



Red Hat OpenShift

Ingress Operator in OpenShift Container Platform

- [OpenShift Container Platform Ingress Operator](#)
- [The Ingress configuration asset](#)
- [Ingress Controller configuration parameters](#)
 - [Ingress Controller TLS security profiles](#)
- [View the default Ingress Controller](#)
- [View Ingress Operator status](#)
- [View Ingress Controller logs](#)
- [View Ingress Controller status](#)
- [Configuring the Ingress Controller](#)
 - [Setting a custom default certificate](#)
 - [Removing a custom default certificate](#)
 - [Autoscaling an Ingress Controller](#)
 - [Scaling an Ingress Controller](#)
 - [Configuring Ingress access logging](#)
 - [Setting Ingress Controller thread count](#)
 - [Configuring an Ingress Controller to use an internal load balancer](#)
 - [Configuring global access for an Ingress Controller on GCP](#)
 - [Setting the Ingress Controller health check interval](#)
 - [Configuring the default Ingress Controller for your cluster to be internal](#)
 - [Configuring the route admission policy](#)
 - [Using wildcard routes](#)
 - [HTTP header configuration](#)
 - [Setting or deleting HTTP request and response headers in an Ingress Controller](#)
 - [Using X-Forwarded headers](#)
 - [Enabling HTTP/2 Ingress connectivity](#)
 - [Configuring the PROXY protocol for an Ingress Controller](#)
 - [Specifying an alternative cluster domain using the appsDomain option](#)
 - [Converting HTTP header case](#)
 - [Using router compression](#)
 - [Exposing router metrics](#)
 - [Customizing HAProxy error code response pages](#)

- [Setting the Ingress Controller maximum connections](#)
- [Additional resources](#)

OpenShift Container Platform Ingress Operator

When you create your OpenShift Container Platform cluster, pods and services running on the cluster are each allocated their own IP addresses. The IP addresses are accessible to other pods and services running nearby but are not accessible to outside clients. The Ingress Operator implements the `IngressController` API and is the component responsible for enabling external access to OpenShift Container Platform cluster services.

The Ingress Operator makes it possible for external clients to access your service by deploying and managing one or more HAProxy-based [Ingress Controllers](#) to handle routing. You can use the Ingress Operator to route traffic by specifying OpenShift Container Platform `Route` and Kubernetes `Ingress` resources. Configurations within the Ingress Controller, such as the ability to define `endpointPublishingStrategy` type and internal load balancing, provide ways to publish Ingress Controller endpoints.

The Ingress configuration asset

The installation program generates an asset with an `Ingress` resource in the `config.openshift.io` API group, `cluster-ingress-02-config.yml`.

