Progress®
Kemp® LoadMaster®

Feature Description LoadMaster Ingress Controller for Kubernetes

Generated on: 16 September 2024

Copyright

Visit the following page online to see Progress Software Corporation's current Product Documentation Copyright Notice/Trademark Legend: Product Documentation Copyright Notice & Trademarks | Progress

Table of Contents

Chapter 1: Introduction 5

Chapter 2: Prerequisites 6

AWS EKS Setup Steps 7

Chapter 3: Install the Ingress Controller 9

Chapter 4: Enable the LoadMaster ECS CM to Kubernetes Integration . . 11

Chapter 5: Service Mode and Ingress Mode 13

Service Mode 14

How Service Mode Works 16

When to Use Service Mode 16

Configure Service Mode 17

Ingress Mode 19

How Ingress Mode Works 22

When to Use Ingress Mode 22

Configure Ingress Mode 23

Chapter 6: Service Object Annotations 25

Chapter 7: Ensuring Connectivity from the LoadMaster ECS CM to Pods 31

Overlay Networks 32

How to Check if an Overlay Network is in Use 32

LoadMaster ECS CM to Pod Connectivity 34

Adding Routes 34

Calico 34

Chapter 8: Creation and Deletion of Objects 37

Chapter 9: LoadMaster ECS CM Ingress Logs 39

1

Introduction

The ECS Connection Manager (ECS CM)LoadMaster Ingress Controller for Kubernetes enables aan LoadMaster Virtual Service to be used to publish access to Kubernetes applications. This enables application server pools in published Virtual Services to dynamically update when Pods running on a Kubernetes platform are added or removed from a proxied service object. This provides a convenient way for managing the connection between the Load Balancer ECS Connection Manager and Container network for connections into Kubernetes (commonly referred to as North-South Traffic).

Because the LoadMaster operates outside of Kubernetes, the Ingress Controller provides a convenient way for managing endpoints for either Containerized or Monolithic Applications or a combination of both through the same device.

You first need to install the Ingress Controller on the LoadMaster. Once installed, you can enable this functionality in Service or Ingress mode.

This functionality provides the following capabilities:

- Automated mapping of Kubernetes Service or Ingress object configurations to the LoadMaster Virtual Services and SubVSs
- Support for mapping Kubernetes annotations to Virtual Service attributes
- Automated addition and deletion of Real Servers as a result of changes in Kubernetes (for example, scale up or scale down operations)

2

Prerequisites

Some prerequisites that must be in place before configuring the Ingress Controller functionality for Kubernetes are listed below:

- At least one licensed and running LoadMaster instance
- A Kube config file with the necessary permissions to read configuration using the Kubernetes Application Programming Interface (API)
- Network access between the LoadMaster and the Kubernetes API server
- Details of the namespace for which objects are defined
- The LoadMaster must be able to route to the Pod CIDR networks within the relevant Kubernetes Clusters and the Cluster Nodes. This may require additional routes to be created. For further details on how to do this, refer to the Ensuring Connectivity from the LoadMaster to Pods section.

The Ingress Controller is currently confirmed to be supported on the following platforms:

- Cloud:
  - Kubernetes running in Azure (Azure Kubernetes Service (AKS))
  - Kubernetes running in Amazon Web Services (AWS) Elastic Kubernetes Service (EKS). For setup steps that you must perform if using AWS EKS, refer to the AWS EKS Setup Steps section.
  - Kubernetes running in Rancher Kubernetes Management Platform

- On-prem Linux: Kubernetes running on Linux Operating System (OS)

Related Links

For details, see the following topics:

- AWS EKS Setup Steps

AWS EKS Setup Steps

If you intend to use the Ingress Controller for Kubernetes in the AWS EKS platform, there are some setup steps that you must perform:

1. Install the AWS Command Line Interface (CLI) 2.0:

   curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

   unzip awscliv2.zip

   sudo ./aws/install

2. Install kubectl:

   curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.18.9/2020-11-02/bin/linux/amd64/kubectl

3. Install eksctl:

   Note: The AWS user must have permissions to create an EKS cluster. There must also be available resources to create one Virtual Private Cloud (VPC).

   ---

   curl --silent --location

   "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp

   sudo mv /tmp/eksctl /usr/local/bin

   eksctl version

4. Run the command to create the cluster:

   eksctl create cluster --name=eksctl --region=us-east-1 --ssh-public-key=aws_eks_ted.pub --nodes=4 --node-type=m5.large

   ---

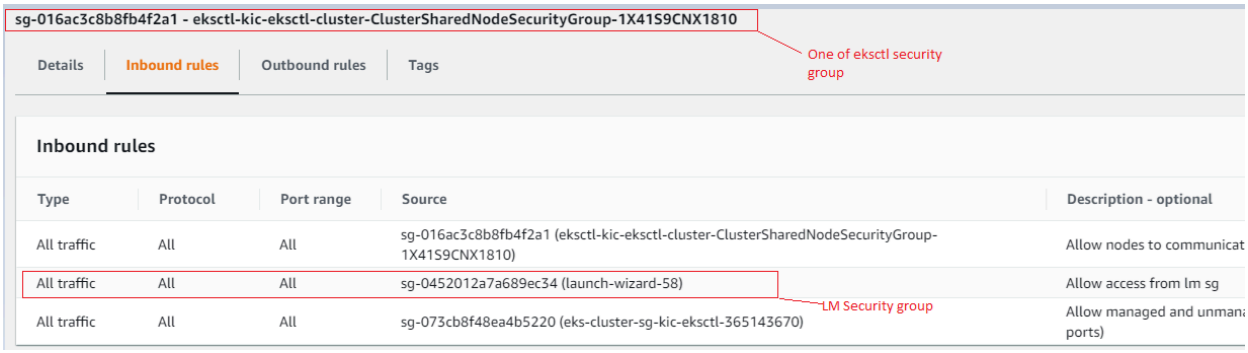
   Note: The above command takes approximately 5 to 10 minutes to execute. Upon successful execution, it creates one VPC, four subnets, and three security groups.

   ---

5. After creating the cluster, create the LoadMaster on AWS with one public interface (subnet) of eksctl and a security group with **All** traffic allowed.

   LoadMaster security group rules: For example, launch-wizard-58

Type	Protocol	Port range
All traffic	All	All



6. Then, add the LoadMaster security group (created above) to all the security groups created by eksctl to ensure communication and to not be restricted by ports, protocols, and so on. The screenshot above shows an example of the eksctl security group.
7. License the LoadMaster, install the addons and upload the Kubeconfig file.
8. Create the configuration on eksctl and health checking should then work.

