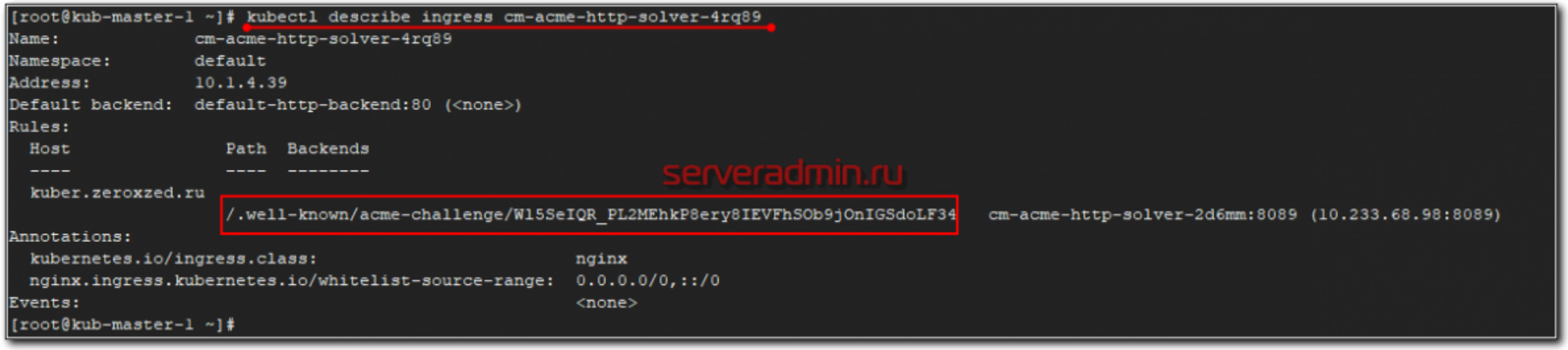

Продолжаю цикл статей по настройке и эксплуатации кластера Kubernetes. Сегодня рассмотрю отдельно настройку Ingress контроллера для публикации сервисов и приложений в кластере. Принцип его работы я кратко показал в обзорной статье по работе с кластером. Сегодня рассмотрим более подробно.

Содержание:

- 1 Цели статьи
- 2 Принцип работы Ingress контроллера
- 3 SSL/TLS сертификаты в Ingress
- 4 Cert-manager

- 5 Логи и конфиг Ingress Controller
- 6 Конфигурация Nginx для Ingress Controller
- 7 Заключение

Цели статьи

1. Рассказать о принципе работы Ingress Controller в кластере Kubernetes.
2. Показать, как добавлять свои сертификаты в конфигурацию ingress.
3. На конкретном примере показать работу cert-manager для управления ssl/tls сертификатами. В том числе автоматическое получение сертификатов от Let’s Encrypt.
4. Показать, где хранятся конфигурация и логи работы ingress controller.
5. Рассказать, как устанавливать дополнительные параметры nginx в ingress контроллере.

Принцип работы Ingress контроллера

Сразу важное замечание, чтобы потом не было путаницы.

- **Ingress** — сущность кластера kubernetes, в которой вы описываете конфигурацию для ingress controller.
- **Ingress Controller** — занимается непосредственно обработкой траффика. Его конфигурация формируется из всех ingress, которые есть в кластере.

Реализация ingress controller может быть на базе различного софта — nginx, haproxy, traefik и т.д. Стандартно Kubernetes использует контроллер на базе Nginx. В моей статье будет именно он. Это самый простой и популярный вариант. В основном все его используют.

В проде обычно ingress контроллеры вешают на отдельные узлы с внешними ip адресами и используют в качестве точки подключения к сервисам. Ingress controller может как сам принимать на себя весь трафик, так и работать за каким-то внешним балансером. Конкретную реализацию выбирать вам. Так как под капотом там Nginx, каких-то особых проблем с безопасностью при размещении Ingress напрямую в интернет нету.

Нужно понимать, что ingress работает на 7-м уровне сетевой модели OSI. Если вам нужен 3-й уровень, используйте другие способы публикации приложений:

- ClusterIP
- NodePort

- LoadBalancer
- ExternalName
- ExternalIP

Так как внутри Ingress контроллера обычный Nginx, вы можете очень гибко управлять распределением запросов по доменам, location и т.д. Вам доступно все, что умеет nginx в функционале proxy_pass.

Сразу приведу ссылку на официальную документацию по ingress controller на базе nginx — https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/. Там есть все то, о чем я расскажу ниже.

На простом примере работу Ingress контроллера я уже показал ранее в статье Работа с кластером Kubernetes. Здесь же я рассмотрю его настройку подробнее, особое внимание уделив работе в связке с cert-manager.