YAML Definition of the `Ingress` resource

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: apps.openshift demos.com
```

The installation program stores this asset in the `cluster-ingress-02-config.yml` file in the `manifests/` directory. This `Ingress` resource defines the cluster-wide configuration for Ingress. This Ingress configuration is used as follows:

- The Ingress Operator uses the domain from the cluster Ingress configuration as the domain for the default Ingress Controller.

- The OpenShift API Server Operator uses the domain from the cluster Ingress configuration. This domain is also used when generating a default host for a `Route` resource that does not specify an explicit host.

Ingress Controller configuration parameters

The `ingresscontrollers.operator.openshift.io` resource offers the following configuration parameters.

| Parameter | Description |
|-----------------------|---|
| <code>domain</code> | <p><code>domain</code> is a DNS name serviced by the Ingress Controller and is used to configure multiple features:</p> <ul style="list-style-type: none">■ For the <code>LoadBalancerService</code> endpoint publishing strategy, <code>domain</code> is used to configure DNS records. See <code>endpointPublishingStrategy</code>.■ When using a generated default certificate, the certificate is valid for <code>domain</code> and its <code>subdomains</code>. See <code>defaultCertificate</code>.■ The value is published to individual <code>Route</code> statuses so that users know where to target external DNS records. <p>The <code>domain</code> value must be unique among all Ingress Controllers and cannot be updated.</p> <p>If empty, the default value is <code>ingress.config.openshift.io/cluster.spec.domain</code>.</p> |
| <code>replicas</code> | <p><code>replicas</code> is the desired number of Ingress Controller replicas. If not set, the default value is <code>2</code>.</p> |




| Parameter | Description |
|----------------------------|--|
| endpointPublishingStrategy | <p><code>endpointPublishingStrategy</code> is used to publish the Ingress Controller endpoints to other networks, enable load balancer integrations, and provide access to other systems.</p> <p>On GCP, AWS, and Azure you can configure the following <code>endpointPublishingStrategy</code> fields:</p> <ul style="list-style-type: none"> ▪ <code>loadBalancer.scope</code> ▪ <code>loadBalancer.allowedSourceRanges</code> <p>If not set, the default value is based on <code>infrastructure.config.openshift.io/cluster.status.platform</code>:</p> <ul style="list-style-type: none"> ▪ Azure: <code>LoadBalancerService</code> (with External scope) ▪ Google Cloud Platform (GCP): <code>LoadBalancerService</code> (with External scope) ▪ Bare metal: <code>NodePortService</code> ▪ Other: <code>HostNetwork</code> |

| Parameter | Description |
|-----------|---|
| | <p>HostNetwork has a <code>hostNetwork</code> field with the following default values for the optional binding ports: <code>httpPort: 80</code>, <code>httpsPort: 443</code>, and <code>statsPort: 1936</code>. With the binding ports, you can deploy multiple Ingress Controllers on the same node for the HostNetwork strategy.</p> <p>Example</p> <pre> apiVersion: operator.openshift.io/v1 kind: IngressController metadata: name: internal namespace: openshift- ingress-operator spec: domain: example.com endpointPublishingStrategy: type: HostNetwork hostNetwork: httpPort: 80 httpsPort: 443 statsPort: 1936 </pre> |
| | <p>On Red Hat OpenStack Platform (RHOSP), the LoadBalancerService endpoint publishing strategy is only supported if a cloud provider is configured to create health monitors. For RHOSP 16.2, this strategy is only possible if you use the Amphora Octavia provider.</p> <p>For more information, see the "Setting cloud provider options" section of the RHOSP installation documentation.</p> |

For most platforms, the `endpointPublishingStrategy` value can be updated. On GCP, you can configure the following `endpointPublishingStrategy` fields:

- `loadBalancer.scope`

| Parameter | Description |
|---------------------------------|--|
| | <ul style="list-style-type: none"> ▪ <code>loadbalancer.providerParameters.gcp.clientAccess</code> ▪ <code>hostNetwork.protocol</code> ▪ <code>nodePort.protocol</code> |
| <code>defaultCertificate</code> | <p>The <code>defaultCertificate</code> value is a reference to a secret that contains the default certificate that is served by the Ingress Controller. When Routes do not specify their own certificate, <code>defaultCertificate</code> is used.</p> <p>The secret must contain the following keys and data: * <code>tls.crt</code> : certificate file contents * <code>tls.key</code> : key file contents</p> <p>If not set, a wildcard certificate is automatically generated and used. The certificate is valid for the Ingress Controller <code>domain</code> and <code>subdomains</code>, and the generated certificate's CA is automatically integrated with the cluster's trust store.</p> <p>The in-use certificate, whether generated or user-specified, is automatically integrated with OpenShift Container Platform built-in OAuth server.</p> |
| <code>namespaceSelector</code> | <p><code>namespaceSelector</code> is used to filter the set of namespaces serviced by the Ingress Controller. This is useful for implementing shards.</p> |
| <code>routeSelector</code> | <p><code>routeSelector</code> is used to filter the set of Routes serviced by the Ingress Controller. This is useful for implementing shards.</p> |

| Parameter | Description |
|--------------------|---|
| nodePlacement | <p><code>nodePlacement</code> enables explicit control over the scheduling of the Ingress Controller.</p> <p>If not set, the defaults values are used.</p> <div>  <div> <p>The <code>nodePlacement</code> parameter includes two parts, <code>nodeSelector</code> and <code>tolerations</code>. For example:</p> <pre>nodePlacement: nodeSelector: matchLabels: kubernetes.io/os: linux tolerations: - effect: NoSchedule operator: Exists</pre> </div> </div> |
| tlsSecurityProfile | <p><code>tlsSecurityProfile</code> specifies settings for TLS connections for Ingress Controllers.</p> <p>If not set, the default value is based on the <code>apiservers.config.openshift.io/cluster</code> resource.</p> <p>When using the <code>Old</code>, <code>Intermediate</code>, and <code>Modern</code> profile types, the effective profile configuration is subject to change between releases. For example, given a specification to use the <code>Intermediate</code> profile deployed on release <code>X.Y.Z</code>, an upgrade to release <code>X.Y.Z+1</code> may cause a new profile configuration to be applied to the Ingress Controller, resulting in a rollout.</p> <p>The minimum TLS version for Ingress Controllers is <code>1.1</code>, and the maximum TLS version is <code>1.3</code>.</p> <div>  <div> <p>Ciphers and the minimum TLS version of the configured security profile are reflected in the <code>TLSProfile</code> status.</p> </div> </div> <div>  <div> <p>The Ingress Operator converts the TLS <code>1.0</code> of an <code>Old</code> or <code>Custom</code> profile to <code>1.1</code>.</p> </div> </div> |

| Parameter | Description |
|------------------------|---|
| <code>clientTLS</code> | <p><code>clientTLS</code> authenticates client access to the cluster and services; as a result, mutual TLS authentication is enabled. If not set, then client TLS is not enabled.</p> <p><code>clientTLS</code> has the required subfields, <code>spec.clientTLS.clientCertificatePolicy</code> and <code>spec.clientTLS.ClientCA</code>.</p> <p>The <code>ClientCertificatePolicy</code> subfield accepts one of the two values: <code>Required</code> or <code>Optional</code>. The <code>ClientCA</code> subfield specifies a config map that is in the openshift-config namespace. The config map should contain a CA certificate bundle.</p> <p>The <code>AllowedSubjectPatterns</code> is an optional value that specifies a list of regular expressions, which are matched against the distinguished name on a valid client certificate to filter requests. The regular expressions must use PCRE syntax. At least one pattern must match a client certificate's distinguished name; otherwise, the Ingress Controller rejects the certificate and denies the connection. If not specified, the Ingress Controller does not reject certificates based on the distinguished name.</p> |

| Parameter | Description |
|----------------|--|
| routeAdmission | <p><code>routeAdmission</code> defines a policy for handling new route claims, such as allowing or denying claims across namespaces.</p> <p><code>namespaceOwnership</code> describes how hostname claims across namespaces should be handled. The default is <code>Strict</code>.</p> <ul style="list-style-type: none">▪ <code>Strict</code> : does not allow routes to claim the same hostname across namespaces.▪ <code>InterNamespaceAllowed</code> : allows routes to claim different paths of the same hostname across namespaces. <p><code>wildcardPolicy</code> describes how routes with wildcard policies are handled by the Ingress Controller.</p> <ul style="list-style-type: none">▪ <code>WildcardsAllowed</code> : Indicates routes with any wildcard policy are admitted by the Ingress Controller.▪ <code>WildcardsDisallowed</code> : Indicates only routes with a wildcard policy of <code>None</code> are admitted by the Ingress Controller. Updating <code>wildcardPolicy</code> from <code>WildcardsAllowed</code> to <code>WildcardsDisallowed</code> causes admitted routes with a wildcard policy of <code>Subdomain</code> to stop working. These routes must be recreated to a wildcard policy of <code>None</code> to be readmitted by the Ingress Controller. <code>WildcardsDisallowed</code> is the default setting. |

| Parameter | Description |
|--------------------------|---|
| IngressControllerLogging | <p><code>logging</code> defines parameters for what is logged where. If this field is empty, operational logs are enabled but access logs are disabled.</p> <ul style="list-style-type: none"> ▪ <code>access</code> describes how client requests are logged. If this field is empty, access logging is disabled. ▪ <code>destination</code> describes a destination for log messages. <ul style="list-style-type: none"> ▪ <code>type</code> is the type of destination for logs: <ul style="list-style-type: none"> ▪ <code>Container</code> specifies that logs should go to a sidecar container. The Ingress Operator configures the container, named logs, on the Ingress Controller pod and configures the Ingress Controller to write logs to the container. The expectation is that the administrator configures a custom logging solution that reads logs from this container. Using container logs means that logs may be dropped if the rate of logs exceeds the container runtime capacity or the custom logging solution capacity. ▪ <code>Syslog</code> specifies that logs are sent to a Syslog endpoint. The administrator must specify an endpoint that can receive Syslog messages. The expectation is that the administrator has configured a custom Syslog instance. ▪ <code>container</code> describes parameters for the <code>Container</code> logging destination type. Currently there are no parameters for container logging, so this field must be empty. ▪ <code>syslog</code> describes parameters for the <code>Syslog</code> logging destination type: <ul style="list-style-type: none"> ▪ <code>address</code> is the IP address of the syslog endpoint that receives log messages. ▪ <code>port</code> is the UDP port number of the syslog endpoint that receives log messages. |

| Parameter | Description |
|-----------|---|
| | <ul style="list-style-type: none"> ■ <code>maxLength</code> is the maximum length of the syslog message. It must be between 480 and 4096 bytes. If this field is empty, the maximum length is set to the default value of 1024 bytes. ■ <code>facility</code> specifies the syslog facility of log messages. If this field is empty, the facility is <code>local1</code>. Otherwise, it must specify a valid syslog facility: <code>kern</code>, <code>user</code>, <code>mail</code>, <code>daemon</code>, <code>auth</code>, <code>syslog</code>, <code>lpr</code>, <code>news</code>, <code>uucp</code>, <code>cron</code>, <code>auth2</code>, <code>ftp</code>, <code>ntp</code>, <code>audit</code>, <code>alert</code>, <code>cron2</code>, <code>local0</code>, <code>local1</code>, <code>local2</code>, <code>local3</code>, <code>local4</code>, <code>local5</code>, <code>local6</code>, or <code>local7</code>. ■ <code>httpLogFormat</code> specifies the format of the log message for an HTTP request. If this field is empty, log messages use the implementation's default HTTP log format. For HAProxy's default HTTP log format, see the HAProxy documentation. |

| Parameter | Description |
|-------------|---|
| httpHeaders | <p><code>httpHeaders</code> defines the policy for HTTP headers.</p> <p>By setting the <code>forwardedHeaderPolicy</code> for the <code>IngressControllerHTTPHeaders</code>, you specify when and how the Ingress Controller sets the <code>Forwarded</code>, <code>X-Forwarded-For</code>, <code>X-Forwarded-Host</code>, <code>X-Forwarded-Port</code>, <code>X-Forwarded-Proto</code>, and <code>X-Forwarded-Proto-Version</code> HTTP headers.</p> <p>By default, the policy is set to <code>Append</code>.</p> <ul style="list-style-type: none"> ▪ <code>Append</code> specifies that the Ingress Controller appends the headers, preserving any existing headers. ▪ <code>Replace</code> specifies that the Ingress Controller sets the headers, removing any existing headers. ▪ <code>IfNone</code> specifies that the Ingress Controller sets the headers if they are not already set. ▪ <code>Never</code> specifies that the Ingress Controller never sets the headers, preserving any existing headers. <p>By setting <code>headerNameCaseAdjustments</code>, you can specify case adjustments that can be applied to HTTP header names. Each adjustment is specified as an HTTP header name with the desired capitalization. For example, specifying <code>X-Forwarded-For</code> indicates that the <code>x-forwarded-for</code> HTTP header should be adjusted to have the specified capitalization.</p> <p>These adjustments are only applied to cleartext, edge-terminated, and re-encrypt routes, and only when using HTTP/1.</p> <p>For request headers, these adjustments are applied only for routes that have the <code>haproxy.router.openshift.io/h1-adjust-case=true</code> annotation. For response headers, these adjustments are applied to all HTTP responses. If this field is empty, no request headers are adjusted.</p> <p><code>actions</code> specifies options for performing certain actions on headers. Headers cannot be set or deleted for TLS passthrough connections. The <code>actions</code> field has additional subfields <code>spec.httpHeader.actions.response</code> and <code>spec.httpHeader.actions.request</code>:</p> <ul style="list-style-type: none"> ▪ The <code>response</code> subfield specifies a list of HTTP response headers to set or delete. ▪ The <code>request</code> subfield specifies a list of HTTP request headers to set or delete. |

| Parameter | Description |
|---------------------------------|---|
| <code>httpCompression</code> | <p><code>httpCompression</code> defines the policy for HTTP traffic compression.</p> <ul style="list-style-type: none"> ▪ <code>mimeType</code> defines a list of MIME types to which compression should be applied. For example, <code>text/css; charset=utf-8</code>, <code>text/html</code>, <code>text/*</code>, <code>image/svg+xml</code>, <code>application/octet-stream</code>, <code>X-custom/customsub</code>, using the format pattern, <code>type/subtype; [;attribute=value]</code>. The types are: application, image, message, multipart, text, video, or a custom type prefaced by <code>X-</code>; e.g. To see the full notation for MIME types and subtypes, see RFC1341 |
| <code>httpErrorCodePages</code> | <p><code>httpErrorCodePages</code> specifies custom HTTP error code response pages. By default, an IngressController uses error pages built into the IngressController image.</p> |
| <code>httpCaptureCookies</code> | <p><code>httpCaptureCookies</code> specifies HTTP cookies that you want to capture in access logs. If the <code>httpCaptureCookies</code> field is empty, the access logs do not capture the cookies.</p> <p>For any cookie that you want to capture, the following parameters must be in your <code>IngressController</code> configuration:</p> <ul style="list-style-type: none"> ▪ <code>name</code> specifies the name of the cookie. ▪ <code>maxLength</code> specifies the maximum length of the cookie. ▪ <code>matchType</code> specifies if the field <code>name</code> of the cookie exactly matches the capture cookie setting or is a prefix of the capture cookie setting. The <code>matchType</code> field uses the <code>Exact</code> and <code>Prefix</code> parameters. <p>For example:</p> <pre>httpCaptureCookies: - matchType: Exact maxLength: 128 name: MYCOOKIE</pre> |

| Parameter | Description |
|--------------------|---|
| httpCaptureHeaders | <p><code>httpCaptureHeaders</code> specifies the HTTP headers that you want to capture in the access logs. If the <code>httpCaptureHeaders</code> field is empty, the access logs do not capture the headers.</p> <p><code>httpCaptureHeaders</code> contains two lists of headers to capture in the access logs. The two lists of header fields are <code>request</code> and <code>response</code>. In both lists, the <code>name</code> field must specify the header name and the <code>maxLength</code> field must specify the maximum length of the header. For example:</p> <pre>httpCaptureHeaders: request: - maxLength: 256 name: Connection - maxLength: 128 name: User-Agent response: - maxLength: 256 name: Content-Type - maxLength: 256 name: Content-Length</pre> |

| Parameter | Description |
|---------------|--|
| tuningOptions | <p>tuningOptions specifies options for tuning the performance of Ingress Controller pods.</p> <ul style="list-style-type: none"> ■ <code>clientFinTimeout</code> specifies how long a connection is held open while waiting for the client response to the server closing the connection. The default timeout is <code>1s</code>. ■ <code>clientTimeout</code> specifies how long a connection is held open while waiting for a client response. The default timeout is <code>30s</code>. ■ <code>headerBufferBytes</code> specifies how much memory is reserved, in bytes, for Ingress Controller connection sessions. This value must be at least <code>16384</code> if HTTP/2 is enabled for the Ingress Controller. If not set, the default value is <code>32768</code> bytes. Setting this field not recommended because <code>headerBufferBytes</code> values that are too small can break the Ingress Controller, and <code>headerBufferBytes</code> values that are too large could cause the Ingress Controller to use significantly more memory than necessary. ■ <code>headerBufferMaxRewriteBytes</code> specifies how much memory should be reserved, in bytes, from <code>headerBufferBytes</code> for HTTP header rewriting and appending for Ingress Controller connection sessions. The minimum value for <code>headerBufferMaxRewriteBytes</code> is <code>4096</code>. <code>headerBufferBytes</code> must be greater than <code>headerBufferMaxRewriteBytes</code> for incoming HTTP requests. If not set, the default value is <code>8192</code> bytes. Setting this field not recommended because <code>headerBufferMaxRewriteBytes</code> values that are too small can break the Ingress Controller and <code>headerBufferMaxRewriteBytes</code> values that are too large could cause the Ingress Controller to use significantly more memory than necessary. ■ <code>healthCheckInterval</code> specifies how long the router waits between health checks. The default is <code>5s</code>. ■ <code>serverFinTimeout</code> specifies how long a connection is held open while waiting for the server response to the client that is closing the connection. The default timeout is <code>1s</code>. ■ <code>serverTimeout</code> specifies how long a connection is held open while waiting for a server response. The default timeout is <code>30s</code>. |

| Parameter | Description |
|-----------|--|
| | <ul style="list-style-type: none"> ■ <code>threadCount</code> specifies the number of threads to create per HAProxy process. Creating more threads allows each Ingress Controller pod to handle more connections, at the cost of more system resources being used. HAProxy supports up to 64 threads. If this field is empty, the Ingress Controller uses the default value of 4 threads. The default value can change in future releases. Setting this field is not recommended because increasing the number of HAProxy threads allows Ingress Controller pods to use more CPU time under load, and prevent other pods from receiving the CPU resources they need to perform. Reducing the number of threads can cause the Ingress Controller to perform poorly. ■ <code>tlsInspectDelay</code> specifies how long the router can hold data to find a matching route. Setting this value too short can cause the router to fall back to the default certificate for edge-terminated, reencrypted, or passthrough routes, even when using a better matched certificate. The default inspect delay is 5s . ■ <code>tunnelTimeout</code> specifies how long a tunnel connection, including websockets, remains open while the tunnel is idle. The default timeout is 1h . ■ <code>maxConnections</code> specifies the maximum number of simultaneous connections that can be established per HAProxy process. Increasing this value allows each ingress controller pod to handle more connections at the cost of additional system resources. Permitted values are 0 , -1 , any value within the range 2000 and 2000000 , or the field can be left empty. <ul style="list-style-type: none"> ■ If this field is left empty or has the value 0 , the Ingress Controller will use the default value of 50000 . This value is subject to change in future releases. ■ If the field has the value of -1 , then HAProxy will dynamically compute a maximum value based on the available ulimits in the running container. This process results in a large computed value that will incur significant memory usage compared to the current default value of 50000 . ■ If the field has a value that is greater than the current operating system limit, the HAProxy process will not start. |

| Parameter | Description |
|-------------------------------|---|
| | <ul style="list-style-type: none"> ■ If you choose a discrete value and the router pod is migrated to a new node, it is possible the new node does not have an identical <code>ulimit</code> configured. In such cases, the pod fails to start. ■ If you have nodes with different <code>ulimits</code> configured, and you choose a discrete value, it is recommended to use the value of <code>-1</code> for this field so that the maximum number of connections is calculated at runtime. |
| <code>logEmptyRequests</code> | <p><code>logEmptyRequests</code> specifies connections for which no request is received and logged. These empty requests come from load balancer health probes or web browser speculative connections (<code>preconnect</code>) and logging these requests can be undesirable. However, these requests can be caused by network errors, in which case logging empty requests can be useful for diagnosing the errors. These requests can be caused by port scans, and logging empty requests can aid in detecting intrusion attempts. Allowed values for this field are <code>Log</code> and <code>Ignore</code>. The default value is <code>Log</code>.</p> <p>The <code>LoggingPolicy</code> type accepts either one of two values:</p> <ul style="list-style-type: none"> ■ <code>Log</code> : Setting this value to <code>Log</code> indicates that an event should be logged. ■ <code>Ignore</code> : Setting this value to <code>Ignore</code> sets the <code>dontlognull</code> option in the HAproxy configuration. |

| Parameter | Description |
|-------------------------|--|
| HTTPEmptyRequestsPolicy | <p>HTTPEmptyRequestsPolicy describes how HTTP connections are handled if the connection times out before a request is received. Allowed values for this field are <code>Respond</code> and <code>Ignore</code>. The default value is <code>Respond</code>.</p> <p>The <code>HTTPEmptyRequestsPolicy</code> type accepts either one of two values:</p> <ul style="list-style-type: none"> ▪ Respond : If the field is set to <code>Respond</code>, the Ingress Controller sends an HTTP <code>400</code> or <code>408</code> response, logs the connection if access logging is enabled, and counts the connection in the appropriate metrics. ▪ Ignore : Setting this option to <code>Ignore</code> adds the <code>http-ignore-probes</code> parameter in the HAproxy configuration. If the field is set to <code>Ignore</code>, the Ingress Controller closes the connection without sending a response, then logs the connection, or incrementing metrics. <p>These connections come from load balancer health probes or web browser speculative connections (preconnect) and can be safely ignored. However, these requests can be caused by network errors, so setting this field to <code>Ignore</code> can impede detection and diagnosis of problems. These requests can be caused by port scans, in which case logging empty requests can aid in detecting intrusion attempts.</p> |



All parameters are optional.

Ingress Controller TLS security profiles



TLS security profiles provide a way for servers to regulate which ciphers a connecting client can use when connecting to the server.

Understanding TLS security profiles

You can use a TLS (Transport Layer Security) security profile to define which TLS ciphers are required by various OpenShift Container Platform components. The OpenShift Container Platform TLS security profiles are based on [Mozilla recommended configurations](#).

You can specify one of the following TLS security profiles for each component:

Table 1. TLS security profiles

| Profile | Description |
|--------------|---|
| Old | <p>This profile is intended for use with legacy clients or libraries. The profile is based on the Old backward compatibility recommended configuration.</p> <p>The <code>Old</code> profile requires a minimum TLS version of 1.0.</p> <div> For the Ingress Controller, the minimum TLS version is converted from 1.0 to 1.1.</div> |
| Intermediate | <p>This profile is the recommended configuration for the majority of clients. It is the default TLS security profile for the Ingress Controller, kubelet, and control plane. The profile is based on the Intermediate compatibility recommended configuration.</p> <p>The <code>Intermediate</code> profile requires a minimum TLS version of 1.2.</p> |
| Modern | <p>This profile is intended for use with modern clients that have no need for backwards compatibility. This profile is based on the Modern compatibility recommended configuration.</p> <p>The <code>Modern</code> profile requires a minimum TLS version of 1.3.</p> |
| Custom | <p>This profile allows you to define the TLS version and ciphers to use.</p> <div> Use caution when using a <code>Custom</code> profile, because invalid configurations can cause problems.</div> |



When using one of the predefined profile types, the effective profile configuration is subject to change between releases. For example, given a specification to use the `Intermediate` profile deployed on release `X.Y.Z`, an upgrade to release `X.Y.Z+1` might cause a new profile configuration to be applied, resulting in a rollout.

Configuring the TLS security profile for the Ingress Controller

To configure a TLS security profile for an Ingress Controller, edit the `IngressController` custom resource (CR) to specify a predefined or custom TLS security profile. If a TLS security profile is not configured, the default value is based on the TLS security profile set for the API server.

Sample `IngressController` CR that configures the `Old` TLS security profile