For further details, refer to the following AWS User Guide: Getting started with Amazon EKS - AWS Management Console and AWS CLI.

3

Install the Ingress Controller

The Ingress Controller is not installed on the LoadMaster by default. You can easily install it by following these steps:

1. In the LoadMaster User Interface (UI), go to **Virtual Services > Kubernetes Settings**.
2. Click **Install**.
3. Wait for the installation to complete and click **OK** on the confirmation message.
4. Reboot the LoadMaster to activate all required add-ons - **System Configuration > System Administration > System Reboot > Reboot**.

Note: It is important to reboot the LoadMaster after upgrading the Ingress Controller add-on. Failure to reboot after upgrading the add-on, and then attempting to apply additional updates, could cause update failures or other issues.

After rebooting, you can use the **Kubernetes Settings** configuration page to enable the LoadMaster Kubernetes integration.

Note: For LoadMasters ECS Connection Managers deployed in AWS or Azure in a High Availability Pair, ensure that the add-ons are installed on both devices.

Note: LoadMaster users (apart from the default admin **bal** user) must be assigned the **All Permissions** option in their user permissions to be permitted access to modify Kubernetes settings in the LoadMaster.

Manual add-on installation

If the LoadMaster is deployed to a location where connectivity to Progress Kemp infrastructure is not available, you can install the Ingress Controller by following the below steps:

1. Download the required add-ons. Go to https://support.kemptechnologies.com/hc/en-us/sections/200409933-Other-Downloads and click **Kemp Ingress Controller Add-Ons**. Here you can download the required add-on files.

2. On LoadMaster, navigate to **System Configuration > System Administration > Update Software**.

3. Under the **Install new Addon Package** menu, add each file and verification file and click **Install Addon Package**.

4. Reboot the LoadMaster to activate all required add-ons, navigate to **System Configuration > System Administration > System Reboot > Reboot**.

4

Enable the LoadMaster ECS CM to Kubernetes Integration

To enable LoadMaster to Kubernetes communication, the LoadMaster must have Kubernetes API access. To enable this, follow the steps below in the LoadMaster UI:

1. In the main menu, go to **Virtual Services > Kubernetes Settings**.
2. Click **Choose File**.
3. Browse to and select your **Kube Config** file.

   Note: If the backup of one LoadMaster is restored to another LoadMaster, it is recommended to upload the **Kube Config** file separately to the restored LoadMaster.

   Note: If using Minikube, it is recommended to embed the certificates used for authentication inside the kubeconfig file. This can be done with the following command in Minikube:**minikube config set embed-certs true**

   Note: The default location for the Kube Config file is ~/.kube/config, for example, if you are using the Azure Cloud Shell you can access this using **/home/<YourName>/.kube/config** using the **Download File** option at the top of the Cloud Shell window.

   Note: This allows the LoadMaster to communicate with Kubernetes.

4. Click **Install**.
5. Once the **Kube Config** file is successfully installed, some information is populated in the **Contexts** section.

Note: The **Name**, **Cluster**, and **User** are shown.

6. Select the relevant **K8S Operations Mode**.

Note: For further details on each mode, including instructions on how to configure each mode in Kubernetes, refer to the Service Mode and Ingress Mode section.

7. Select the **Namespace to Watch**.

Note: All namespaces are watched if this field is unset. You can only filter on either a single name space or all name spaces. There is no capability to select multiple name spaces at this time.

8. Optional: Set the **Ingress Watch Timeout** in seconds.

Note: This is the Ingress Controller watch timeout. Valid values range from 30 - 900.

When configured correctly, details on the Kubernetes **Nodes** and relevant objects (Ingress Objects with an Ingress class specification of "kempLB" and **Service Objects** labeled "**kempLB:Enabled**") are shown at the bottom of the screen.

5

Service Mode and Ingress Mode

The Ingress Controller supports two modes of operation:

- Service Mode: In this mode, the LoadMaster configuration is driven through the configuration of Service Objects in Kubernetes.
- Ingress Mode: In this mode, the LoadMaster configuration is driven through the configuration of the Ingress Object in Kubernetes.

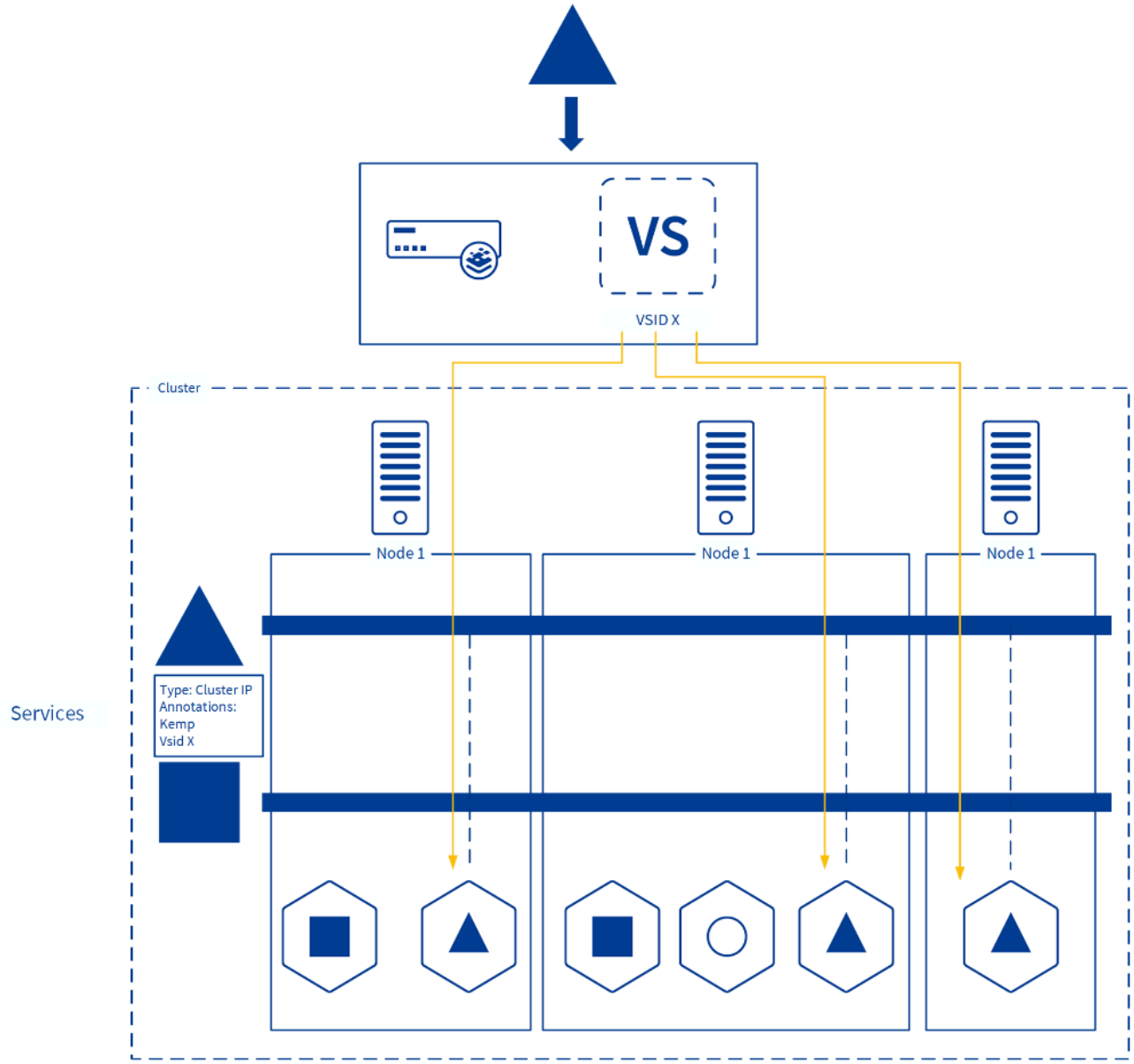
For further details on each mode, refer to the sections below.