SSL/TLS сертификаты в Ingress

Начну с одного из основного функционала — настройка SSL/TLS сертификатов. Сейчас без них никуда. Для того, чтобы использовать сертификаты в конфигурации Ingress, их нужно загрузить в кластер Kubernetes. Сертификаты для Ingress хранятся в секретах Кубера. Создать такой секрет можно следующим образом.

```
# kubectl create secret tls my-tls — key my-tls.key — cert my-tls.cert
```

Дальше этот секрет можно использовать в конфигурации ingress, например вот так.

```
apiVersion: extensions/v1beta1
 kind: Ingress
 metadata:
   name: ingress-my-tls
   annotations:
     ingress.kubernetes.io/ssl-redirect: "true"
```

```
spec:
  tls:
  - hosts:
    - ingress.example.com
    secretName: my-tls
  rules:
  - host: ingress.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: nginx
          servicePort: 80
```

Если у вас есть Wildcard-сертификат, вы можете добавить его в kubernetes и установить дефолтным для всех доменов в ingress, где явно не указан персональный сертификат. Для этого необходимо добавить дополнительный параметр **default-ssl-certificate** в deployment для ingress контроллера. О том, как это сделать, подробно рассказано в статье на success.docker.com.

Cert-manager

Cert-manager — утилита в кластере Kubernetes, которая умеет автоматически получать и продлевать сертификаты от различных удостоверяющих центров, в том числе от бесплатного Let's Encrypt. При этом она интегрируется с Ingress Controller.

В своей работе cert-manager использует CustomResourceDefinitions, которые нужно будет предварительно добавить в API кубернетиса — Issuer, ClusterIssuer, Certificate. Давайте установим cert-manager в свой кластер Kubernetes. Я буду использовать для этого helm и инструкцию с официального сайта.

```
# kubectl apply — validate=false -f
https://raw.githubusercontent.com/jetstack/cert-manager/release-0.11/deploy/manifests/00-crds.yaml
# kubectl create namespace cert-manager
# helm repo add jetstack https://charts.jetstack.io
```

```
# helm repo update
# helm install cert-manager — namespace cert-manager — version v0.11.0 jetstack/cert-manager
```

```
[root@kub-master-1 ~]# kubectl apply --validate=false -f https://raw.githubusercontent.com/jetstack/cert-manager/release-0.11/deploy/manifests/00-crds.yaml
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io created
[root@kub-master-1 ~]# kubectl create namespace cert-manager
namespace/cert-manager created
[root@kub-master-1 ~]# helm repo add jetstack https://charts.jetstack.io
"jetstack" has been added to your repositories
[root@kub-master-1 ~]# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "jetstack" chart repository
...Successfully got an update from the "bitnami" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. □ Happy Helming!□
[root@kub-master-1 ~]# helm install cert-manager --namespace cert-manager --version v0.11.0 jetstack/cert-manager
NAME: cert-manager
LAST DEPLOYED: Tue Nov 26 10:44:52 2019
NAMESPACE: cert-manager
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
cert-manager has been deployed successfully!

In order to begin issuing certificates, you will need to set up a ClusterIssuer
or Issuer resource (for example, by creating a 'letsencrypt-staging' issuer).

More information on the different types of issuers and how to configure them
can be found in our documentation:

https://docs.cert-manager.io/en/latest/reference/issuers.html

For information on how to configure cert-manager to automatically provision
Certificates for Ingress resources, take a look at the `ingress-shim`
documentation:

https://docs.cert-manager.io/en/latest/reference/ingress-shim.html
```


Проверяем, что все необходимые поды стартовали.

```
# kubectl get pods — namespace cert-manager
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-584d89d5f9-w46jh              1/1     Running   0          3m25s
cert-manager-cainjector-6df9c84978-lcnb4   1/1     Running   0          3m25s
cert-manager-webhook-69c9b44f6-rnkwz       1/1     Running   0          3m25s
```

Теперь проверим, что cert-manager нормально установился и работает. Для этого можно выпустить самоподписанный сертификат. Создаем манифест *test-resources.yaml*.

```
apiVersion: v1
kind: Namespace
metadata:
  name: cert-manager-test
---
apiVersion: cert-manager.io/v1alpha2
kind: Issuer
metadata:
  name: test-selfsigned
  namespace: cert-manager-test
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: selfsigned-cert
  namespace: cert-manager-test
spec:
  commonName: example.com
  secretName: selfsigned-cert-tls
```

```
  issuerRef:
    name: test-selfsigned
```

Применяем.

```
# kubectl apply -f test-resources.yaml
namespace/cert-manager-test created
issuer.cert-manager.io/test-selfsigned created
certificate.cert-manager.io/selfsigned-cert created
```

Проверяем, что получилось.

```
# kubectl describe certificate -n cert-manager-test
```

```
[root@kub-master-1 ~]# kubectl describe certificate -n cert-manager-test
Name:         selfsigned-cert
Namespace:    cert-manager-test
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"cert-manager.io/v1alpha2","kind":"Certificate","metadata":{"annotations":{},"name":"selfsigned-cert","namespace":"cert-mana...
API Version:  cert-manager.io/v1alpha2
Kind:         Certificate
Metadata:
  Creation Timestamp:  2019-11-26T07:54:11Z
  Generation:          1
  Resource Version:    15066969
  Self Link:           /apis/cert-manager.io/v1alpha2/namespaces/cert-manager-test/certificates/selfsigned-cert
  UID:                 adc39006-a69a-4f8d-817f-4e6c738b4f75
Spec:
  Common Name:  example.com
  Issuer Ref:
    Name:       test-selfsigned
  Secret Name:  selfsigned-cert-tls
Status:
  Conditions:
    Last Transition Time:  2019-11-26T07:54:20Z
    Message:               Certificate is up to date and has not expired
    Reason:                Ready
    Status:                True
    Type:                  Ready
  Not After:               2020-02-24T07:54:20Z
Events:
  Type    Reason        Age   From          Message
  ----    ------        ----  ----          -------
  Normal  GeneratedKey  26s   cert-manager  Generated a new private key
  Normal  Requested     26s   cert-manager  Created new CertificateRequest resource "selfsigned-cert-2334779822"
  Normal  Issued        26s   cert-manager  Certificate issued successfully
[root@kub-master-1 ~]#
```