```
apiVersion: operator.openshift.io/v1
kind: IngressController
...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
...
```

The TLS security profile defines the minimum TLS version and the TLS ciphers for TLS connections for Ingress Controllers.

You can see the ciphers and the minimum TLS version of the configured TLS security profile in the `IngressController` custom resource (CR) under `Status.Tls Profile` and the configured TLS security profile under `Spec.Tls Security Profile`. For the `Custom` TLS security profile, the specific ciphers and minimum TLS version are listed under both parameters.



The HAProxy Ingress Controller image supports TLS 1.3 and the Modern profile.

The Ingress Operator also converts the TLS 1.0 of an `Old` or `Custom` profile to 1.1.

Prerequisites

- You have access to the cluster as a user with the `cluster-admin` role.

Procedure

- Edit the `IngressController` CR in the `openshift-ingress-operator` project to configure the TLS security profile:

```
$ oc edit IngressController default -n openshift-ingress-operator
```

- Add the `spec.tlsSecurityProfile` field:

Sample IngressController CR for a Custom profile

```
apiVersion: operator.openshift.io/v1
kind: IngressController
...
spec:
  tlsSecurityProfile:
    type: Custom (1)
    custom: (2)
      ciphers: (3)
        - ECDHE-ECDSA-CHACHA20-POLY1305
        - ECDHE-RSA-CHACHA20-POLY1305
        - ECDHE-RSA-AES128-GCM-SHA256
        - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
  ...
```

- 1 Specify the TLS security profile type (Old, Intermediate, or Custom). The default is Intermediate.

Specify the appropriate field for the selected type:

- `old: {}`
- 2
 - `intermediate: {}`
 - `custom:`

- 3 For the `custom` type, specify a list of TLS ciphers and minimum accepted TLS version.

- Save the file to apply the changes.

Verification

- Verify that the profile is set in the IngressController CR:

```
$ oc describe IngressController default -n openshift-ingress-operator
```

Example output

```

Name:          default
Namespace:     openshift-ingress-operator
Labels:        <none>
Annotations:   <none>
API Version:   operator.openshift.io/v1
Kind:          IngressController
...
Spec:
...
  Tls Security Profile:
    Custom:
      Ciphers:
        ECDHE-ECDSA-CHACHA20-POLY1305
        ECDHE-RSA-CHACHA20-POLY1305
        ECDHE-RSA-AES128-GCM-SHA256
        ECDHE-ECDSA-AES128-GCM-SHA256
      Min TLS Version:  VersionTLS11
      Type:              Custom
...

```

Configuring mutual TLS authentication

You can configure the Ingress Controller to enable mutual TLS (mTLS) authentication by setting a `spec.clientTLS` value. The `clientTLS` value configures the Ingress Controller to verify client certificates. This configuration includes setting a `clientCA` value, which is a reference to a config map. The config map contains the PEM-encoded CA certificate bundle that is used to verify a client's certificate. Optionally, you can also configure a list of certificate subject filters.

If the `clientCA` value specifies an X509v3 certificate revocation list (CRL) distribution point, the Ingress Operator downloads and manages a CRL config map based on the HTTP URI X509v3 CRL Distribution Point specified in each provided certificate. The Ingress Controller uses this config map during mTLS/TLS negotiation. Requests that do not provide valid certificates are rejected.

Prerequisites

- You have access to the cluster as a user with the `cluster-admin` role.
- You have a PEM-encoded CA certificate bundle.

- If your CA bundle references a CRL distribution point, you must have also included the end-entity or leaf certificate to the client CA bundle. This certificate must have included an HTTP URI under **CRL Distribution Points**, as described in RFC 5280. For example:

```
Issuer: C=US, O=Example Inc, CN=Example Global G2 TLS RSA
SHA256 2020 CA1
      Subject: SOME SIGNED CERT                      X509v3 CRL
Distribution Points:
      Full Name:
      URI:http://crl.example.com/example.crl
```

Procedure

- In the `openshift-config` namespace, create a config map from your CA bundle:

```
$ oc create configmap \
  router-ca-certs-default \
  --from-file=ca-bundle.pem=client-ca.crt \ (1)
  -n openshift-config
```

1 The config map data key must be `ca-bundle.pem`, and the data value must be a CA certificate in PEM format.

- Edit the `IngressController` resource in the `openshift-ingress-operator` project:

```
$ oc edit IngressController default -n openshift-ingress-operator
```

- Add the `spec.clientTLS` field and subfields to configure mutual TLS:

Sample `IngressController` CR for a `clientTLS` profile that specifies filtering patterns

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  clientTLS:
    clientCertificatePolicy: Required
    clientCA:
      name: router-ca-certs-default
      allowedSubjectPatterns:
        - "^/CN=example.com/ST=NC/C=US/O=Security/OU=OpenShift$"
```

View the default Ingress Controller

The Ingress Operator is a core feature of OpenShift Container Platform and is enabled out of the box.

