# Helm chart components

## nginx-ingress-controller

Nginx ingress controller executes 2 binaries: nginx binary and nginx ingress binary. Nginx is an nginx web server, nginx-ingress is a self-written binary. The goal of this Ingress controller is the assembly of a configuration file (nginx.conf).

There are 2 widely used nginx-ingress realizations: the first is supported by Kubernetes community, and the second is supported by NGINX Inc community. The first is based on the custom NGINX build with several third-party modules, while the second is based on NGINX official mainline build. A full comparison is published here: <https://github.com/nginxinc/kubernetes-ingress/blob/master/docs/nginx-ingress-controllers.md>

ModSecurity is a web application firewall which is used to protect your web application from a variety of attacks. ModSecurity is a third-party module for nginx.

CloudLinux detects and blocks malicious malware - rootkits, backdoors, worms, known exploits and zero-day attacks - by means of self written rules. These rules are written for ModSecurity v3. CloudLinux constantly updates security rules, and offers its own nginx ingress controller with them. CloudLinux requires nginx built with ModSecurity-nginx module, 3-rd version.

There is a docker-image of nginx ingress controller, kubernetes community, built with modsecurity 3, supported by quay: <https://quay.io/repository/kubernetes-ingress-controller/nginx-ingress-controller?tab=info>

The full list of 3-rd party modules, supported by Quay builds:

* [nginx-http-auth-digest](https://github.com/atomx/nginx-http-auth-digest)
* [ngx\_http\_substitutions\_filter\_module](https://github.com/yaoweibin/ngx_http_substitutions_filter_module)
* [nginx-opentracing](https://github.com/opentracing-contrib/nginx-opentracing)
* [opentracing-cpp](https://github.com/opentracing/opentracing-cpp)
* [zipkin-cpp-opentracing](https://github.com/rnburn/zipkin-cpp-opentracing)
* [dd-opentracing-cpp](https://github.com/DataDog/dd-opentracing-cpp)
* [ModSecurity-nginx](https://github.com/SpiderLabs/ModSecurity-nginx) (only supported in x86\_64)
* [brotli](https://github.com/google/brotli)
* [geoip2](https://github.com/leev/ngx_http_geoip2_module)

The latest version supports ModSecurity 3.20

<https://github.com/kubernetes/ingress-nginx/tree/master/images/nginx>

## modsecurity configuration storage

The nginx-ingress repository, supported by kubernetes community, contains the NGINX controller built around the [Kubernetes Ingress resource](http://kubernetes.io/docs/user-guide/ingress/) that uses [ConfigMap](https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/#understanding-configmaps-and-pods) to store the NGINX configuration. ConfigMap is stored in etcd, that’s why its size is restricted to 1Mb.

CloudLinux modsecurity rules size is about 4Mb, and it can be increased significantly in the future. That’s why ConfigMaps aren’t used as a storage of CloudLinux security rules.

Often k8s pods contain InitContainers. InitContainers - for example - download configuration files and fix them for a current k8s deployment: fix service addresses, network configurations etc.

CloudLinux ingress controller prepares security rules by means of InitContainer, and uses an EmptyDir volume type as a configuration storage. By default, emptyDir volumes are stored on whatever medium is backing the node - that might be disk or SSD or network storage, depending on your environment. EmptyDir is created during POD initialization, and all containers of the same pod can share it.

TODO: compare persistent and non-persistent volumes. Check which k8s volume types are compatible with nginx-ingress (when nginx ingress will receive notifications about configuration changes)

## Kubernetes ingress object

Ingress is an API object that manages external access to the services in a cluster, typically HTTP. Ingress exposes HTTP and HTTPS routes from outside the cluster to [services](https://kubernetes.io/docs/concepts/services-networking/service/) within the cluster.

Behavior of the specific kubernetes ingress object is customized by its annotations. A full list of supported annotations is published here: <https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/>

For example, if the ingress controller should apply ModSecurity checks, annotations must enable a modsecurity module (look at nginx.ingress.kubernetes.io/enable-modsecurity value). Moreover, a modsecurity snippet should allow security engine - value SecRuleEngine, point to log format and security configuration files. CloudLinux InitContainers store CloudLinux security rules in the /etc/nginx/cloudlinux/im360rules/ folder, and the next ingress annotation is correct:

apiVersion: v1

items:

- apiVersion: extensions/v1beta1

kind: Ingress

metadata:

annotations:

nginx.ingress.kubernetes.io/enable-modsecurity: "true"

nginx.ingress.kubernetes.io/modsecurity-snippet: |

SecRuleEngine On

SecRequestBodyAccess On

SecAuditEngine RelevantOnly

SecAuditLogParts ABIJDEFHZ

SecAuditLog /var/log/modsec\_audit.log

SecRule REQUEST\_HEADERS:User-Agent "test-block-header" "log,deny,id:107,status:403,msg:\'CLoudLinux test Identified\'"

Include /etc/nginx/cloudlinux/im360rules/\*conf

Any rules can be added by means of SecRule annotation.

## 3-rd party docker images

CloudLinux helm chart depends on the next nginx-ingress docker image:

controller:

name: controller

image:

repository: quay.io/kubernetes-ingress-controller/nginx-ingress-controller

tag: "0.26.1"

CloudLinux InitContainers depend on busybox docker images.