Все в порядке. Сертификат выпущен, cert-manager работает. Посмотреть информацию о сертификате можно командой.

```
# kubectl -n cert-manager-test get certificate selfsigned-cert -o yaml
```

Описание секрета, где хранится сертификат.

```
# kubectl -n cert-manager-test describe secret selfsigned-cert-tls
```

А вот так можно увидеть сам сертификат из секрета.

```
# kubectl -n cert-manager-test get secret selfsigned-cert-tls -o yaml
```

Для того, чтобы узнать, кем был выпущен сертификат, нужно посмотреть информацию об issuer.

```
# kubectl -n cert-manager-test get issuer
NAME              AGE
test-selfsigned   8m16s
```

Напоминаю, что сущности Issuer, Certificat по-умолчанию отсутствуют в Kubernetes. Мы их добавили через CustomResourceDefinitions перед установкой cert-manager.

Мы убедились, что все нормально работает. Почистим за собой все тестовые сущности, удалив namespace, где работали.

```
# kubectl delete ns cert-manager-test
namespace "cert-manager-test" deleted
```

Переходим к выпуску сертификатов с помощью Let’s Encrypt. Для этого добавляем нового ClusterIssuer с помощью манифеста *clusterissuer.yaml*.

```
apiVersion: cert-manager.io/v1alpha2
kind: ClusterIssuer
```

```
metadata:
  name: letsencrypt
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: root@serveradmin.ru
    privateKeySecretRef:
      name: letsencrypt
    solvers:
    - http01:
        ingress:
          class: nginx
```

```
# kubectl apply -f clusterissuer.yaml
clusterissuer.cert-manager.io/letsencrypt created
```

Давайте теперь подключим сертификат от Let's Encrypt к магазину носков, который я рассматривал в статье по работе с кластером. Напомню, что взять его можно по ссылке — https://github.com/microservices-demo/microservices-demo. Там в директории /deploy/kybernetes есть готовый *complete-demo.yaml*, где описаны все сущности kubernetes для установки. Очень удобно использовать для тестов. Единственный нюанс — я изменил namespace sock-shop, просто убрав его, чтобы поставить магазин в namespace default, чтобы потом не указывать постоянно namespace в командах. Это просто ради удобства в написании статьи и отладки, чтобы было быстрее. Вы можете этого не делать.

Качаем, редактируем и применяем манифест.

```
# wget
https://raw.githubusercontent.com/microservices-demo/microservices-demo/master/deploy/kubernetes/complete-demo.yaml
# kubectl apply -f complete-demo.yaml
```

У вас должны подняться поды и все остальное от этого магазина. В частности нас интересуют service.

```
[root@kub-master-1 ansible]# kubectl get pod
NAME                              READY   STATUS    RESTARTS   AGE
carts-56c6fb966b-bjkcn            1/1     Running   0          85m
carts-db-5678cc578f-9h8zg         1/1     Running   0          85m
catalogue-644549d46f-p5mnv        1/1     Running   0          85m
catalogue-db-6ddc796b66-bkrdb     1/1     Running   0          85m
cm-acme-http-solver-pf2tq         1/1     Running   0          52m
front-end-5594987df6-5dp8z        1/1     Running   0          85m
orders-749cdc8c9-x84tx            1/1     Running   0          85m
orders-db-5cfc68c4cf-n529t        1/1     Running   0          85m
payment-54f55b96b9-pd869          1/1     Running   0          85m
queue-master-6fff667867-2489t     1/1     Running   0          85m
rabbitmq-bdfd84d55-xz4f4          1/1     Running   0          85m
shipping-78794fdb4f-sc2bx         1/1     Running   0          85m
user-77cff48476-s88cd             1/1     Running   0          85m
user-db-99685d75b-kbx88           1/1     Running   0          85m
[root@kub-master-1 ansible]# kubectl get service
NAME                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
carts                     ClusterIP   10.233.6.140    <none>        80/TCP           85m
carts-db                  ClusterIP   10.233.41.82    <none>        27017/TCP        85m
catalogue                 ClusterIP   10.233.49.226   <none>        80/TCP           85m
catalogue-db              ClusterIP   10.233.29.125   <none>        3306/TCP         85m
cm-acme-http-solver-2d6mm NodePort    10.233.62.190   <none>        8089:30140/TCP   52m
front-end                 NodePort    10.233.6.200    <none>        80:30001/TCP     85m
kubernetes                ClusterIP   10.233.0.1      <none>        443/TCP          8d
orders                    ClusterIP   10.233.18.140   <none>        80/TCP           85m
orders-db                 ClusterIP   10.233.44.237   <none>        27017/TCP        85m
payment                   ClusterIP   10.233.36.96    <none>        80/TCP           85m
queue-master              ClusterIP   10.233.28.116   <none>        80/TCP           85m
rabbitmq                  ClusterIP   10.233.37.170   <none>        5672/TCP         85m
shipping                  ClusterIP   10.233.8.253    <none>        80/TCP           85m
user                      ClusterIP   10.233.37.150   <none>        80/TCP           85m
user-db                   ClusterIP   10.233.23.151   <none>        27017/TCP        85m
[root@kub-master-1 ansible]#
```