Related Links

For details, see the following topics:

- Service Mode
- Ingress Mode

Service Mode



Service mode is an operating mode developed by Progress Kemp to meet the needs of Network Operations and Application Developers to work together more seamlessly despite different tool chains and working practices. This is done by enabling the Network Operations team to create dedicated Virtual Services that can be provided to Application Developers operating in Kubernetes. By dynamically updating based on the state of the defined application in Kubernetes this allows for dynamic application publishing without exposing full access to the LoadMaster.

Service mode allows you to expose a service on the Kubernetes cluster that is not available to access externally by tagging the service with a Virtual Service ID. The LoadMaster makes API calls to determine what pods are linked to the service and then adds the appropriate Real Servers to the Virtual Service.

The advantages and disadvantages of service mode are listed below.

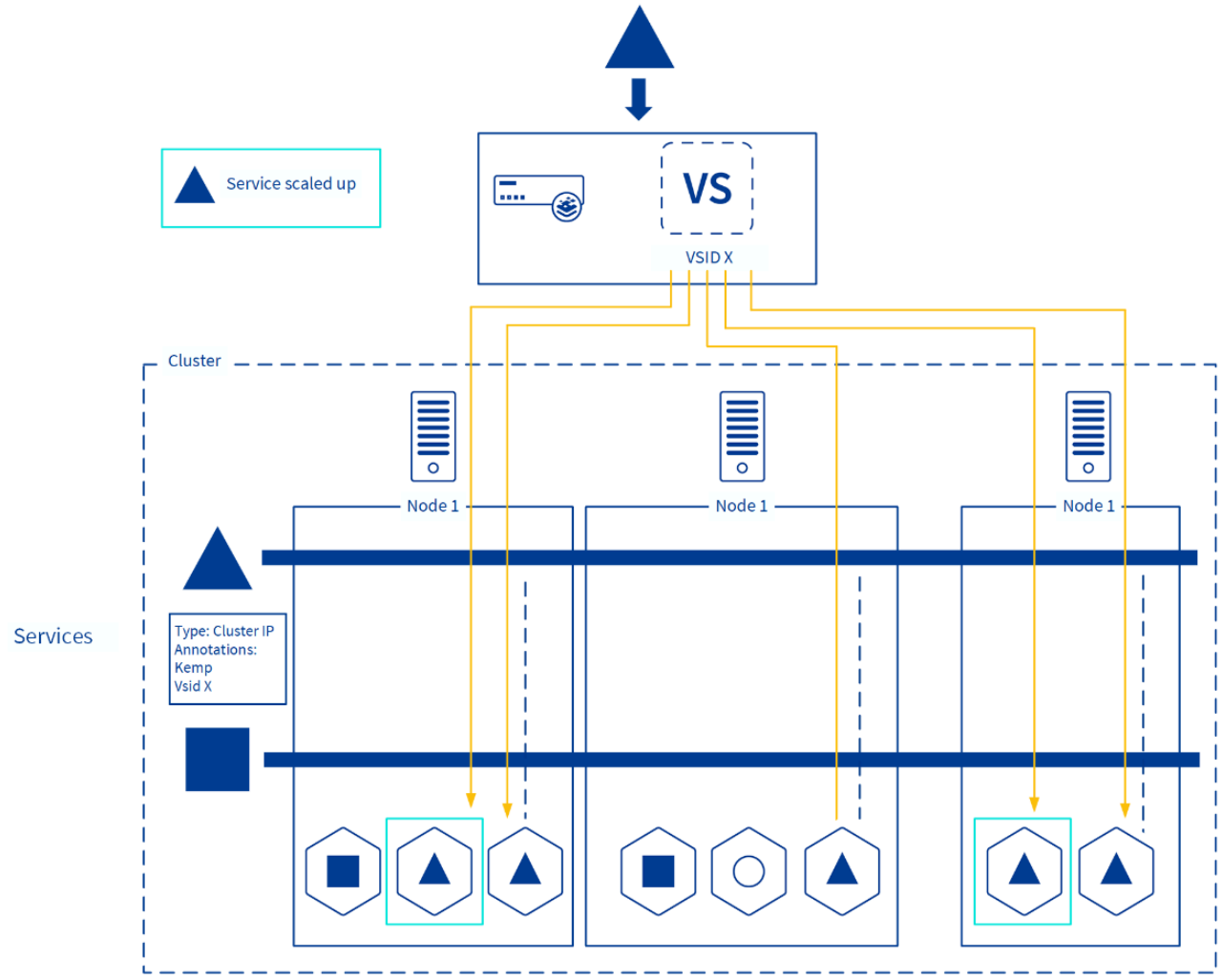
Advantages:

- Efficient routing of traffic to pods
- Eliminates unnecessary East-West traffic
- LoadMaster changes are restricted to the specific Virtual Service defined

Disadvantages:

- May need additional routes to pods defined
- The pod network must not overlap with network IP addresses
- Nodes must be on the same subnet as the LoadMaster
- A single Virtual Service per Kubernetes Service is required

To enable Service Mode, a Kubernetes service is annotated as being "KempLB enabled" along with a Virtual Service reference (typically the Virtual Service ID) to link it to a Virtual Service on the LoadMaster. The LoadMaster monitors the configuration and if it matches an existing Virtual Service, the appropriate Real Servers are created to route traffic into the service pods. In this mode, the Virtual Service must pre-exist and no new Virtual Service is created. If the service scales, more Real Servers will be added.