# Nginx ingress deployment

## Types of ingress

<https://kubernetes.io/docs/concepts/services-networking/ingress/>

There are 3 types of ingress deployment:

* single service ingress, which exposes a single service;
* Simple fannout. A fanout configuration routes traffic from a single IP address to more than one Service, based on the HTTP URI being requested;
* Name based virtual hosting. Name-based virtual hosts support routing HTTP traffic to multiple host names at the same IP address.

How to configure the last type of ingress - name-based virtual hosting - is described below.

## Protected applications

A testing environment is similar to this one: <https://medium.com/@awkwardferny/enabling-modsecurity-in-the-kubernetes-ingress-nginx-controller-111f9c877998>

In the beginning, pods(containers) and services should be deployed. The next deployments can be used:

*apiVersion: v1*

*kind: Service*

*metadata:*

*name: hello-kubernetes-first*

*spec:*

*type: ClusterIP*

*ports:*

*- port: 80*

*targetPort: 8080*

*selector:*

*app: hello-kubernetes-first*

*---*

*apiVersion: apps/v1*

*kind: Deployment*

*metadata:*

*name: hello-kubernetes-first*

*spec:*

*replicas: 3*

*selector:*

*matchLabels:*

*app: hello-kubernetes-first*

*template:*

*metadata:*

*labels:*

*app: hello-kubernetes-first*

*spec:*

*containers:*

*- name: hello-kubernetes*

*image: paulbouwer/hello-kubernetes:1.5*

*ports:*

*- containerPort: 8080*

*env:*

*- name: MESSAGE*

*value: Hello from the first deployment!*

A service named “hello-kubernetes-second” and its pods have to be created in a similar way.

Next, check that services and deployments have been created successfully:

*kubeusr@master-node:~$ kubectl get deployments -n default*

*NAME READY UP-TO-DATE AVAILABLE AGE*

*hello-kubernetes-first 3/3 3 3 1h*

*hello-kubernetes-second 3/3 3 3 1h*

*kubeusr@master-node:~$ kubectl get services -n default*

*NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE*

*hello-kubernetes-first ClusterIP 10.109.254.98 <none> 80/TCP 1h*

*hello-kubernetes-second ClusterIP 10.100.1.206 <none> 80/TCP 1h*

## Ingress deployment

Next, setup the Ingress Rules, enabling ModSecurity with a custom snippet:

*apiVersion: extensions/v1beta1*

*kind: Ingress*

*metadata:*

*annotations:*

*nginx.ingress.kubernetes.io/enable-modsecurity: "true"*

*nginx.ingress.kubernetes.io/modsecurity-snippet: |*

*SecDebugLog /var/log/modsec\_debug.log*

*SecRuleEngine On*

*SecRequestBodyAccess On*

*SecAuditEngine RelevantOnly*

*SecAuditLogParts ABIJDEFHZ*

*SecAuditLog /var/log/modsec\_audit.log*

*SecRule REQUEST\_HEADERS:User-Agent "cloudlinux-test" "log,deny,id:107,status:403,msg:\'CloudLinux test Identified\'"*

*Include /etc/nginx/cloudlinux/im360rules/\*conf*

*name: cloudlinux-kubernetes-ingress*

*namespace: default*

*spec:*

*rules:*

*- host: hw1.example.com*

*http:*

*paths:*

*- backend:*

*serviceName: hello-kubernetes-first*

*servicePort: 80*

*path: /*

*- host: hw2.example.com*

*http:*

*paths:*

*- backend:*

*serviceName: hello-kubernetes-second*

*servicePort: 80*

*path: /*

*status:*

*loadBalancer: {}*

All traffic designed for hw1.example.com will be delivered to hello-kubernetes-first, and all traffic designed for hw2.example.com will be delivered to hello-kubernetes-second.

The next command shows ingress state:

***kubeusr@master-node****:****~****$ kubectl get ingress*

*NAME HOSTS ADDRESS PORTS AGE*

*cloudlinux-kubernetes-ingress hw1.example.com,hw2.example.com 80 1h*

## Helm chart deployment

CloudLinux ingress helm chart should be deployed. It can be installed from repository ADD\_URL\_HERE, or it can be downloaded and installed from

*kubeusr@master-node:~$helm install testingress /tmp/clouglinux-nginx-ingress.tgz*

Next, check that chart has been installed successfully:

***kubeusr@master-node****:****~****$helm list*

*NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION*

*testingress default 1 2020-01-08 22:58:58.112016415 +0000 UTC deployed cloudlinux-nginx-ingress-1.26.20200108 0.26.1*

***k***

Now check that nginx ingress pods are running:

***kubeusr@master-node****:****~****$ kubectl get pods -n default*

*NAME READY STATUS RESTARTS AGE*

*testingress-nginx-ingress-controller-7955b947-2pd5r 1/1 Running 0 1h*

*testingress-nginx-ingress-default-backend-5c6d47dcd9-7cbwg 1/1 Running 0 1h*

Add dns records for hw1.example.com and hw2.example.com, and test traffic routing:

curl <http://hw2.example.com/> will return page with <td>hello-kubernetes-second-ID</td>,

curl [http://hw1.example.com/](http://hw2.example.com/) will return page with <td>hello-kubernetes-first-ID</td>.

curl --user-agent "cloudlinux-test" <http://hw2.example.com/> will return 403 error, and specially prepared traffic will be blocked too.