Для сервиса front-end настраиваем ingress, сразу выпуская сертификат.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-socks-tls
  annotations:
    kubernetes.io/ingress.class: "nginx"
    cert-manager.io/cluster-issuer: "letsencrypt"
spec:
  rules:
  - host: kuber.zeroxzed.ru
    http:
      paths:
      - backend:
          serviceName: front-end
          servicePort: 80
  tls:
  - hosts:
    - kuber.zeroxzed.ru
    secretName: socks-tls-2
```

Применяем и проверяем.

```
# kubectl apply -f ingress-sock-tls.yaml
# kubectl describe ingress ingress-socks-tls
```

```
[root@kub-master-1 ~]# kubectl describe ingress ingress-socks-tls
Name:             ingress-socks-tls
Namespace:        default
Address:          10.1.4.39
Default backend:  default-http-backend:80 (<none>)
TLS:
  socks-tls-2 terminates kuber.zeroxzed.ru
Rules:
  Host               Path  Backends
  ----               ----  --------
  kuber.zeroxzed.ru
                     front-end:80 (10.233.68.93:8079)
Annotations:
  kubernetes.io/ingress.class:                      nginx
  cert-manager.io/cluster-issuer:                   letsencrypt
  kubectl.kubernetes.io/last-applied-configuration:  {"apiVersion":"extensions/v1beta1","kind":"Ingress","metadata":{"annotations":{"cert-manager.io/cluster-issuer":
"letsencrypt","kubernetes.io/ingress.class":"nginx"},"name":"ingress-socks-tls","namespace":"default"},"spec":{"rules":[{"host":"kuber.zeroxzed.ru","http":{"paths":[
{"backend":{"serviceName":"front-end","servicePort":80}}]}}],"tls":[{"hosts":["kuber.zeroxzed.ru"],"secretName":"socks-tls-2"}]}}

Events:  <none>
[root@kub-master-1 ~]#
```


Настройка ingress с сертификатом применилась. Сайт с магазином носков должен быть доступен по доменному имени. Теперь посмотрим, что у нас с сертификатами.

В момент написания статьи у меня возникли проблемы с подтверждением сертификата от lets encrypt, хотя ранее по такой же схеме я успешно с ними работал. В процессе разбирательств я выяснил, в чем проблема. Оказывается, перед тем, как отправить запрос на подтверждение в lets encrypt, cert-manager сам выполняет проверку домена по доменному имени. У меня кластер был без внешнего IP, скрыт за шлюзом, с которого просто выполнялся проброс портов через NAT. При такой схеме изнутри кластера при обращении к внешнему доменному имени, запрос не попадал на pod с ingress controller.

Чтобы все эта конструкция работала правильно с точки зрения cert-manager и его проверок, надо было либо настраивать корректно перенаправление портов не только с внешнего интерфейса внутрь кластера, но и с внутренней сети обратно внутрь на ноду с ingress controller. Либо настраивать внутренний dns сервер, который по внешнему доменному имени будет возвращать локальный ip адрес ноды с ingress controller.

Мне не захотелось всем этим заниматься в тестовом кластере. Но это даже хорошо, так как я дальше сразу покажу, как разбирать проблемы и куда смотреть в случае, если они появятся. В общем случае того, что мы уже сделали достаточно для того, чтобы сертификаты от Let’s Encrypt заработали в кластере Kubernetes. Итак, проверяем выпущенный сертификат.

```
# kubectl get certificate
NAME          READY   SECRET        AGE
socks-tls-2   False   socks-tls-2   73m
```

Он выпущен, но по какой-то причине не готов. Смотрим его описание.

```
# kubectl describe certificate socks-tls-2
```

```
[root@kub-master-1 ~]# kubectl describe certificate socks-tls-2
Name:         socks-tls-2
Namespace:    default
Labels:       <none>
Annotations:  <none>
API Version:  cert-manager.io/v1alpha2
Kind:         Certificate
Metadata:
  Creation Timestamp:  2019-12-03T08:49:49Z
  Generation:          1
  Owner References:
    API Version:           extensions/v1beta1
    Block Owner Deletion:  true
    Controller:            true
    Kind:                  Ingress
    Name:                  ingress-socks-tls
    UID:                   ca4b5c80-bcc6-4673-9671-287badc66064
  Resource Version:        16818431
  Self Link:               /apis/cert-manager.io/v1alpha2/namespaces/default/certificates/socks-tls-2
  UID:                     8f4ad88b-ea86-4972-8980-a609b67f80c8
Spec:
  Dns Names:
    kuber.zeroxzed.ru
  Issuer Ref:
    Group:      cert-manager.io
    Kind:       ClusterIssuer
    Name:       letsencrypt
  Secret Name:  socks-tls-2
Status:
  Conditions:
    Last Transition Time:  2019-12-03T08:49:48Z
    Message:               Waiting for CertificateRequest "socks-tls-2-2550861952" to complete
    Reason:                InProgress
    Status:                False
    Type:                  Ready
Events:                    <none>
[root@kub-master-1 ~]#
```

Посмотрим внимательно на запрос.