To illustrate the operation of service mode, if the number of pods scales up from three to five, for example, the Virtual Service updates automatically and will have five Real Servers.

How Service Mode Works

In Service Mode, a Virtual Service is created on the LoadMaster and details of this (for example VSID) is provided to the Application Developers (or persons configuring Kubernetes). When publishing a Kubernetes Service it is labeled as kempLB-enabled and associated with the Virtual Service (for example, through the VSID). The LoadMaster automatically detects this and configures itself to match the Service configuration defined in Kubernetes by adding the appropriate Real Servers. This Virtual Service then dynamically changes when pods are added/created/destroyed.

When to Use Service Mode

Service Mode is suitable if you are a:

- NetOps Team without knowledge or access of the Kubernetes infrastructure

- NetOps Team who uses a different deployment and configuration management tool chain than their AppDev Teams
- Managed Service Provider operating shared Kubernetes and Network infrastructure while their customers self-manage their Kubernetes-based applications

Configure Service Mode

Refer to the sections below for instructions on how to configure service mode.

Configure a LoadMaster an ECS CM Virtual Service

You must configure a Virtual Service:

1. In the LoadMaster UI, go to **Virtual Services > Add New**.
2. Expand the **Real Servers** section.
3. Ensure the **Real Server Check Method** is set to **HTTP Protocol**.
4. Select **GET** as the **HTTP Method**.
5. Do not modify any of the other default settings.
6. Take note of the Virtual Service **Id** number.

Note: The Virtual Service **Id** number is available at the top of the Virtual Service modify screen. This is needed to connect your Kubernetes service to the LoadMaster Virtual Service.

Configure the Kubernetes Service

Follow the steps below to link an existing service in Kubernetes to the LoadMaster Virtual Service:

1. Modify the relevant YAML file.
2. Add the following label:

   kempLB: Enabled

3. Add the following annotation:

   "vsid":"<VirtualServiceID>"

   Note: Throughout this document, angular brackets (**< >**) indicate a value that is variable that you must configure appropriately based on your environment.

   ---

4. In the Kubernetes Shell, apply the following configuration:

   kubectl apply -f <Filename>.yaml

5. In the Kubernetes Shell, run the following command to see the service details:

   kubectl describe service <Name>

   Here is an example of a YAML file:

```
apiVersion: v1
kind: Service
metadata:
        name: <Name>
        labels:
                kempLB: Enabled
        annotations:
                "vsid": "<VirtualServiceID>"
spec:
        type: ClusterIP
        ports:
                - port: <VirtualServicePort>
        selector:
                app: <ApplicationSelector>
```

Verify the Application

The LoadMaster should have picked up that a Kubernetes service has been labeled as **kempLB=Enabled** and configured the LoadMaster Virtual Service with matching VSID with Real Servers corresponding to Kubernetes Services pods. To verify this, follow the steps below:

1. In the LoadMaster UI, go to **Virtual Services > View/Modify Services**.

   Note: You should see the status as up/green and a Real Server matching the IP address of your Kubernetes pod.

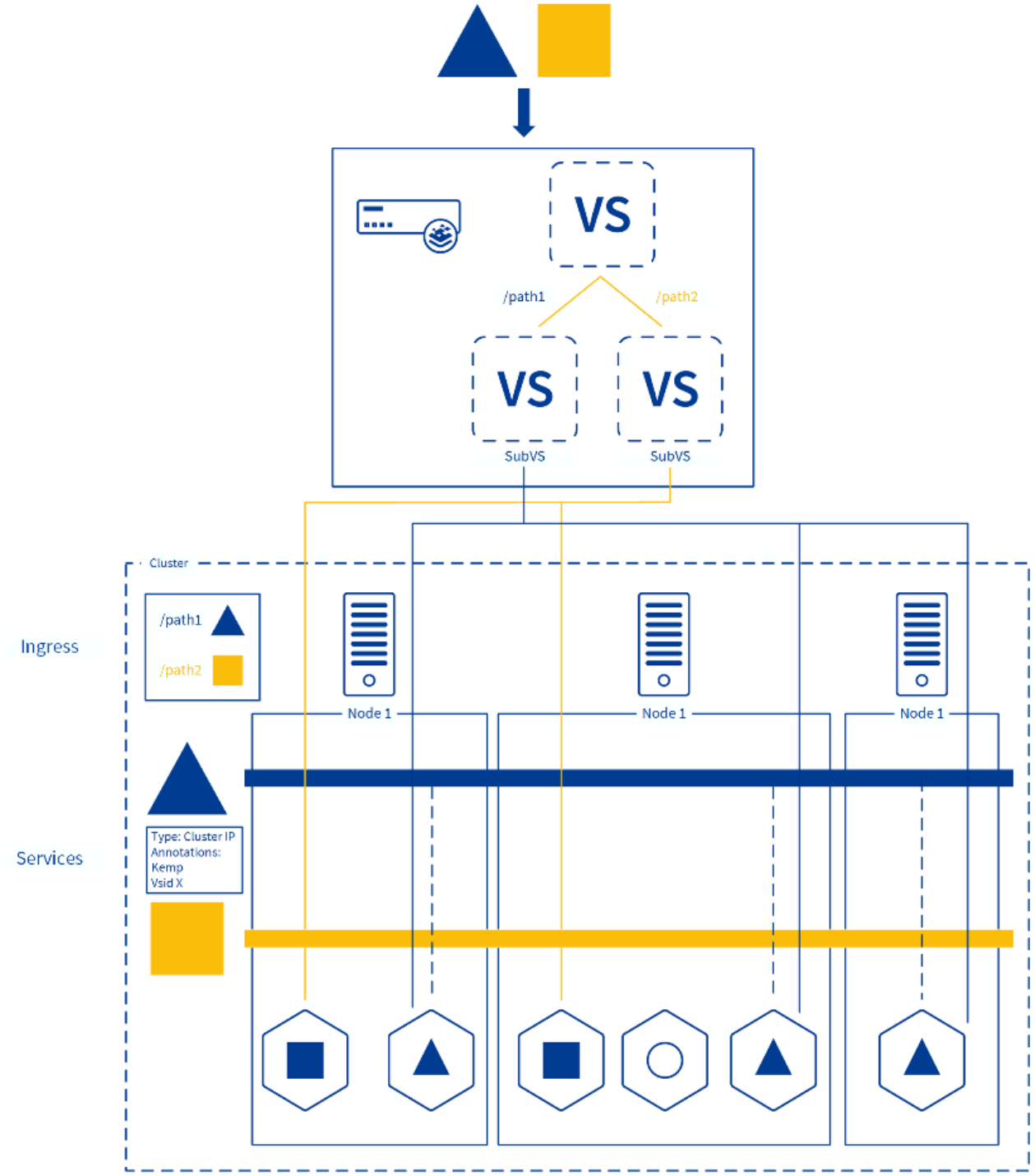
2. Go to the Virtual Service IP address in your browser.