Every new OpenShift Container Platform installation has an `ingresscontroller` named `default`. It can be supplemented with additional Ingress Controllers. If the default `ingresscontroller` is deleted, the Ingress Operator will automatically recreate it within a minute.

Procedure

- View the default Ingress Controller:

```
$ oc describe --namespace=openshift-ingress-operator
ingresscontroller/default
```

View Ingress Operator status

You can view and inspect the status of your Ingress Operator.

Procedure

- View your Ingress Operator status:

```
$ oc describe clusteroperators/ingress
```


View Ingress Controller logs

You can view your Ingress Controller logs.

Procedure

- View your Ingress Controller logs:

```
$ oc logs --namespace=openshift-ingress-operator  
deployments/ingress-operator -c <container_name>
```

View Ingress Controller status

You can view the status of a particular Ingress Controller.

Procedure

- View the status of an Ingress Controller:

```
$ oc describe --namespace=openshift-ingress-operator  
ingresscontroller/<name>
```

Configuring the Ingress Controller

Setting a custom default certificate

As an administrator, you can configure an Ingress Controller to use a custom certificate by creating a Secret resource and editing the `IngressController` custom resource (CR).

Prerequisites

- You must have a certificate/key pair in PEM-encoded files, where the certificate is signed by a trusted certificate authority or by a private trusted certificate authority that you configured in a custom PKI.
- Your certificate meets the following requirements:
 - The certificate is valid for the ingress domain.
 - The certificate uses the `subjectAltName` extension to specify a wildcard domain, such as `*.apps.ocp4.example.com`.
- You must have an `IngressController` CR. You may use the default one:

```
$ oc --namespace openshift-ingress-operator get
ingresscontrollers
```

Example output

```
NAME      AGE
default   10m
```



If you have intermediate certificates, they must be included in the `tls.crt` file of the secret containing a custom default certificate. Order matters when specifying a certificate; list your intermediate certificate(s) after any server certificate(s).

Procedure

The following assumes that the custom certificate and key pair are in the `tls.crt` and `tls.key` files in the current working directory. Substitute the actual path names for `tls.crt` and `tls.key`. You also may substitute another name for `custom-certs-default` when creating the Secret resource and referencing it in the IngressController CR.



This action will cause the Ingress Controller to be redeployed, using a rolling deployment strategy.

- Create a Secret resource containing the custom certificate in the `openshift-ingress` namespace using the `tls.crt` and `tls.key` files.

```
$ oc --namespace openshift-ingress create secret tls custom-
certs-default --cert=tls.crt --key=tls.key
```

- Update the IngressController CR to reference the new certificate secret:

```
$ oc patch --type=merge --namespace openshift-ingress-operator
ingresscontrollers/default \
  --patch '{"spec":{"defaultCertificate":{"name":"custom-certs-
default"}}}'
```

- Verify the update was effective:

```
$ echo Q |\n  openssl s_client -connect console-openshift-console.apps.\n<domain>:443 -showcerts 2>/dev/null |\n  openssl x509 -noout -subject -issuer -enddate
```

where:

<domain>

Specifies the base domain name for your cluster.

Example output

```
subject=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN =\n*.apps.example.com\nissuer=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN =\nexample.com\nnotAfter=May 10 08:32:45 2022 GM
```



You can alternatively apply the following YAML to set a custom default certificate:

```
apiVersion: operator.openshift.io/v1\nkind: IngressController\nmetadata:\n  name: default\n  namespace: openshift-ingress-operator\nspec:\n  defaultCertificate:\n    name: custom-certs-default
```

The certificate secret name should match the value used to update the CR. Once the IngressController CR has been modified, the Ingress Operator updates the Ingress Controller's deployment to use the custom certificate.

Removing a custom default certificate

As an administrator, you can remove a custom certificate that you configured an Ingress Controller to use.

Prerequisites

- You have access to the cluster as a user with the `cluster-admin` role.
- You have installed the OpenShift CLI (`oc`).
- You previously configured a custom default certificate for the Ingress Controller.

Procedure

- To remove the custom certificate and restore the certificate that ships with OpenShift Container Platform, enter the following command:

```
$ oc patch -n openshift-ingress-operator
ingresscontrollers/default \
  --type json -p $'- op: remove\n path:
/spec/defaultCertificate'
```

There can be a delay while the cluster reconciles the new certificate configuration.

Verification

- To confirm that the original cluster certificate is restored, enter the following command:

```
$ echo Q | \
  openssl s_client -connect console-openshift-console.apps.
<domain>:443 -showcerts 2>/dev/null | \
  openssl x509 -noout -subject -issuer -enddate
```

where:

<domain>

Specifies the base domain name for your cluster.

Example output

```
subject=CN = *.apps.<domain>
issuer=CN = ingress-operator@1620633373
notAfter=May 10 10:44:36 2023 GMT
```

Autoscaling an Ingress Controller

Automatically scale an Ingress Controller to dynamically meet routing performance or availability requirements such as the requirement to increase throughput. The following procedure provides an example for scaling up the default IngressController.

Prerequisites

- You have the OpenShift CLI (oc) installed.
- You have access to an OpenShift Container Platform cluster as a user with the cluster-admin role.
- You have the Custom Metrics Autoscaler Operator installed.
- You are in the openshift-ingress-operator project namespace.

Procedure

- Create a service account to authenticate with Thanos by running the following command:

```
$ oc create serviceaccount thanos && oc describe serviceaccount thanos
```

Example output

```
Name:                thanos
Namespace:           openshift-ingress-operator
Labels:              <none>
Annotations:         <none>
Image pull secrets:  thanos-dockercfg-b4l9s
Mountable secrets:   thanos-dockercfg-b4l9s
Tokens:              thanos-token-c422q
Events:              <none>
```

- Define a TriggerAuthentication object within the openshift-ingress-operator namespace using the service account's token.
 - Define the variable secret that contains the secret by running the following command:

```
$ secret=$(oc get secret | grep thanos-token | head -n 1 |  
awk '{ print $1 }')
```

- Create the `TriggerAuthentication` object and pass the value of the `secret` variable to the `TOKEN` parameter:

```
$ oc process TOKEN="$secret" -f - <<EOF | oc apply -f -  
apiVersion: template.openshift.io/v1  
kind: Template  
parameters:  
- name: TOKEN  
objects:  
- apiVersion: keda.sh/v1alpha1  
  kind: TriggerAuthentication  
  metadata:  
    name: keda-trigger-auth-prometheus  
  spec:  
    secretTargetRef:  
    - parameter: bearerToken  
      name: \${TOKEN}  
      key: token  
    - parameter: ca  
      name: \${TOKEN}  
      key: ca.crt  
EOF
```

- Create and apply a role for reading metrics from Thanos:
 - Create a new role, `thanos-metrics-reader.yaml`, that reads metrics from pods and nodes:

thanos-metrics-reader.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: thanos-metrics-reader
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - nodes
  verbs:
  - get
- apiGroups:
  - metrics.k8s.io
  resources:
  - pods
  - nodes
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
```

- Apply the new role by running the following command:

```
$ oc apply -f thanos-metrics-reader.yaml
```

- Add the new role to the service account by entering the following commands:

```
$ oc adm policy add-role-to-user thanos-metrics-reader -z
thanos --role-namespace=openshift-ingress-operator
```

```
$ oc adm policy -n openshift-ingress-operator add-cluster-role-
to-user cluster-monitoring-view -z thanos
```



The argument `add-cluster-role-to-user` is only required if you use cross-namespace queries. The following step uses a query from the `kube-metrics` namespace which requires this argument.

- Create a new `ScaledObject` YAML file, `ingress-autoscaler.yaml`, that targets the default Ingress Controller deployment:

Example `ScaledObject` definition

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: ingress-scaler
spec:
  scaleTargetRef: (1)
    apiVersion: operator.openshift.io/v1
    kind: IngressController
    name: default
    envSourceContainerName: ingress-operator
  minReplicaCount: 1
  maxReplicaCount: 20 (2)
  cooldownPeriod: 1
  pollingInterval: 1
  triggers:
  - type: prometheus
    metricType: AverageValue
    metadata:
      serverAddress: https://thanos-querier.openshift-
monitoring.svc.cluster.local:9091 (3)
      namespace: openshift-ingress-operator (4)
      metricName: 'kube-node-role'
      threshold: '1'
      query: 'sum(kube_node_role{role="worker",service="kube-
state-metrics"})' (5)
      authModes: "bearer"
    authenticationRef:
      name: keda-trigger-auth-prometheus
```

- 1 The custom resource that you are targeting. In this case, the Ingress Controller.

- 2 Optional: The maximum number of replicas. If you omit this field, the default maximum is set to 100 replicas.
- 3 The Thanos service endpoint in the `openshift-monitoring` namespace.
- 4 The Ingress Operator namespace.
- 5 This expression evaluates to however many worker nodes are present in the deployed cluster.



If you are using cross-namespace queries, you must target port 9091 and not port 9092 in the `serverAddress` field. You also must have elevated privileges to read metrics from this port.