```
# kubectl describe CertificateRequest socks-tls-2-2550861952
```

```
[root@kub-master-1 ~]# kubectl describe CertificateRequest socks-tls-2-2550861952
Name:         socks-tls-2-2550861952
Namespace:    default
Labels:       <none>
Annotations:  cert-manager.io/certificate-name: socks-tls-2
              cert-manager.io/private-key-secret-name: socks-tls-2
API Version:  cert-manager.io/vlalpha2
Kind:         CertificateRequest
Metadata:
  Creation Timestamp:  2019-12-03T08:49:50Z
  Generation:          1
  Owner References:
    API Version:           cert-manager.io/vlalpha2
    Block Owner Deletion:  true
    Controller:            true
    Kind:                  Certificate
    Name:                  socks-tls-2
    UID:                   8f4ad88b-ea86-4972-8980-a609b67f80c8
  Resource Version:        16818452
  Self Link:               /apis/cert-manager.io/vlalpha2/namespaces/default/certificaterequests/socks-tls-2-2550861952
  UID:                     12d89c06-c437-4fe9-a825-c33b28c7425e
Spec:
  Csr:  LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0KTUlJQ216Q0NBWE1DQVFBd0Z6RVZNQk1HQTFVRUNoTU1ZMlZ5ZEMxdFlXNWhaM1Z5TUlJQklqQU5CZ2txaGtpRwo5dzBCQVFFRkFBT0NBUThBTUlJQkNnS0NBUUVBeDJ6cTZQeXBWMS9SWFIzY3RHb2F3enAwUkJtL3Z3TWR4aXpkCnYyZzY2bytPQUN0ekNlaCtRWWNlcUppMDEvWm9JQUNEcXREMXBLZkp4M0w4cVJTclBodlpTUlhpdlV2M2FmaHEKWVFVWkkyTlR6RGJHelhtdDMvS0pJVXdtdzZqblZia0lHRm8vQStuZk9sUkpQK0pjWHVWblljQmM1L0lObG1VUgo5YjYrandZMStTdmZKcEJHZEFFOEdlN1NjWU95aUVJbmkvL214THNRN3Zld2NWcXN1NzZOQzFZbEZoR2cxcGlKC1UxM0JYUVYzaFM0KzVJVGt2R3JUVndVNFBpMVVJWG9INUR4RkxFWmRUT3R6dldRWEJ0UU9oMnRTUFNITWtDN2MKbVJBOW9PMy8rV3pmemFwb0xEcDI3aEpzb0xpN0lhL1VBc0ZobjRqa3Z3QjZIQlBZd3dJREFRQUJvQzh3TFFZSgpLblpJaHZjTkFRa09NU0F3SGpBY0JnTlZIUkVFR1RBVGdoRnJkV0psY2klNlpYSnZlSHBsWkMleWRUQU5CZ2txCmhraUc5dzBCQVFzRkFBT0NBUUVBY0Y4UTJLM1RCQ1lVZGM1dzhkcmxOR1p1SVdqNzdYV0kzZWhoN2VyalRSM1IKNFZlTGlHZXcxejNaaTg0d2dEU0k3S0wrV0RCVnE3VklJeldnb0dhRVNwZEVEU1I3YXR5bWV2S2wrdUZMSnNRZwpGZnRaSzA4ZEJNbzJNSHFnRS9PQ1R4ai9CZjJYdExoMjNEVmFjaDJXZ3orRVMxNGp4NEVWdDlJaUl5ZFoyRDBzCldkWkR5L0piUE9XTE9nK2hRZlFNQ1doVXliaWVwNnZtWnI3alZxRFZYTUV5bHZldFd4SGRBN3ZMYVB1SzhRdEwKNVZvY2JTUVlGcWRaSnZEQXNDSi9WNEVZVzNVc3VGK0EvdnZHUnp5cFYwSEhpSGRLR21WTVZCdUtLOWtTbk90MQpuRldDMGVoQ0dZTmVjWEUvRFg3bVRsVER6YUtodDVHMUJNY3drUitVQWc9PQotLS0tLUVORCBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0K
  Issuer Ref:
    Group:  cert-manager.io
    Kind:   ClusterIssuer
    Name:   letsencrypt
Status:
  Conditions:
    Last Transition Time:  2019-12-03T08:49:48Z
    Message:               Waiting on certificate issuance from order default/socks-tls-2-2550861952-2613688448: "pending"
    Reason:                Pending
    Status:                False
    Type:                  Ready
Events:                    <none>
[root@kub-master-1 ~]#
```

Проверяем order, который в статусе pending.