Note: You should be able to open the appropriate application.

If you deploy more replicas of the service, these should also appear as Real Servers in the LoadMaster Virtual Service. In your Kubernetes Shell, run the following command to see this work:

kubectl scale --replicas=2 deployment/<Application>

Ingress Mode



Ingress Mode is designed to allow DevOps Teams to use the LoadMaster as an Ingress Controller for their Kubernetes Clusters in place of containerized Ingress controllers.

Ingress mode allows you to define the Ingress with an Ingress controller. The LoadMaster creates one Virtual

Service with one or more SubVSs. Any Virtual Services created by Kubernetes are labeled as **L7+K8S** in the **Layer** column on the **View/Modify Services** screen.

Each SubVS maps to the corresponding service in Kubernetes as defined by the Ingress Object rules. This means you do not need a separate Virtual Service for each service. The one Virtual Service performs routing based on the path and/or host as defined. If a new pod is added, a new Real Server gets automatically added to the relevant SubVS.

In Ingress Mode, Ingress IP address and port numbers are defined in the Kubernetes Ingress Object in addition to attributes used to route specific traffic to particular service pods. In this mode, the Virtual Service is created based on the information defined (if appropriate values are used). If more traffic paths and services are added, these are updated on the Virtual Service with each service typically represented as a SubVS. If a service scales, more Real Servers are added.

In Ingress mode, it is possible to define multiple Ingress resources. You can map the master Ingress resource (with multiple associated Slave Ingress resources) to one Virtual Service using the following annotation:

kemp.ax/ingress.ref: "<namespace>:<ingressname>"

It is necessary to ensure that either:

- All the Ingress resources are defined in one namespace (Master, Slave, and Services), or;
- The master Ingress is defined in the "default" namespace and the Slave Ingress resources are defined in different namespaces

---

Note: Multiple Ingress resources cannot be mapped to one Virtual Service if the master Ingress is defined in a namespace other than "default" and the Slave Ingress resources are also defined in different namespaces.

Here is an example YAML file:

Note: The new Ingress Class (**ingressClassName: lmingress**) is supported on Kubernetes versions 1.22 and above.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
        name: master-ingress
        namespace: default
        annotations:
                "kemp.ax/vsip": "10.35.46.104"
                "kemp.ax/vsport": "443"
                "kemp.ax/vsprot": "tcp"
                "kemp.ax/vstype": "http"
                "kemp.ax/sslaccel": "1"
                "kemp.ax/certfile": "server"
spec:
        ingressClassName: lmingress
        defaultBackend:
                service:
                        name: qa-default
```

```
                        port:
                                number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
        name: slave-ingress1
        namespace: slave1
        annotations:
                "kemp.ax/ingress.ref": "default/master-ingress"
spec:
        ingressClassName: lmingress
        rules:
        - host: slave1.kemptech.net
                http:
                        paths:
                        - path: /api/v1/deployer
                                pathType: Prefix
                                backend:
                                        service:
                                                name: service1
                                                port:
                                                        number: 80
                        - path: /api/v1/logout
                                pathType: Prefix
                                backend:
                                        service:
                                        name: service1
                                        port:
                                                number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
        name: slave-ingress2
        namespace: slave2
        annotations:
                "kemp.ax/ingress.ref": "default/master-ingress"
spec:
        ingressClassName: lmingress
        rules:
        - host: slave2.kemptech.net
                http:
                        paths:
                        - path: /api/v1/server
                                pathType: Prefix
                                backend:
                                        service:
                                                name: service2
                                                port:
                                                        number: 80
                        - path: /api/v1/authenticate
                                pathType: Prefix
                                backend:
```

```
service:
        name: service2
        port:
                number: 80
```

Note: If the LoadMaster firmware is updated from an older version (other than 7.2.55) that already has an Ingress resource defined, after updating the older configuration of Virtual Services and SubVSes remains unchanged until you delete the older SubVS entries to get the updated rule entries for the SubVSes.

Ingress Mode is the standard Kubernetes Ingress Controller operating mode designed for cross-functional teams operating purely through the Kubernetes API.

The advantages and disadvantages of Ingress mode are listed below.

Advantages:

- Efficient routing of traffic to pods
- Eliminates unnecessary East-West traffic
- Single Virtual Service for multiple services
- No need for double load balancing
- Kubernetes endpoints can be administered along with monolithic load-balanced services
- Create multiple rules for one service/SubVS instead of multiple SubVSs

Disadvantages:

- May need routes to pods defined
- Pod network must not overlap with network IP addresses
- Nodes must be on the same subnet as the LoadMaster

How Ingress Mode Works

The Kubernetes API defines an 'Ingress Controller' object which allows DevOps Teams to publish their Services to external systems and users.

Ingress Mode allows DevOps Teams to define a new class of Ingress Controller using the LoadMaster which automatically detects and matches the configuration as defined in Kubernetes.

The LoadMaster detects when Services Endpoints (that is, Kubernetes Pods) are added or removed and ensures the LoadMaster SubVS Real Servers are updated accordingly.

When to Use Ingress Mode

Ingress Mode is designed and recommended if you are a:

- Cross-functional DevOps Team who own and operate both the network and Kubernetes infrastructure in addition to the applications
- NetOps Team with knowledge of and access to the Kubernetes infrastructure
- NetOps Team who use the same deployment tool chain and processes as their AppDev Teams
- Managed Service Provider operating shared Kubernetes and Network infrastructure in addition to managing their customers' Kubernetes-based applications

Configure Ingress Mode

Refer to the sections below for instructions on how to configure ingress mode.

Create a Kubernetes Ingress Controller

You must define a Kubernetes Ingress Object that stores the configuration that the LoadMaster requires. Here is an example YAML file:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
        name: kemp-ingress
        annotations:
                "kemp.ax/vsip": "<VirtualServiceAddress>"
                "kemp.ax/vsport": "<VirtualServicePort>"
                "kemp.ax/vsprot": "<tcp/udp>"
spec:
        ingressClassName: lmingress
        rules:
        - host: <HostAddress1>
                http:
                        paths:
                        - path: </Path1>
                              pathType: <Prefix/ImplementationSpecific/Exact>
                              backend:
                                service:
                                          name: <ServiceName1>
                                          port:
                                            number: <ServicePort1>
          - host: <HostAddress2>
    http:
      paths:
      - path: </Path2>
        pathType: <Prefix/ImplementationSpecific/Exact>
        backend:
          service:
          name: <ServiceName2>
          port:
            number: <ServicePort2>
```

Note: You cannot use both **spec.ingressClassName** and **metadata.annotations.kubernetes.io/ingress.class** together. If you do, you will get an error like "The Ingress "<ingress-name>" is invalid: annotations.kubernetes.io/ingress.class: Invalid value: "kempLB": can not be set when the class field is also set. However, you can use one of the two methods (annotation or ingressClassName) by itself - just not both.

