANs for hostnames in the certificates.
2. The MarkLogic Docker image must be run in privileged mode. If the image is not run in this mode, many calls that use sudo in the startup script will fail because the required permissions are lacking. The image will also be unable to create a user with the required permissions.
3. The latest released version of CentOS 7 has known security vulnerabilities with respect to glib2 CVE-2016-3191, CVE-2015-8385, CVE-2015-8387, CVE-2015-8390, CVE-2015-8394, CVE-2016-3191, glibc CVE-2019-1010022, pcre CVE-2015-8380, CVE-2015-8387, CVE-2015-8390, CVE-2015-8393, CVE-2015-8394, and SQLite CVE-2019-5827. These libraries are included in the CentOS base image but, to-date, fixes are not available. Even though these libraries may be present in the base image that is used by MarkLogic Server, they are not used by MarkLogic Server itself. Becasue of this, there is no impact or mitigation required.
4. The latest released version of fluent/fluent-bit:2.2.2 has known security vulnerabilities with respect to libcom-err2 CVE-2022-1304, libgcrypt20 CVE-2021-33560, libgnutls30 CVE-2024-0567, libldap-2.4-2 CVE-2023-2953, libzstd1 CVE-2022-4899, and zlib1g CVE-2023-45853. These libraries are included in the Debian base image, but, to-date, fixes are not available. For libpq5 CVE-2024-0985, a future upgrade of the fluent-bit image will include the fix. Updates and mitigation strategies will be provided as soon as more information is available.
5. The latest released version of redhat/ubi9:9.3 has known security vulnerabilities with respect to setup tools GHSA-r9hx-vwmv-q579. A future upgrade of the Redhat ubi image should include the fix.
6. The security context `allowPrivilegeEscalation` is set to `TRUE`by default in the `values.yaml` file and cannot be changed to run the current MarkLogic container. Work is in progress to allow the MarkLogic container to run in "rootless" mode.
7. The Readiness and Startup Probe are not compatible with HA deployment. These probes may fail if there is a Security database failover. As of the 1.0.2 Helm Chart release, the startup and readiness probes are disabled by default.

10. Technical support

MarkLogic 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 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 Stack Overflow.

11. Copyright

For copyright information, see Product Documentation and Copyright Notice.