```
# kubectl describe order socks-tls-2-2550861952-2613688448
```

```
[root@kub-master-1 ~]# kubectl describe order socks-tls-2-2550861952-2613688448
Name:         socks-tls-2-2550861952-2613688448
Namespace:    default
Labels:       <none>
Annotations:  cert-manager.io/certificate-name: socks-tls-2
              cert-manager.io/private-key-secret-name: socks-tls-2
API Version:  acme.cert-manager.io/v1alpha2
Kind:         Order
Metadata:
  Creation Timestamp:  2019-12-03T08:49:50Z
  Generation:          1
  Owner References:
    API Version:           cert-manager.io/v1alpha2
    Block Owner Deletion:  true
    Controller:            true
    Kind:                  CertificateRequest
    Name:                  socks-tls-2-2550861952
    UID:                   12d89c06-c437-4fe9-a825-c33b28c7425e
  Resource Version:        16818455
  Self Link:               /apis/acme.cert-manager.io/v1alpha2/namespaces/default/orders/socks-tls-2-2550861952-2613688448
  UID:                     319cda93-0d0f-4ec0-93b8-fcb8e1a889d9
Spec:
  Csr:  LS0tLS1CRUdTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0KTUlJQ216Q0NBWE1DQVFBd0Z6RVZNQklHQTFVRUNoTU1ZM1Z5ZEMxdF1XNWhaM1Z5TUlJQklqQU5CZ2txaGtpRwo5dzBCQVFFRkFBT0NBUThBT
UlJQkNnS0NBUUVBeDJ6cTZQeXBWMS9SWFIzY3RHb2F3enAwUkJtL3Z3TWR4aXpkCnYyZzY2bytPQUN0ekNlaCtRWWNlcUppMDEvWm9JQUNEcXREMXBLZkp4M0w4cVJTc1BodlpTUlhpdlV2M2FmaHEKWVFVWkkyTlR6RG
JHelhtdDMvS0pJVXdtdzZqblZia01HRm8vQStuZk9sUkpQK0pjWHVWblljQmMlL0lObG1VUgo5YjYrandZMStTdmZKcEJHZEFFOEdlN1NjWU95aUVJbmkvL214THNRN3Zld2NWcXN1NzZOQzFZbEZoR2cxcG1KClUxM0J
YUVYzaFM0KzVJVGt2R3JUVndVNFBpMVVJWG9INUR4RkxFWmRUT3R6dldRWEJ0UU9oMnRTUFNITWtDN2MKbVJBOW9PMy8rV3pmemFwb0xEcDI3aEpzb0xpN01hL1VBc0ZobjRqa3Z3QjZIQlBZd3dJREFRQUJvQzh3TFFZ
SgpLblpJaHZjTkFRa09NU0F3SGpBY0JnT1ZIUkVFRlRBVGdoRnJkV0psY2klNlpYSnZlSHBsWkM1eWRUQU5CZ2txCmhraUc5dzBCQVFzRkFBT0NBUUVBY0Y4UTJLM1RCQ1lVZGM1dzhkcmxORlplSVdqNzdYV0kzZWhoN
2VyalRSM1IKNFZlTGlHZXcxejNaaTg0d2dEU0k3S0wrV0RCVnE3VklJeldnb0dhRVNwZEVEU1I3YXR5bWV2SmtDZMSlNRZwpGZnRaSzA4ZEJNbzJNSHFnRS9PQlR4ai9CZjJYdExoMjNEVmFjaDJXZ3orRVMxNGp4NE
VWdDlJaU15ZFoyRDBzC1dkWkR5L0piUE9XTE9nK2hRZlFNQldoVX1iaWVwNnZtWnI3alZxRFZYIUV5bHZldFd4SGRBN3ZMYVB1SzhRdEwKNVZvY2JTUV1GcWRaSnZEQXNDSi9WNEVZVzNVc3VGK0EvdnZHUnp5cFYwSEh
pSGRLR21WTVZCdUtLOWtTbk90MQpuRldDMGVoQ0dZTmVjWEUvRFg3bVRsVER6YUtodDVHMUJNY3drUitVQWc9PQotLS0tLUVORCBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0K
  Dns Names:
    kuber.zeroxzed.ru
  Issuer Ref:
    Group:  cert-manager.io
    Kind:   ClusterIssuer
    Name:   letsencrypt
Status:
  Authorizations:
    Challenges:
      Token:     Wl5SeIQR_PL2MEhkP8ery8IEVFhSOb9jOnIGSdoLF34
      Type:      http-01
      URL:       https://acme-v02.api.letsencrypt.org/acme/chall-v3/1525492060/55xizA
      Token:     Wl5SeIQR_PL2MEhkP8ery8IEVFhSOb9jOnIGSdoLF34
      Type:      dns-01
      URL:       https://acme-v02.api.letsencrypt.org/acme/chall-v3/1525492060/7JBzpw
      Token:     Wl5SeIQR_PL2MEhkP8ery8IEVFhSOb9jOnIGSdoLF34
      Type:      tls-alpn-01
      URL:       https://acme-v02.api.letsencrypt.org/acme/chall-v3/1525492060/2sDPnA
    Identifier:  kuber.zeroxzed.ru
    URL:         https://acme-v02.api.letsencrypt.org/acme/authz-v3/1525492060
  Finalize URL:  https://acme-v02.api.letsencrypt.org/acme/finalize/72557006/1660372142
  State:         pending
  URL:           https://acme-v02.api.letsencrypt.org/acme/order/72557006/1660372142
Events:          <none>
[root@kub-master-1 ~]#
```

Order создает Challenge, который должен быть виден после создания order в Events, но у меня уже нет инфы. Смотрю на Challendge через kubectl.