In your Kubernetes Shell session, run:

kubectl apply -f <Filename>.yaml

In your Kubernetes Shell session, run the following command to see the Ingress details:

kubectl describe ingress <Name>

Verify the Application

The LoadMaster should have picked up that a Kubernetes Ingress controller has been published and configured a new Virtual Service (if necessary) and the required SubVS(es) for each Kubernetes Services pod. To verify this, follow the steps below:

1. In the LoadMaster UI, go to **Virtual Services > View/Modify Services**.

   Note: You should see the status as up/green and a SubVS. If you click the Virtual Service IP address it should expand to show the SubVS with a Real Server matching the IP address of your Kubernetes pod.

2. Modify the hosts file of your client system to resolve an address to the Virtual Service IP address.
3. Go to this address in your browser. This should open the application. By using different hostnames or paths you should be able to access the different services as defined in the Ingress Rules.
4. If you deploy more replicas of the service, these should also appear as Real Servers in the LoadMaster SubVS. To do this, in your Cloud Shell run:

   kubectl scale --replicas=2 deployment/<Application>

6

Service Object Annotations

The following table lists all the annotations that can be used within the Kubernetes YAML configuration file to define various Virtual Service attributes. All of these attributes can be used within **Service** objects in the YAML file, regardless of whether you are using Ingress Mode or Service Mode. The table below lists the annotation name, the corresponding LoadMaster Application Programming Interface (API) command name, and a short description. Refer to the RESTful API documentation for settings and other details about each annotation.

Kubernetes Annotation Name	API Parameter Name	Description
addvia	AddVia	Specific headers to be added to HTTP requests.
alertthresh	AlertThreshold	Web Application Firewall (WAF): This is the threshold of incidents per hour before sending an alert. 0 = disabled.
allowhttp2	AllowHTTP2	Enable/disable HTTP2 for this Virtual Service. SSL Acceleration must be enabled before HTTP2 can be enabled.
altaddress	AltAddress	The alternate address for a Virtual Service.
cache	Cache	Enable/disable the caching of URLs.

Kubernetes Annotation Name	API Parameter Name	Description
cachepercent	CachePercent	Maximum percentage of cache space permitted for a Virtual Service.
certfile	CertFile	A list of certificate identifiers (strings) separated by spaces.
checkhost	CheckHost	Host name for Real Server health checks for a Virtual Service.
checkport	CheckPort	Port number for Real Server health checks for a Virtual Service.
checktype	CheckType	Set the Health Check method (for example, ICMP, TCP, and so on).
checkurl	CheckUrl	URL for Real Server health checks for a Virtual Service.
checkuse1.1	CheckUse1.1	Enable/disable using HTTP 1.1 for health checks for a Virtual Service.
ciphers	Ciphers	A list of cipher names to be used for a Virtual Service separated by colons (:).
cipherset	CipherSet	The cipher set name to be used for a Virtual Service.
clientcert	ClientCert	Enable client certificates for a Virtual Service.
compress	Compress	Enable/disable file compression for a Virtual Service.
copyhdrfrom	CopyHdrFrom	Name of the source header field to copy into a new header field before forwarding request to Real Servers.
copyhdrto	CopyHdrTo	Name of the header field into which the header data from **copyhdrfrom** is to be copied.
defaultgw	DefaultGW	IP address of the default gateway for a Virtual Service. If the **defaultgw** is not set for a Virtual Service, the global Default Gateway value is used.
enable	Enable	Activate/deactivate a Virtual Service.
enhealthchecks	EnhancedHealthChecks	Enable/disable the **rsminimum** parameter. If disabled, only one Real Server being available marks the

Kubernetes Annotation Name	API Parameter Name	Description
		Virtual Service "up".
errorcode	ErrorCode	An HTTP error code to return if no Real Servers are available.
errorurl	ErrorUrl	A redirect URL to be returned if no Real Servers are available.
espenabled	EspEnabled	Enable/disable Edge Security Pack (ESP) features (for example, Single Sign On).
extraports	ExtraPorts	Additional listening ports for the Virtual Service.
followvsid	FollowVSID	Enable/disable port following. 0 is disabled; to enable, set to the Virtual Service ID of the Virtual Service to follow. Virtual Service IDs 1 and 2 cannot be used.
forcel7	ForceL7	Enable/disable using the Layer 7 engine even if the Virtual Service traffic is Layer 4.
idletime	Idletime	The length of time (in seconds) that a Virtual Service connection may remain idle before it is closed. 0 means use the **conntimeout** value.
inauthmode	InputAuthMode	The client authentication mode to be used.
locbindaddr	LocalBindAddrs	A space-separated list of IP addresses to use as alternate source addresses when scaling over 64K connections is enabled.
machlen	MatchLen	Number of bytes to search in server responses when using binary health checks.
multiconnect	MultiConnect	Enable/disable multiplexing of multiple client requests over a single Real Server connection.
needhostname	NeedHostName	When enabled, the host name must be included in a client request or the connection is dropped.
non_local	non_local	Enable non-local Real Servers. Transparency must be disabled on