- Apply the custom resource definition by running the following command:

```
$ oc apply -f ingress-autoscaler.yaml
```

Verification

- Verify that the default Ingress Controller is scaled out to match the value returned by the `kube-state-metrics` query by running the following commands:
 - Use the `grep` command to search the Ingress Controller YAML file for replicas:

```
$ oc get ingresscontroller/default -o yaml | grep replicas:
```

Example output

```
replicas: 3
```

- Get the pods in the `openshift-ingress` project:

```
$ oc get pods -n openshift-ingress
```

Example output

| NAME | READY | STATUS | RESTARTS |
|---|-------|---------|----------|
| AGE | | | |
| router-default-7b5df44ff-19pmm 17h | 2/2 | Running | 0 |
| router-default-7b5df44ff-s5sl5 3d22h | 2/2 | Running | 0 |
| router-default-7b5df44ff-wwsth 66s | 2/2 | Running | 0 |

Additional resources

- [Enabling monitoring for user-defined projects](#)
- [Installing the custom metrics autoscaler](#)
- [Understanding custom metrics autoscaler trigger authentications](#)
- [Configuring the custom metrics autoscaler to use OpenShift Container Platform monitoring](#)
- [Understanding how to add custom metrics autoscalers](#)

Scaling an Ingress Controller

Manually scale an Ingress Controller to meeting routing performance or availability requirements such as the requirement to increase throughput. `oc` commands are used to scale the `IngressController` resource. The following procedure provides an example for scaling up the default `IngressController`.



Scaling is not an immediate action, as it takes time to create the desired number of replicas.

Procedure

- View the current number of available replicas for the default `IngressController`:

```
$ oc get -n openshift-ingress-operator
ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
```

Example output

- Scale the default `IngressController` to the desired number of replicas using the `oc patch` command. The following example scales the default `IngressController` to 3 replicas:

```
$ oc patch -n openshift-ingress-operator
ingresscontroller/default --patch '{"spec":{"replicas": 3}}' --
type=merge
```

Example output

```
ingresscontroller.operator.openshift.io/default patched
```

- Verify that the default `IngressController` scaled to the number of replicas that you specified:

```
$ oc get -n openshift-ingress-operator
ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
```

Example output



You can alternatively apply the following YAML to scale an Ingress Controller to three replicas:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 3 (1)
```

1 If you need a different amount of replicas, change the `replicas` value.

Configuring Ingress access logging

You can configure the Ingress Controller to enable access logs. If you have clusters that do not receive much traffic, then you can log to a sidecar. If you have high traffic clusters, to avoid exceeding the capacity of the logging stack or to integrate with a logging infrastructure outside of OpenShift Container Platform, you can forward logs to a custom syslog endpoint. You can also specify the format for access logs.

Container logging is useful to enable access logs on low-traffic clusters when there is no existing Syslog logging infrastructure, or for short-term use while diagnosing problems with the Ingress Controller.

Syslog is needed for high-traffic clusters where access logs could exceed the OpenShift Logging stack's capacity, or for environments where any logging solution needs to integrate with an existing Syslog logging infrastructure. The Syslog use-cases can overlap.

Prerequisites

- Log in as a user with `cluster-admin` privileges.

Procedure

Configure Ingress access logging to a sidecar.

- To configure Ingress access logging, you must specify a destination using `spec.logging.access.destination`. To specify logging to a sidecar container, you must specify `Container` `spec.logging.access.destination.type`. The following example is an Ingress Controller definition that logs to a `Container` destination:

```

apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Container

```

- When you configure the Ingress Controller to log to a sidecar, the operator creates a container named `logs` inside the Ingress Controller Pod:

```

$ oc -n openshift-ingress logs deployment.apps/router-default -c logs

```

Example output

```

2020-05-11T19:11:50.135710+00:00 router-default-57dfc6cd95-bpmk6 router-default-57dfc6cd95-bpmk6 haproxy[108]:
174.19.21.82:39654 [11/May/2020:19:11:50.133] public
be_http:hello-openshift:hello-openshift/pod:hello-openshift:hello-openshift:10.128.2.12:8080 0/0/1/0/1 200 142 -
- --NI 1/1/0/0/0 0/0 "GET / HTTP/1.1"

```

Configure Ingress access logging to a Syslog endpoint.

- To configure Ingress access logging, you must specify a destination using `spec.logging.access.destination`. To specify logging to a Syslog endpoint destination, you must specify `Syslog` for `spec.logging.access.destination.type`. If the destination type is `Syslog`, you must also specify a destination endpoint using `spec.logging.access.destination.syslog.endpoint` and you can specify a facility using `spec.logging.access.destination.syslog.facility`. The following example is an Ingress Controller definition that logs to a `Syslog` destination:

```

apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514

```



The `syslog` destination port must be UDP.

Configure Ingress access logging with a specific log format.

- You can specify `spec.logging.access.httpLogFormat` to customize the log format. The following example is an Ingress Controller definition that logs to a `syslog` endpoint with IP address 1.2.3.4 and port 10514:

```

apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
          httpLogFormat: '%ci:%cp [%t] %ft %b/%s %B %bq %HM %HU
%HV'

```

Disable Ingress access logging.

- To disable Ingress access logging, leave `spec.logging` or `spec.logging.access` empty:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access: null
```

Allow the Ingress Controller to modify the HAProxy log length when using a sidecar.

- Use `spec.logging.access.destination.syslog.maxLength` if you are using `spec.logging.access.destination.type: Syslog`.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          maxLength: 4096
          port: 10514
```

- Use `spec.logging.access.destination.container.maxLength` if you are using `spec.logging.access.destination.type: Container`.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Container
        container:
          maxLength: 8192
```

Setting Ingress Controller thread count

A cluster administrator can set the thread count to increase the amount of incoming connections a cluster can handle. You can patch an existing Ingress Controller to increase the amount of threads.

Prerequisites

- The following assumes that you already created an Ingress Controller.

Procedure

- Update the Ingress Controller to increase the number of threads:

```
$ oc -n openshift-ingress-operator patch
ingresscontroller/default --type=merge -p '{"spec":
{"tuningOptions": {"threadCount": 8}}}'
```



If you have a node that is capable of running large amounts of resources, you can configure

`spec.nodePlacement.nodeSelector` with labels that match the capacity of the intended node, and configure

`spec.tuningOptions.threadCount` to an appropriately high value.

Configuring an Ingress Controller to use an internal load balancer

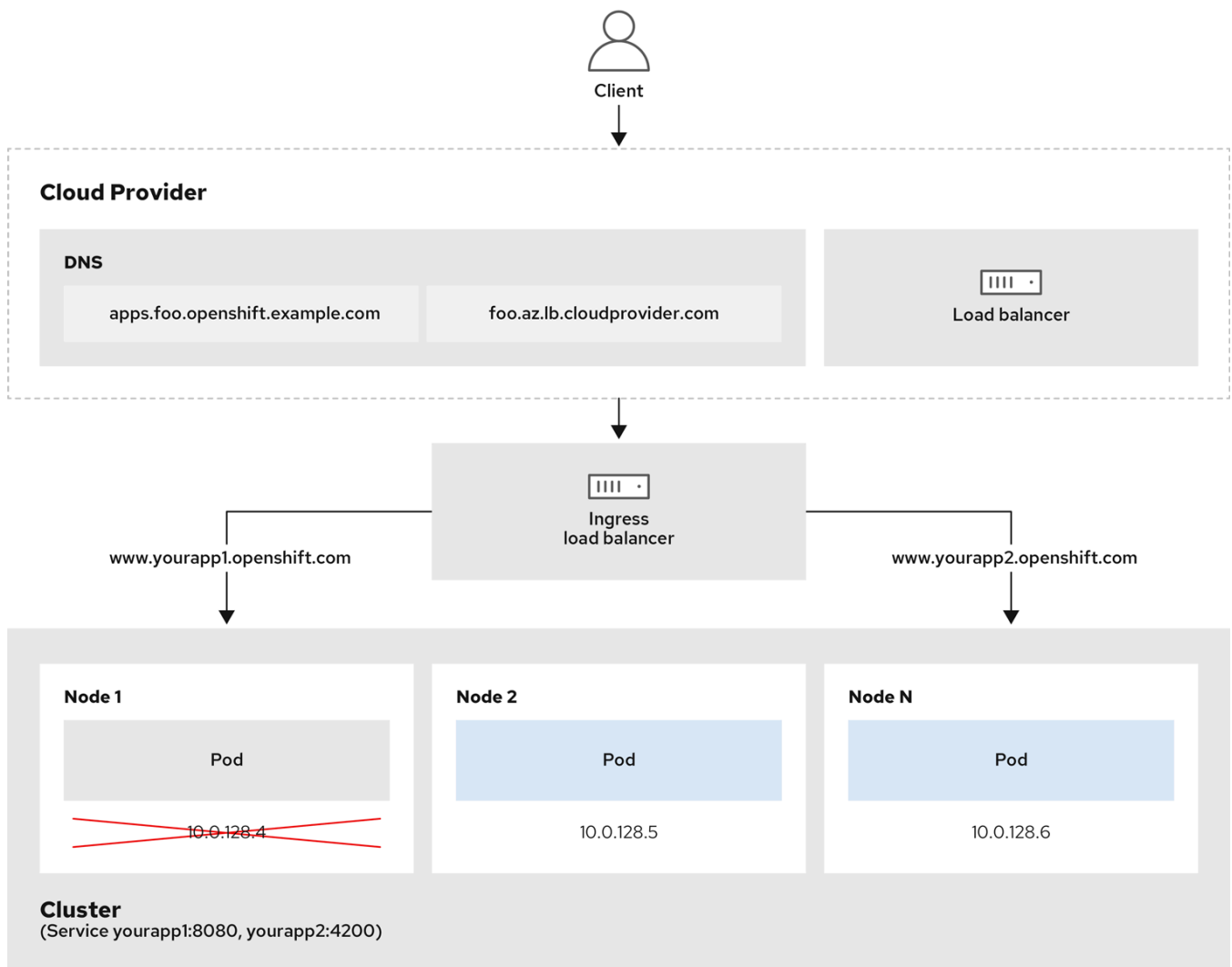
When creating an Ingress Controller on cloud platforms, the Ingress Controller is published by a public cloud load balancer by default. As an administrator, you can create an Ingress Controller that uses an internal cloud load balancer.



If your cloud provider is Microsoft Azure, you must have at least one public load balancer that points to your nodes. If you do not, all of your nodes will lose egress connectivity to the internet.



If you want to change the `scope` for an `IngressController`, you can change the `.spec.endpointPublishingStrategy.loadBalancer.scope` parameter after the custom resource (CR) is created.