```
# kubectl get challenge
NAME                                          STATE     DOMAIN              AGE
socks-tls-2-2550861952-2613688448-2720565039   pending   kuber.zeroxzed.ru   98m
```

```
# kubectl describe Challenge socks-tls-2-2550861952-2613688448-2720565039
```

```
[root@kub-master-1 ~]# kubectl describe Challenge socks-tls-2-2550861952-2613688448-2720565039
Name:         socks-tls-2-2550861952-2613688448-2720565039
Namespace:    default
Labels:       <none>
Annotations:  <none>
API Version:  acme.cert-manager.io/v1alpha2
Kind:         Challenge
Metadata:
  Creation Timestamp:  2019-12-03T08:49:54Z
  Finalizers:
    finalizer.acme.cert-manager.io
  Generation:  1
  Owner References:
    API Version:           cert-manager.io/v1alpha2
    Block Owner Deletion:  true
    Controller:            true
    Kind:                  Order
    Name:                  socks-tls-2-2550861952-2613688448
    UID:                   319cda93-0d0f-4ec0-93b8-fcb8e1a889d9
  Resource Version:        16818484
  Self Link:               /apis/acme.cert-manager.io/v1alpha2/namespaces/default/challenges/socks-tls-2-2550861952-2613688448-2720565039
  UID:                     dbf1f2b9-459d-45f3-87ee-f80d78d31788
Spec:
  Authz URL:  https://acme-v02.api.letsencrypt.org/acme/authz-v3/1525492060
  Dns Name:   kuber.zeroxzed.ru
  Issuer Ref:
    Group:  cert-manager.io
    Kind:   ClusterIssuer
    Name:   letsencrypt
  Key:      W15SeIQR_PL2MEhkP8ery8IEVFhSOb9jOnIGSdoLF34.jtsQYZllxB7mNXIqglOsp2bJRwfdsR2o5kwHfuvsXXE
  Solver:
    http01:
      Ingress:
        Class:  nginx
  Token:        W15SeIQR_PL2MEhkP8ery8IEVFhSOb9jOnIGSdoLF34
  Type:         http-01
  URL:          https://acme-v02.api.letsencrypt.org/acme/chall-v3/1525492060/55xizA
  Wildcard:     false
Status:
  Presented:   true
  Processing:  true
  Reason:      Waiting for http-01 challenge propagation: wrong status code '404', expected '200'
  State:       pending
Events:        <none>
[root@kub-master-1 ~]#
```

А тут уже ошибка:

```
Waiting for http-01 challenge propagation: wrong status code '404', expected '200'
```

Суть ошибки ясна. Cert-manager хочет проверить url, но вместо кода ответа 200, получает 404. Смотрим, что конкретно он проверяет. Для этого смотрим список ingress в кластере.

```
# kubectl get ingress
NAME                         HOSTS               ADDRESS     PORTS     AGE
cm-acme-http-solver-4rq89    kuber.zeroxzed.ru   10.1.4.39   80        101m
ingress-socks-tls            kuber.zeroxzed.ru   10.1.4.39   80, 443   134m
```

Ингресс cm-acme-http-solver-4rq89 поднял временно cert-manager, чтобы проверить домен. Смотрим внимательно на этот ingress.

```
# kubectl describe ingress cm-acme-http-solver-4rq89
```

```
[root@kub-master-1 ~]# kubectl describe ingress cm-acme-http-solver-4rq89
Name:             cm-acme-http-solver-4rq89
Namespace:        default
Address:          10.1.4.39
Default backend:  default-http-backend:80 (<none>)
Rules:
  Host               Path  Backends
  ----               ----  --------
  kuber.zeroxzed.ru
                     /.well-known/acme-challenge/W15SeIQR_PL2MEhkP8ery8IEVFhSOb9jOnIGSdoLF34   cm-acme-http-solver-2d6mm:8089 (10.233.68.98:8089)
Annotations:
  kubernetes.io/ingress.class:                        nginx
  nginx.ingress.kubernetes.io/whitelist-source-range:  0.0.0.0/0,::/0
Events:                                               <none>
[root@kub-master-1 ~]#
```


Указанный url должен успешно работать как снаружи, так и изнутри кластера. Без этого cert-manager и lets encrypt не смогут подтвердить сертификат.

Сам выпущенный сертификат можно посмотреть в секрете, в котором он хранится (формат base64).

```
# kubectl get secret socks-tls-2 -o yaml
```

На этом по Cert-Manager в кластере Kubernetes все. Надеюсь, у меня получилось понятно объяснить и показать как с ним работать.

Логи и конфиг Ingress Controller

Для того, чтобы дебажить работу ingress controller, надо как минимум иметь доступ к его логам и файлу конфигурации. По своей сути это просто docker образ с nginx внутри. Посмотреть логи можно следующим образом.

```
# kubectl logs ingress-nginx-controller-twwpb -n ingress-nginx
```

Перед этим посмотрите, какое имя и в каком namespace работает интересующий вас pod с ingress controller на борту. По-умолчанию он в ingress-nginx будет запущен.

```
# kubectl get pod -A | grep ingress-nginx
```

Логи можно направить куда-нибудь в файл, чтобы было удобно читать и анализировать. Конфигурацию nginx внутри ingress controller можно посмотреть следующим образом. Ее лучше тоже сразу в файл выгружать, чтобы удобнее было читать.