Kubernetes Annotation Name	API Parameter Name	Description
		the relevant Virtual Services.
ocspverify	OCSPVerify	Enable/disable OCSP verification of client certificates for a Virtual Service.
outauthmode	OutputAuthMode	Specify the Real Server authentication mode to be used.
persist	Persist	The type of persistence (stickiness) to use for a Virtual Service.
perstout	PersistTimeout	The length of time (in seconds) after the last connection that the LoadMaster will remember the persistence information.
portfollow	PortFollow	Deprecated. Use **followvsid** instead.
preprec	PreProcPrecedence	The name of an existing Content Matching Rule whose place in the execution order you want to modify.
preprecpos	PreProcPrecedencePos	An integer specifying the execution order of the Content Matching Rule whose name is given by the value of **preprec**.
qos	QoS	Sets a Type of Service (ToS) value in the IP header of packets outbound from a Virtual Service.
querytag	QueryTag	The query tag to be matched if the **persist** type is set to query-hash.
reqprec	RequestPrecedence	The name of an existing Request Rule whose place in the execution order you want to modify.
reqprecpos	RequestPrecedencePos	An integer specifying the execution order of the Request Rule whose name is given by the value of **reqprec**.
reqrules	RequestRules	Returns the list of request rules that are assigned to the Virtual Service.
respprec	ResponsePrecedence	The name of an existing Response Rule whose place in the execution order you want to modify.
resprecpos	ResponsePrecedencePos	An integer specifying the execution order of the Response Rule whose

Kubernetes Annotation Name	API Parameter Name	Description
		name is given by the value of **respprec**.
resprules	ResponseRules	Returns the list of response rules that are assigned to the Virtual Service.
rsminimum	RsMinimum	The minimum number of Real Servers required to be available for the Virtual Service to be considered up.
rsnihostname	ReverseSNIHostname	The SNI Hostname to use when connecting to Real Servers.
shed	Schedule	The scheduling or load balancing method for a Virtual Service.
sechdropt	SecurityHeaderOptions	Add the Strict-Transport-Security header to all LoadMaster-generated messages (ESP and error messages).
serverinit	ServerInit	Permit local connections to the Real Server before any client connections have been received for the Virtual Service.
sslaccel	SSLAcceleration	Enables/disables SSL acceleration (decryption) for incoming Virtual Service traffic.
sslreencrypt	SSLReencrypt	Enables/disables SSL encryption on connections to Real Servers.
sslreverse	SSLReverse	Enabling this parameter means that the data from the LoadMaster to the Real Server is re-encrypted. This is only relevant for Virtual Services with the **Service Type** set to **Generic**.
sslrewrite	SSLRewrite	Enable/disable rewriting of location URLs when a redirect is being used.
standbyaddr	StandbyAddr	The IP address of the **Sorry** server that is to be used when no Real Servers are available.
standbyport	StandbyPort	The port number of the **Sorry** server.
starttlsmode	StartTLSMode	Set the mode used for HTTP/HTTPS and STARTTLS type Virtual Services.

Kubernetes Annotation Name	API Parameter Name	Description
subnetorig	SubnetOriginating	Enable/disable using LoadMaster's subnet IP address as the source IP for traffic originating from a Real Server on a subnet configured on the system.
tlstype	TLSType	Specifies the SSL/TLS versions supported by a Virtual Service.
transparent	Transparent	Enable/Disable transparency on a Virtual Service.
useforsnatt	UseforSnatt	Enable/disable use of the Virtual Service IP address as the source address for outbound packets from Real Servers.
verify	Verify	WAF: Enable/disable intrusion detection and behavior.
vsip	VSAddress	The IP Address of the Virtual Service.
vsname	NickName	The name of the Virtual Service.
vsport	Protocol	The port number of the Virtual Service.
vstype	VStype	The type of the Virtual Service.

7

Ensuring Connectivity from the LoadMaster ECS CM to Pods

When running Kubernetes there are multiple options for the configuration of how pods can communicate. Kubernetes requirements are that pods on a node can communicate with all pods on all nodes without Network Address Translation (NAT). Agents on a node (for example, system daemons, kubelet) can communicate with all pods on that node, and pods in the host network of a node can communicate with all pods on all nodes without NAT. For further details, refer to the following Kubernetes page: Cluster Networking.

There are several ways to implement these and it varies based on the Container Network Interface (CNI) plugins used, cloud provider integrations, or any Border Gateway Protocol (BGP) peering with the physical network in use.

If the pods are routable from outside the network there is no requirement for outbound Secure Network Address Translation (SNAT) and pods can be accessed directly without using Kubernetes Services. The disadvantage is that the IP addresses used must be reserved for pod use.

Related Links

For details, see the following topics:

- Overlay Networks
- How to Check if an Overlay Network is in Use
- LoadMaster ECS CM to Pod Connectivity
- Adding Routes

Overlay Networks

An overlay network allows network devices to communicate across an underlying network (referred to as the underlay) without the underlay network having any knowledge of the devices connected to the overlay network.

There are several common overlay network types that can be used such as Flannel Calico and Weave Net. Typically, when using an overlay network, pods are assigned with an address from a set of addresses assigned to the overlay network and this is different from the Host Network in which the Node's IP addresses are from.

In most cases when using an overlay network, it is typical that pods are not routable.

How to Check if an Overlay Network is in Use

If you are unsure, the simplest way to check if an overlay network is used and pods are not routable is to view the pod IP addresses and compare them to the IP addresses of the nodes. If these are on a different network space then routes must be added.

For example:

[root@master-node ~]# kubectl get nodes -o wide

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
master-node	Ready	master	46d	v1.	19.2 10.1.151. 30	<none>	CentOS	Linux 7 (Core)	3.10.0-11 27.19.1.e l7.x86_6 4 docker:// 19.3.13
node-1	Ready	<none>	46d	v1.	19.2 10.1.151. 31	<none>	CentOS	Linux 7 (Core)	3.10.0-11 27.19.1.e l7.x86_6 4 docker:// 19.3.13
node-2	Ready	<none>	46d	v1.	19.2 10.1.151. 32	<none>	CentOS	Linux 7 (Core)	3.10.0-11 27.19.1.e l7.x86_6 4 docker:// 19.3.13

[root@master-node ~]#

[root@master-node ~]# kubectl get pods -o wide

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
azure-vote-back-59d587dbb7-x297k	1/1	Running	1	3d	192.168.84.150	node-1	<none>	<none>		
azurefront-5f47b44b9b-nhdwb	1/1	Running	1	3d	192.168.247.14	node-2	<none>	<none>		
frontend-6c6d6dfd4d-6xzpd	1/1	Running	1	3d	192.168.247.2	node-2	<none>	<none>		
frontend-6c6d6dfd4d-g26cb	1/1	Running	1	3d	192.168.247.1	node-2	<none>	<none>		
frontend-6c6d6dfd4d-qbbsv	1/1	Running	1	3d	192.168.84.155	node-1	<none>	<none>		
redis-master-f46ff57fd-jjhwx	1/1	Running	1	3d	192.168.84.153	node-1	<none>	<none>		
redis-slave-bbc7f655d-5xnkk	1/1	Running	1	3d	192.168.84.156	node-1	<none>	<none>		
redis-slave-bbc7f655d-n9vgh	1/1	Running	1	3d	192.168.247.12	node-2	<none>	<none>		

In the above example, it is clear that the PODs are on a different network to the nodes.