202_OpenShift_0222

Figure 1. Diagram of LoadBalancer

The preceding graphic shows the following concepts pertaining to OpenShift Container Platform Ingress LoadBalancerService endpoint publishing strategy:

- You can load balance externally, using the cloud provider load balancer, or internally, using the OpenShift Ingress Controller Load Balancer.
- You can use the single IP address of the load balancer and more familiar ports, such as 8080 and 4200 as shown on the cluster depicted in the graphic.
- Traffic from the external load balancer is directed at the pods, and managed by the load balancer, as depicted in the instance of a down node. See the [Kubernetes Services documentation](#) for implementation details.

Prerequisites

- Install the OpenShift CLI (`oc`).

- Log in as a user with `cluster-admin` privileges.

Procedure

- Create an `IngressController` custom resource (CR) in a file named `<name>-ingress-controller.yaml`, such as in the following example:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> (1)
spec:
  domain: <domain> (2)
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal (3)
```

- 1 Replace `<name>` with a name for the `IngressController` object.
- 2 Specify the `domain` for the application published by the controller.
- 3 Specify a value of `Internal` to use an internal load balancer.

- Create the Ingress Controller defined in the previous step by running the following command:

```
$ oc create -f <name>-ingress-controller.yaml (1)
```

- 1 Replace `<name>` with the name of the `IngressController` object.

- Optional: Confirm that the Ingress Controller was created by running the following command:

```
$ oc --all-namespaces=true get ingresscontrollers
```

Configuring global access for an Ingress Controller on GCP

An Ingress Controller created on GCP with an internal load balancer generates an internal IP address for the service. A cluster administrator can specify the global access option, which enables clients in any region within the same VPC network and compute region as the load balancer, to reach the workloads running on your cluster.

For more information, see the GCP documentation for [global access](#).

Prerequisites

- You deployed an OpenShift Container Platform cluster on GCP infrastructure.
- You configured an Ingress Controller to use an internal load balancer.
- You installed the OpenShift CLI (`oc`).

Procedure

- Configure the Ingress Controller resource to allow global access.



You can also create an Ingress Controller and specify the global access option.

- Configure the Ingress Controller resource:

```
$ oc -n openshift-ingress-operator edit  
ingresscontroller/default
```

- Edit the YAML file:

Sample `clientAccess` configuration to Global

```
spec:  
  endpointPublishingStrategy:  
    loadBalancer:  
      providerParameters:  
        gcp:  
          clientAccess: Global (1)  
          type: GCP  
          scope: Internal  
        type: LoadBalancerService
```

1 Set `gcp.clientAccess` to `Global`.

- Save the file to apply the changes.
- Run the following command to verify that the service allows global access:

```
$ oc -n openshift-ingress edit svc/router-default -o yaml
```

The output shows that global access is enabled for GCP with the annotation, `networking.gke.io/internal-load-balancer-allow-global-access`.

Setting the Ingress Controller health check interval

A cluster administrator can set the health check interval to define how long the router waits between two consecutive health checks. This value is applied globally as a default for all routes. The default value is 5 seconds.

Prerequisites

- The following assumes that you already created an Ingress Controller.

Procedure

- Update the Ingress Controller to change the interval between back end health checks:

```
$ oc -n openshift-ingress-operator patch
ingresscontroller/default --type=merge -p '{"spec":
{"tuningOptions": {"healthCheckInterval": "8s"}}}'
```



To override the `healthCheckInterval` for a single route, use the route annotation

`router.openshift.io/haproxy.health.check.interval`

Configuring the default Ingress Controller for your cluster to be internal

You can configure the default Ingress Controller for your cluster to be internal by deleting and recreating it.



If your cloud provider is Microsoft Azure, you must have at least one public load balancer that points to your nodes. If you do not, all of your nodes will lose egress connectivity to the internet.



If you want to change the `scope` for an `IngressController`, you can change the `.spec.endpointPublishingStrategy.loadBalancer.scope` parameter after the custom resource (CR) is created.

Prerequisites

- Install the OpenShift CLI (`oc`).
- Log in as a user with `cluster-admin` privileges.

Procedure

- Configure the default Ingress Controller for your cluster to be internal by deleting and recreating it.

```
$ oc replace --force --wait --filename - <<EOF
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: default
spec:
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal
EOF
```

Configuring the route admission policy

Administrators and application developers can run applications in multiple namespaces with the same domain name. This is for organizations where multiple teams develop microservices that are exposed on the same hostname.



Allowing claims across namespaces should only be enabled for clusters with trust between namespaces, otherwise a malicious user could take over a hostname. For this reason, the default admission policy disallows hostname claims across namespaces.

Prerequisites

- Cluster administrator privileges.

Procedure

- Edit the `.spec.routeAdmission` field of the `ingresscontroller` resource variable using the following command:

```
$ oc -n openshift-ingress-operator patch
ingresscontroller/default --patch '{"spec":{"routeAdmission":
{"namespaceOwnership":"InterNamespaceAllowed"}}}' --type=merge
```

Sample Ingress Controller configuration

```
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
  ...
```



You can alternatively apply the following YAML to configure the route admission policy:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
```

Using wildcard routes

The HAProxy Ingress Controller has support for wildcard routes. The Ingress Operator uses `wildcardPolicy` to configure the `ROUTER_ALLOW_WILDCARD_ROUTES` environment variable of the Ingress Controller.

The default behavior of the Ingress Controller is to admit routes with a wildcard policy of `None`, which is backwards compatible with existing `IngressController` resources.

Procedure

- Configure the wildcard policy.
 - Use the following command to edit the `IngressController` resource:

```
$ oc edit IngressController
```

- Under `spec`, set the `wildcardPolicy` field to `WildcardsDisallowed` or `WildcardsAllowed`:

```
spec:
  routeAdmission:
    wildcardPolicy: WildcardsDisallowed # or
    WildcardsAllowed
```

HTTP header configuration

OpenShift Container Platform provides different methods for working with HTTP headers. When setting or deleting headers, you can use specific fields in the Ingress Controller or an individual route to modify request and response headers. You can also set certain headers by using route annotations. The various ways of configuring headers can present challenges when working together.



You can only set or delete headers within an `IngressController` or `Route CR`, you cannot append them. If an HTTP header is set with a value, that value must be complete and not require appending in the future. In situations where it makes sense to append a header, such as the `X-Forwarded-For` header, use the `spec.httpHeaders.forwardedHeaderPolicy` field, instead of `spec.httpHeaders.actions`.

Order of precedence

When the same HTTP header is modified both in the Ingress Controller and in a route, HAProxy prioritizes the actions in certain ways depending on whether it is a request or response header.

- For HTTP response headers, actions specified in the Ingress Controller are executed after the actions specified in a route. This means that the actions specified in the Ingress Controller take precedence.
- For HTTP request headers, actions specified in a route are executed after the actions specified in the Ingress Controller. This means that the actions specified in the route take precedence.

For example, a cluster administrator sets the X-Frame-Options response header with the value `DENY` in the Ingress Controller using the following configuration:

Example IngressController spec

```
apiVersion: operator.openshift.io/v1
kind: IngressController
# ...
spec:
  httpHeaders:
    actions:
      response:
        - name: X-Frame-Options
          action:
            type: Set
            set:
              value: DENY
```

A route owner sets the same response header that the cluster administrator set in the Ingress Controller, but with the value `SAMEORIGIN` using the following configuration:

Example Route spec

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  httpHeaders:
    actions:
      response:
        - name: X-Frame-Options
          action:
            type: Set
            set:
              value: SAMEORIGIN
```

When both the `IngressController` spec and `Route` spec are configuring the X-Frame-Options header, then the value set for this header at the global level in the Ingress Controller will take precedence, even if a specific route allows frames.

This prioritization occurs because the `haproxy.config` file uses the following logic, where the Ingress Controller is considered the front end and individual routes are considered the back end. The header value `DENY` applied to the front end configurations overrides the same header with the value `SAMEORIGIN` that is set in the back end:

```
frontend public
  http-response set-header X-Frame-Options 'DENY'

frontend fe_sni
  http-response set-header X-Frame-Options 'DENY'

frontend fe_no_sni
  http-response set-header X-Frame-Options 'DENY'

backend be_secure:openshift-monitoring:alertmanager-main
  http-response set-header X-Frame-Options 'SAMEORIGIN'
```

Additionally, any actions defined in either the Ingress Controller or a route override values set using route annotations.

Special case headers

The following headers are either prevented entirely from being set or deleted, or allowed under specific circumstances:

Table 2. Special case header configuration options

| Header name | Configurable using IngressController spec | Configurable using Route spec | Reason for disallowment | Configurable using another method |
|-------------|---|-------------------------------|-------------------------|-----------------------------------|
| | | | | |

| Header name | Configurable using IngressController spec | Configurable using Route spec | Reason for disallowment | Configurable using another method |
|-------------|---|-------------------------------|--|-----------------------------------|
| proxy | No | No | The proxy HTTP request header can be used to exploit vulnerable CGI applications by injecting the header value into the HTTP_PROXY environment variable. The proxy HTTP request header is also non-standard and prone to error during configuration. | No |
| host | No | Yes | When the host HTTP request header is set using the IngressController CR, HAProxy can fail when looking up the correct route. | No |

| Header name | Configurable using IngressController spec | Configurable using Route spec | Reason for disallowment | Configurable using another method |
|---------------------------|---|-------------------------------|--|--|
| strict-transport-security | No | No | The strict-transport-security HTTP response header is already handled using route annotations and does not need a separate implementation. | Yes: the <code>haproxy.router.openshift.io/htsts_header</code> route annotation |
| cookie and set-cookie | No | No | The cookies that HAProxy sets are used for session tracking to map client connections to particular back-end servers. Allowing these headers to be set could interfere with HAProxy's session affinity and restrict HAProxy's ownership of a cookie. | Yes: <ul style="list-style-type: none"> the <code>haproxy.router.openshift.io/disable_cookie</code> route annotation the <code>haproxy.router.openshift.io/cookie_name</code> route annotation |

Setting or deleting HTTP request and response headers in an Ingress Controller

You can set or delete certain HTTP request and response headers for compliance purposes or other reasons. You can set or delete these headers either for all routes served by an Ingress Controller or for specific routes.

For example, you might want to migrate an application running on your cluster to use mutual TLS, which requires that your application checks for an X-Forwarded-Client-Cert request header, but the OpenShift Container Platform default Ingress Controller provides an X-SSL-Client-Der request header.

The following procedure modifies the Ingress Controller to set the X-Forwarded-Client-Cert request header, and delete the X-SSL-Client-Der request header.

Prerequisites

- You have installed the OpenShift CLI (`oc`).
- You have access to an OpenShift Container Platform cluster as a user with the `cluster-admin` role.

Procedure

- Edit the Ingress Controller resource:

```
$ oc -n openshift-ingress-operator edit
ingresscontroller/default
```

- Replace the X-SSL-Client-Der HTTP request header with the X-Forwarded-Client-Cert HTTP request header:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    actions: (1)
      request: (2)
      - name: X-Forwarded-Client-Cert (3)
        action:
          type: Set (4)
          set:
            value: "%{+Q}[ssl_c_der,base64]" (5)
      - name: X-SSL-Client-Der
        action:
          type: Delete
```

- 1 The list of actions you want to perform on the HTTP headers.