```
# kubectl exec -it ingress-nginx-controller-twwpb -n ingress-nginx cat /etc/nginx/nginx.conf > ~/ingress-nginx.conf
```

Посмотрите на конфиг. Там по сути обычный nginx. Можете разобраться в проблеме, если считаете, что что-то работает не так, как вы ожидаете.

Можно зайти в сам контейнер с ingress controller и там посмотреть все сертификаты в исходном виде. Например вот так.

```
# kubectl exec -it ingress-nginx-controller-twwpb -n ingress-nginx cat /etc/ingress-controller/ssl/default-fake-
certificate.pem > ~/default-fake-certificate.pem
```

Это дефолтный сапоподписанный сертификат, который выпускает ingress, когда вы включаете в нем tls. Если у домена включен tls, но по какой-то причине не работает его сертификат, будет подставлен дефолтный.

Конфигурация Nginx для Ingress Controller

Как я уже говорил, в моем примере под капотом Ingress контроллера работает обычный Nginx с помощью специально подготовленного DaemonSet в неймспейсе ingress-nginx.

```
# kubectl get all -n ingress-nginx
NAME                                   READY   STATUS    RESTARTS   AGE
```

```
pod/ingress-nginx-controller-twwpb   1/1     Running   4          75d
NAME                                       DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
AGE
daemonset.apps/ingress-nginx-controller   1         1         1       1            1           node-
role.kubernetes.io/ingress=true   75d
```

Для того, чтобы задать глобальные параметры для ingress можно использовать configmap. Давайте для примера изменим некоторые глобальные параметры ingress. Напомню, что он запущен в namespace ingress-nginx. Добавим туда configmap для ingress-nginx и применим его.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
data:
  proxy-connect-timeout: "222"
  proxy-read-timeout: "222"
  proxy-send-timeout: "222"
```

```
# kubectl apply -f nginx-config.yaml
configmap/ingress-nginx configured
```

Проверяем конфигурацию ingress, выгрузив конфиг nginx в файл.

```
# kubectl exec -it ingress-nginx-controller-twwpb -n ingress-nginx cat /etc/nginx/nginx.conf > ~/ingress-nginx.conf
```

Во всех виртуальных хостах должны измениться параметры:

```
proxy_connect_timeout 222s;
proxy_send_timeout 222s;
proxy_read_timeout 222s;
```

Так же конфигурацию nginx можно задавать с помощью аннотаций. Возьмем для примера тот же магазин носок и его ingress манифест. Добавим туда аннотации.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-socks-tls
  annotations:
    kubernetes.io/ingress.class: "nginx"
    cert-manager.io/cluster-issuer: "letsencrypt"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "301"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "301"
spec:
  rules:
  - host: kuber.zeroxzed.ru
    http:
      paths:
      - backend:
          serviceName: front-end
          servicePort: 80
  tls:
  - hosts:
    - kuber.zeroxzed.ru
    secretName: socks-tls-2
```

Применяем.

```
# kubectl apply -f ingress-sock-tls.yaml
ingress.extensions/ingress-socks-tls configured
```

Проверяем конфигурацию nginx. В виртуальном хосте из манифеста ingress должны установиться указанные параметры nginx.

```
proxy_connect_timeout 222s;
proxy_send_timeout 301s;
proxy_read_timeout 301s;
```

Если что-то пойдет не так, смотрите логи пода с Ingress Controller. Там будут видны ошибки. Вот пример ошибок, когда я в манифестах пытался сразу же поставить значения с секундами.

```
W1203 13:11:07.756139       6 configmap.go:339] unexpected error merging defaults: 2 error(s) decoding:

* cannot parse 'proxy-connect-timeout' as int: strconv.ParseInt: parsing "20s": invalid syntax
* cannot parse 'proxy-read-timeout' as int: strconv.ParseInt: parsing "20s": invalid syntax
```

Секунды проставляются автоматом при передаче значений в nginx. В манифестах их указывать не нужно. Я так понимаю, это особенность установки ingress controller через Kubespray. Если будете ставить через Helm или как-то еще, то это может работать по-другому. Например, во многих примерах в интернете я видел, что параметры nginx для ingress controller в манифестах указывают в виде 20s, 30m и т.д.

Заключение

> Не понравилась статья и хочешь научить меня администрировать? Пожалуйста, я люблю учиться. Комментарии в твоем распоряжении.

Расскажи, как сделать правильно!

На этом рассказ про Ingress завершаю. Постарался описать все основные моменты, с которыми сталкиваешься при начальной настройке Ingress Controller. Этой базы достаточно, чтобы опубликовать реальный интернет ресурс в кластере Kubernetes.

Другие статьи по k8s:

- Установка kubernetes,
- Работа с кластером, основные сущности,
- Дисковые тома,
- Работа с Helm.

Напоминаю, что эта статья является частью единого и последовательного цикла статей по настройке и управлению кластером Kubernetes.

Сдавайте вступительный тест по и присоединяйтесь к новому набору!.

Помогла статья? Подписывайся на telegram канал автора

Анонсы всех статей, плюс много другой полезной и интересной информации, которая не попадает на сайт.