LoadMaster ECS CM to Pod Connectivity

When using the Ingress Controller for Kubernetes, because the LoadMaster operates outside the Kubernetes Cluster, when it receives traffic it needs a mechanism to route to the correct pods. This means there must be network connectivity between the LoadMaster and the pods. The simplest way to achieve this is to manually create the necessary routes. It is important to note that to do this the LoadMaster must have an interface in the same network as the Kubernetes Hosts.

Adding Routes

To see what routes are necessary, you can use the following Kubernetes commands.

For most CNIs the following can be used:

```
kubectl get nodes -o jsonpath="{range .items[*]}{'Destination:
'}{.spec.podCIDR}{'\t'}{'Gateway: '}{.status.addresses[0].address}{'\n'}{end}"
```

For example:

```
[root@master-node ~]# kubectl get nodes -o jsonpath="{range
.items[*]}{'Destination: '}{.spec.podCIDR}{'\t'}{'Gateway:
'}{.status.addresses[0].address}{'\n'}{end}"

Destination: 192.168.0.0/24        Gateway: 10.1.151.30

Destination: 192.168.1.0/24        Gateway: 10.1.151.31

Destination: 192.168.2.0/24        Gateway: 10.1.151.32
```

The above command tells us what routes are needed on the LoadMaster and you can add these routes in the LoadMaster UI by going to **System Configuration > Network Setup > Additional Routes**.

Destination	Gateway
192.168.0.0/24	10.1.151.30
192.168.1.0/24	10.1.151.31
192.168.2.0/24	10.1.151.32

Calico

If Calico is being used for the overlay network, the commands to use are a little different.

```
kubectl get nodes -o jsonpath="{range .items[*]}{'Destination:
}{.metadata.annotations.projectcalico\.org/IPv4IPIPTunnelAddr}{'\tGateway:
'}{.metadata.annotations.projectcalico\.org/IPv4Address}{'\n'}"

kubectl get IPpool -o jsonpath="{range .items[*]}{'Destination Mask:
/'}{.spec.blockSize}{'\n'}"
```

For example:

```
[root@master-node ~]# kubectl get nodes -o jsonpath="{range
.items[*]}{'Destination: '}{.metadata.annotations.projectcalico\.org/
IPv4IPIPTunnelAddr}{'\tGateway: '}{.metadata.annotations.projectcalico\.org/
IPv4Address}{'\n'}"

Destination: 192.168.77.128        Gateway: 10.1.151.30/24

Destination: 192.168.84.128        Gateway: 10.1.151.31/24

Destination: 192.168.247.0        Gateway: 10.1.151.32/24

Destination:         Gateway:
```

Note: The above does not provide the netmask to use for the Destination Address Space. By default, Calico uses /26 but this can be verified using the following command.

```
[root@master-node ~]# kubectl get IPpool -o jsonpath="{range
.items[*]}{'Destination Mask: /'}{.spec.blockSize}{'\n'}"

Destination Mask: /26

Destination Mask: /

[root@master-node ~]#
```

Fixed Static Routes

Destination	Gateway	Operation
192.168.77.128/26	10.1.151.30	Delete
192.168.84.128/26	10.1.151.31	Delete
192.168.247.0/26	10.1.151.32	Delete

Routes to add to the LoadMaster (in **System Configuration > Network Setup > Additional Routes**):

Destination	Gateway
192.168.77.128/26	10.1.151.30
192.168.84.128/26	10.1.151.31
192.168.247.0/26	10.1.151.32

Note: The subnet mask defaults to /26 for Calico.

8

Creation and Deletion of Objects

When using the Ingress Controller for Kubernetes it is important to understand the expected behavior concerning the deletion and creation of Virtual Services and Real Servers on the LoadMaster.

'Desired State' is one of the core principles of Kubernetes, whereby a desired configuration is defined and the Kubernetes Controller monitors this and implements any changes required to meet the desired state (for example creating/destroying pods in the cluster). When using the Ingress Controller for Kubernetes, the LoadMaster operates similarly. The LoadMaster periodically monitors the desired state of how traffic should reach services and updates the configuration (if needed) to reflect this.

It is important to understand some expected behavior when using the Ingress Controller for Kubernetes in the two modes of operation:

- Service Mode: In this mode, a service may be annotated with attributes to link it to an existing Virtual Service (typically the Virtual Service ID) on the LoadMaster. The LoadMaster monitors the configuration and if it matches an existing Virtual Service, the appropriate Real Servers are created to route traffic into the service pods. In this mode, the Virtual Service must pre-exist and no new Virtual Services are created. If the service scales, more Real Servers are added.If Real Servers are manually deleted on the LoadMaster, the service definition will still map to an existing Virtual Service and will simply re-add these Real Servers. However, If a Virtual Service is deleted, the service will no longer have a valid Virtual Service defined in the Service configuration and no Virtual Service will be recreated. You can delete a Virtual Service using the LoadMaster (along with removing the Virtual Service ID on the Service) but when removing a Real Server this should be done using Kubernetes pod scaling.

- Ingress Mode: In Ingress Mode, the LoadMaster uses the Ingress Object created in Kubernetes. This defines Ingress IP:Port numbers in addition to other attributes used to route to particular services. In this mode, the Virtual Service itself is created based on the information defined here (if appropriate values are used). If more traffic paths and services are added, these are updated on the Virtual Service with each service typically represented as a SubVS. If any one service scales, more Real Servers are added. If the

Virtual Service or Real Server is deleted on the LoadMaster, this results in the Virtual Service/Real Server being recreated as long as the Ingress object still exists in Kubernetes which defined these as a desired state. When deleting a Virtual Service, you should do this using the Kubernetes Ingress Object and when adding/removing Real Servers this should be done using Kubernetes pod scaling.

9

LoadMaster ECS CM Ingress Logs

You can access the Ingress Controller logs by going to **System Configuration > Logging Options > Extended Log Files**. There are two log types:

- **Ingress Controller Logs**
- **Ingress Resource Watcher Logs**

The Ingress logs are only shown when:

- The Ingress Controller is installed, and;
- Ingress logs exist

The following information can be seen in the logs:

- Issues connecting with Kubernetes
- Virtual Services created by the Ingress Controller
- Changes made as a result of scaling
- Configuration changes
- Unrecognized Virtual Service IDs
- Incorrect YAML configuration