- 2 The type of header you want to change. In this case, a request header.
- 3 The name of the header you want to change. For a list of available headers you can set or delete, see *HTTP header configuration*.
- 4 The type of action being taken on the header. This field can have the value `Set` or `Delete`.

When setting HTTP headers, you must provide a `value`. The value can be a string from a list of available directives for that header, for example `DENY`, or it can be a dynamic value that will be interpreted using HAProxy's dynamic value syntax. In this case, a dynamic value is added.



For setting dynamic header values for HTTP responses, allowed sample fetchers are `res.hdr` and `ssl_c_der`. For setting dynamic header values for HTTP requests, allowed sample fetchers are `req.hdr` and `ssl_c_der`. Both request and response dynamic values can use the `lower` and `base64` converters.

- Save the file to apply the changes.

Using X-Forwarded headers

You configure the HAProxy Ingress Controller to specify a policy for how to handle HTTP headers including `Forwarded` and `X-Forwarded-For`. The Ingress Operator uses the `HTTPHeaders` field to configure the `ROUTER_SET_FORWARDED_HEADERS` environment variable of the Ingress Controller.

Procedure

- Configure the `HTTPHeaders` field for the Ingress Controller.
 - Use the following command to edit the `IngressController` resource:

```
$ oc edit IngressController
```

- Under `spec`, set the `HTTPHeaders` policy field to `Append`, `Replace`, `IfNone`, or `Never`:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    forwardedHeaderPolicy: Append
```

Example use cases

As a cluster administrator, you can:

- Configure an external proxy that injects the `X-Forwarded-For` header into each request before forwarding it to an Ingress Controller.

To configure the Ingress Controller to pass the header through unmodified, you specify the `never` policy. The Ingress Controller then never sets the headers, and applications receive only the headers that the external proxy provides.

- Configure the Ingress Controller to pass the `X-Forwarded-For` header that your external proxy sets on external cluster requests through unmodified.

To configure the Ingress Controller to set the `X-Forwarded-For` header on internal cluster requests, which do not go through the external proxy, specify the `if-none` policy. If an HTTP request already has the header set through the external proxy, then the Ingress Controller preserves it. If the header is absent because the request did not come through the proxy, then the Ingress Controller adds the header.

As an application developer, you can:

- Configure an application-specific external proxy that injects the `X-Forwarded-For` header.

To configure an Ingress Controller to pass the header through unmodified for an application's Route, without affecting the policy for other Routes, add an annotation `haproxy.router.openshift.io/set-forwarded-headers: if-none` or `haproxy.router.openshift.io/set-forwarded-headers: never` on the Route for the application.



You can set the `haproxy.router.openshift.io/set-forwarded-headers` annotation on a per route basis, independent from the globally set value for the Ingress Controller.

Enabling HTTP/2 Ingress connectivity

You can enable transparent end-to-end HTTP/2 connectivity in HAProxy. It allows application owners to make use of HTTP/2 protocol capabilities, including single connection, header compression, binary streams, and more.

You can enable HTTP/2 connectivity for an individual Ingress Controller or for the entire cluster.

To enable the use of HTTP/2 for the connection from the client to HAProxy, a route must specify a custom certificate. A route that uses the default certificate cannot use HTTP/2. This restriction is necessary to avoid problems from connection coalescing, where the client re-uses a connection for different routes that use the same certificate.

The connection from HAProxy to the application pod can use HTTP/2 only for re-encrypt routes and not for edge-terminated or insecure routes. This restriction is because HAProxy uses Application-Level Protocol Negotiation (ALPN), which is a TLS extension, to negotiate the use of HTTP/2 with the back-end. The implication is that end-to-end HTTP/2 is possible with passthrough and re-encrypt and not with insecure or edge-terminated routes.



For non-passthrough routes, the Ingress Controller negotiates its connection to the application independently of the connection from the client. This means a client may connect to the Ingress Controller and negotiate HTTP/1.1, and the Ingress Controller may then connect to the application, negotiate HTTP/2, and forward the request from the client HTTP/1.1 connection using the HTTP/2 connection to the application. This poses a problem if the client subsequently tries to upgrade its connection from HTTP/1.1 to the WebSocket protocol, because the Ingress Controller cannot forward WebSocket to HTTP/2 and cannot upgrade its HTTP/2 connection to WebSocket. Consequently, if you have an application that is intended to accept WebSocket connections, it must not allow negotiating the HTTP/2 protocol or else clients will fail to upgrade to the WebSocket protocol.

Procedure

Enable HTTP/2 on a single Ingress Controller.

- To enable HTTP/2 on an Ingress Controller, enter the `oc annotate` command:

```
$ oc -n openshift-ingress-operator annotate  
ingresscontrollers/<ingresscontroller_name>  
ingress.operator.openshift.io/default-enable-http2=true
```

Replace `<ingresscontroller_name>` with the name of the Ingress Controller to annotate.

Enable HTTP/2 on the entire cluster.

- To enable HTTP/2 for the entire cluster, enter the `oc annotate` command:

```
$ oc annotate ingresses.config/cluster  
ingress.operator.openshift.io/default-enable-http2=true
```



You can alternatively apply the following YAML to add the annotation:

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
  annotations:
    ingress.operator.openshift.io/default-enable-http2: "true"
```

Configuring the PROXY protocol for an Ingress Controller

A cluster administrator can configure [the PROXY protocol](#) when an Ingress Controller uses either the `HostNetwork` or `NodePortService` endpoint publishing strategy types. The PROXY protocol enables the load balancer to preserve the original client addresses for connections that the Ingress Controller receives. The original client addresses are useful for logging, filtering, and injecting HTTP headers. In the default configuration, the connections that the Ingress Controller receives only contain the source address that is associated with the load balancer.

This feature is not supported in cloud deployments. This restriction is because when OpenShift Container Platform runs in a cloud platform, and an IngressController specifies that a service load balancer should be used, the Ingress Operator configures the load balancer service and enables the PROXY protocol based on the platform requirement for preserving source addresses.



You must configure both OpenShift Container Platform and the external load balancer to either use the PROXY protocol or to use TCP.



The PROXY protocol is unsupported for the default Ingress Controller with installer-provisioned clusters on non-cloud platforms that use a Keepalived Ingress VIP.

Prerequisites

- You created an Ingress Controller.

Procedure

- Edit the Ingress Controller resource:

```
$ oc -n openshift-ingress-operator edit  
ingresscontroller/default
```

- Set the PROXY configuration:
 - If your Ingress Controller uses the hostNetwork endpoint publishing strategy type, set the `spec.endpointPublishingStrategy.hostNetwork.protocol` subfield to PROXY:

Sample hostNetwork configuration to PROXY

```
spec:  
  endpointPublishingStrategy:  
    hostNetwork:  
      protocol: PROXY  
    type: HostNetwork
```

- If your Ingress Controller uses the NodePortService endpoint publishing strategy type, set the `spec.endpointPublishingStrategy.nodePort.protocol` subfield to PROXY:

Sample nodePort configuration to PROXY

```
spec:  
  endpointPublishingStrategy:  
    nodePort:  
      protocol: PROXY  
    type: NodePortService
```

Specifying an alternative cluster domain using the appsDomain option

As a cluster administrator, you can specify an alternative to the default cluster domain for user-created routes by configuring the `appsDomain` field. The `appsDomain` field is an optional domain for OpenShift Container Platform to use instead of the default, which is specified in the `domain` field. If you specify an alternative domain, it overrides the default cluster domain for the purpose of determining the default host for a new route.

For example, you can use the DNS domain for your company as the default domain for routes and ingresses for applications running on your cluster.

Prerequisites

- You deployed an OpenShift Container Platform cluster.
- You installed the `oc` command line interface.

Procedure

- Configure the `appsDomain` field by specifying an alternative default domain for user-created routes.
 - Edit the ingress `cluster` resource:

```
$ oc edit ingresses.config/cluster -o yaml
```

- Edit the YAML file:

Sample `appsDomain` configuration to `test.example.com`

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: apps.example.com           (1)
  appsDomain: <test.example.com>    (2)
```

- 1 Specifies the default domain. You cannot modify the default domain after installation.

- Optional: Domain for OpenShift Container Platform infrastructure to use for application routes. Instead of the default prefix, `apps`, you can use an alternative prefix like `test`.

- Verify that an existing route contains the domain name specified in the `appsDomain` field by exposing the route and verifying the route domain change:



Wait for the `openshift-apserver` finish rolling updates before exposing the route.

- Expose the route:

```
$ oc expose service hello-openshift
route.route.openshift.io/hello-openshift exposed
```

Example output:

```
$ oc get routes
NAME                                HOST/PORT
PATH    SERVICES                PORT          TERMINATION  WILDCARD
hello-openshift  hello_openshift-
<my_project>.test.example.com
hello-openshift  8080-tcp                      None
```

Converting HTTP header case

HAProxy lowercases HTTP header names by default, for example, changing `Host: xyz.com` to `host: xyz.com`. If legacy applications are sensitive to the capitalization of HTTP header names, use the Ingress Controller `spec.httpHeaders.headerNameCaseAdjustments` API field for a solution to accommodate legacy applications until they can be fixed.



Because OpenShift Container Platform includes HAProxy 2.6, be sure to add the necessary configuration by using `spec.httpHeaders.headerNameCaseAdjustments` before upgrading.

Prerequisites

- You have installed the OpenShift CLI (`oc`).
- You have access to the cluster as a user with the `cluster-admin` role.

Procedure

As a cluster administrator, you can convert the HTTP header case by entering the `oc patch` command or by setting the `HeaderNameCaseAdjustments` field in the Ingress Controller YAML file.

- Specify an HTTP header to be capitalized by entering the `oc patch` command.
 - Enter the `oc patch` command to change the HTTP host header to `Host`:

```
$ oc -n openshift-ingress-operator patch
ingresscontrollers/default --type=merge --patch='{"spec":
{"httpHeaders":{"headerNameCaseAdjustments":["Host"]}}'
```

- Annotate the route of the application:

```
$ oc annotate routes/my-application
haproxy.router.openshift.io/h1-adjust-case=true
```

The Ingress Controller then adjusts the `host` request header as specified.

- Specify adjustments using the `HeaderNameCaseAdjustments` field by configuring the Ingress Controller YAML file.
 - The following example Ingress Controller YAML adjusts the `host` header to `Host` for HTTP/1 requests to appropriately annotated routes:

Example Ingress Controller YAML

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    headerNameCaseAdjustments:
      - Host
```

- The following example route enables HTTP response header name case adjustments using the `haproxy.router.openshift.io/h1-adjust-case` annotation:

Example route YAML

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true (1)
  name: my-application
  namespace: my-application
spec:
  to:
    kind: Service
    name: my-application
```

1 Set `haproxy.router.openshift.io/h1-adjust-case` to true.

Using router compression

You configure the HAProxy Ingress Controller to specify router compression globally for specific MIME types. You can use the `mimeTypes` variable to define the formats of MIME types to which compression is applied. The types are: application, image, message, multipart, text, video, or a custom type prefaced by "X-". To see the full notation for MIME types and subtypes, see [RFC1341](#).



Memory allocated for compression can affect the max connections. Additionally, compression of large buffers can cause latency, like heavy regex or long lists of regex.

Not all MIME types benefit from compression, but HAProxy still uses resources to try to compress if instructed to. Generally, text formats, such as html, css, and js, formats benefit from compression, but formats that are already compressed, such as image, audio, and video, benefit little in exchange for the time and resources spent on compression.

Procedure

- Configure the `httpCompression` field for the Ingress Controller.
 - Use the following command to edit the `IngressController` resource:

```
$ oc edit -n openshift-ingress-operator
ingresscontrollers/default
```

- Under `spec`, set the `httpCompression` policy field to `mimeTypes` and specify a list of MIME types that should have compression applied:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpCompression:
    mimeTypes:
      - "text/html"
      - "text/css; charset=utf-8"
      - "application/json"
    ...
```

Exposing router metrics

You can expose the HAProxy router metrics by default in Prometheus format on the default stats port, 1936. The external metrics collection and aggregation systems such as Prometheus can access the HAProxy router metrics. You can view the HAProxy router metrics in a browser in the HTML and comma separated values (CSV) format.

Prerequisites

- You configured your firewall to access the default stats port, 1936.

Procedure

- Get the router pod name by running the following command:

```
$ oc get pods -n openshift-ingress
```

Example output

| NAME | READY | STATUS | RESTARTS |
|----------------------------------|-------|---------|----------|
| AGE | | | |
| router-default-76bffffb66c-46qwp | 1/1 | Running | 0 |
| 11h | | | |

- Get the router's username and password, which the router pod stores in the `/var/lib/haproxy/conf/metrics-auth/statsUsername` and `/var/lib/haproxy/conf/metrics-auth/statsPassword` files:

- Get the username by running the following command:

```
$ oc rsh <router_pod_name> cat metrics-auth/statsUsername
```

- Get the password by running the following command:

```
$ oc rsh <router_pod_name> cat metrics-auth/statsPassword
```

- Get the router IP and metrics certificates by running the following command:

```
$ oc describe pod <router_pod>
```

- Get the raw statistics in Prometheus format by running the following command:

```
$ curl -u <user>:<password> http://<router_IP>:
<stats_port>/metrics
```

- Access the metrics securely by running the following command:

```
$ curl -u user:password https://<router_IP>:
<stats_port>/metrics -k
```

- Access the default stats port, 1936, by running the following command:

```
$ curl -u <user>:<password> http://<router_IP>:
<stats_port>/metrics
```

Example output

```

...
# HELP haproxy_backend_connections_total Total number of
connections.
# TYPE haproxy_backend_connections_total gauge
haproxy_backend_connections_total{backend="http",namespace="def
ault",route="hello-route"} 0
haproxy_backend_connections_total{backend="http",namespace="def
ault",route="hello-route-alt"} 0
haproxy_backend_connections_total{backend="http",namespace="def
ault",route="hello-route01"} 0
...
# HELP haproxy_exporter_server_threshold Number of servers
tracked and the current threshold value.
# TYPE haproxy_exporter_server_threshold gauge
haproxy_exporter_server_threshold{type="current"} 11
haproxy_exporter_server_threshold{type="limit"} 500
...
# HELP haproxy_frontend_bytes_in_total Current total of
incoming bytes.
# TYPE haproxy_frontend_bytes_in_total gauge
haproxy_frontend_bytes_in_total{frontend="fe_no_sni"} 0
haproxy_frontend_bytes_in_total{frontend="fe_sni"} 0
haproxy_frontend_bytes_in_total{frontend="public"} 119070
...
# HELP haproxy_server_bytes_in_total Current total of incoming
bytes.
# TYPE haproxy_server_bytes_in_total gauge
haproxy_server_bytes_in_total{namespace="",pod="",route="",serv
er="fe_no_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="",pod="",route="",serv
er="fe_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="default",pod="docker-
registry-5-nk5fz",route="docker-
registry",server="10.130.0.89:5000",service="docker-registry"}
0
haproxy_server_bytes_in_total{namespace="default",pod="hello-
rc-vkjqx",route="hello-
route",server="10.130.0.90:8080",service="hello-svc-1"} 0
...

```

- Launch the stats window by entering the following URL in a browser:

```
http://<user>:<password>@<router_IP>:<stats_port>
```

- Optional: Get the stats in CSV format by entering the following URL in a browser:

```
http://<user>:<password>@<router_ip>:1936/metrics;csv
```

Customizing HAProxy error code response pages

As a cluster administrator, you can specify a custom error code response page for either 503, 404, or both error pages. The HAProxy router serves a 503 error page when the application pod is not running or a 404 error page when the requested URL does not exist. For example, if you customize the 503 error code response page, then the page is served when the application pod is not running, and the default 404 error code HTTP response page is served by the HAProxy router for an incorrect route or a non-existing route.

Custom error code response pages are specified in a config map then patched to the Ingress Controller. The config map keys have two available file names as follows: `error-page-503.http` and `error-page-404.http`.

Custom HTTP error code response pages must follow the [HAProxy HTTP error page configuration guidelines](#). Here is an example of the default OpenShift Container Platform HAProxy router [http 503 error code response page](#). You can use the default content as a template for creating your own custom page.

By default, the HAProxy router serves only a 503 error page when the application is not running or when the route is incorrect or non-existent. This default behavior is the same as the behavior on OpenShift Container Platform 4.8 and earlier. If a config map for the customization of an HTTP error code response is not provided, and you are using a custom HTTP error code response page, the router serves a default 404 or 503 error code response page.



If you use the OpenShift Container Platform default 503 error code page as a template for your customizations, the headers in the file require an editor that can use CRLF line endings.

Procedure

- Create a config map named `my-custom-error-code-pages` in the `openshift-config` namespace:

```
$ oc -n openshift-config create configmap my-custom-error-code-  
pages \  
--from-file=error-page-503.http \  
--from-file=error-page-404.http
```



If you do not specify the correct format for the custom error code response page, a router pod outage occurs. To resolve this outage, you must delete or correct the config map and delete the affected router pods so they can be recreated with the correct information.

- Patch the Ingress Controller to reference the `my-custom-error-code-pages` config map by name:

```
$ oc patch -n openshift-ingress-operator  
ingresscontroller/default --patch '{"spec":  
{"httpErrorCodePages":{"name":"my-custom-error-code-pages"}}}'  
--type=merge
```

The Ingress Operator copies the `my-custom-error-code-pages` config map from the `openshift-config` namespace to the `openshift-ingress` namespace. The Operator names the config map according to the pattern, `<your_ingresscontroller_name>-errorpages`, in the `openshift-ingress` namespace.

- Display the copy:

```
$ oc get cm default-errorpages -n openshift-ingress
```

Example output

| NAME | DATA | AGE |
|--------------------|------|---------|
| default-errorpages | 2 | 25s (1) |

- 1 The example config map name is `default-errorpages` because the default Ingress Controller custom resource (CR) was patched.

- Confirm that the config map containing the custom error response page mounts on the router volume where the config map key is the filename that has the custom HTTP error code response:

- For 503 custom HTTP custom error code response:

```
$ oc -n openshift-ingress rsh <router_pod> cat  
/var/lib/haproxy/conf/error_code_pages/error-page-503.http
```

- For 404 custom HTTP custom error code response:

```
$ oc -n openshift-ingress rsh <router_pod> cat  
/var/lib/haproxy/conf/error_code_pages/error-page-404.http
```

Verification

Verify your custom error code HTTP response:

- Create a test project and application:

```
$ oc new-project test-ingress
```

```
$ oc new-app django-psql-example
```

- For 503 custom http error code response:
 - Stop all the pods for the application.
 - Run the following curl command or visit the route hostname in the browser:

```
$ curl -vk <route_hostname>
```

- For 404 custom http error code response:
 - Visit a non-existent route or an incorrect route.
 - Run the following curl command or visit the route hostname in the browser:

```
$ curl -vk <route_hostname>
```

- Check if the `errorfile` attribute is properly in the `haproxy.config` file:

```
$ oc -n openshift-ingress rsh <router> cat  
/var/lib/haproxy/conf/haproxy.config | grep errorfile
```

Setting the Ingress Controller maximum connections

A cluster administrator can set the maximum number of simultaneous connections for OpenShift router deployments. You can patch an existing Ingress Controller to increase the maximum number of connections.

Prerequisites

- The following assumes that you already created an Ingress Controller

Procedure

- Update the Ingress Controller to change the maximum number of connections for HAProxy:

```
$ oc -n openshift-ingress-operator patch  
ingresscontroller/default --type=merge -p '{"spec":  
{"tuningOptions": {"maxConnections": 7500}}}'
```



If you set the `spec.tuningOptions.maxConnections` value greater than the current operating system limit, the HAProxy process will not start. See the table in the "Ingress Controller configuration parameters" section for more information about this parameter.

Additional resources

- [Configuring a custom PKI](#)



Copyright © 2024 Red Hat, Inc.