12. Работа с Gateway API (Локальное использование)

12.1. Введение в Gateway API

Gateway API — это современная альтернатива Ingress, предоставляющая более гибкие и расширяемые возможности маршрутизации сетевого трафика внутри Kubernetes. В этой практике мы настроим Gateway API для локального тестирования без использования облачных сервисов.

12.2. Установка Gateway API

1. Установим CRD (Custom Resource Definitions) для Gateway API:

```
kubectl apply -f https://github.com/kubernetes-sigs/gateway-
api/releases/download/v0.8.0/standard-install.yaml
```

2. Проверяем установку:

```
kubectl get crds | grep gateway
```

12.3. Установка контроллера Gateway

1. Установим контроллер `gateway-api-admission-controller` для работы с Gateway API:

```
kubectl apply -f https://github.com/kubernetes-sigs/gateway-
api/releases/download/v0.8.0/admission-webhook.yaml
```

2. Проверяем запущенные поды:

```
kubectl get pods -n gateway-api
```

12.4. Создание приложения и сервиса

1. Создаем файл `deploy-gateway.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: echo-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: echo
  template:
    metadata:
      labels:
        app: echo
    spec:
      containers:
      - name: echo
        image: hashicorp/http-echo  # Используется для тестового ответа
        args:
        - "-text=Hello, Gateway!"
        ports:
```

```
        - containerPort: 5678
---
apiVersion: v1
kind: Service
metadata:
  name: echo-service
spec:
  selector:
    app: echo
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5678
  type: ClusterIP
```

2. Применяем манифест:

```
kubectl apply -f deploy-gateway.yaml
kubectl get svc
```

12.5. Создание Gateway

1. Создаем файл `gateway.yaml`:

```
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: Gateway
metadata:
  name: echo-gateway
spec:
  gatewayClassName: nginx  # Определяет класс Gateway, который используется
для настройки сетевого шлюза. Обычно указывает на конкретный контроллер,
например, Nginx.
  listeners:
  - protocol: HTTP  # Протокол, который будет обслуживать Gateway (HTTP или
HTTPS).
    port: 80  # Порт, на котором Gateway будет принимать входящие запросы.
    name: http  # Имя слушателя, помогающее идентифицировать правила
маршрутизации.
    allowedRoutes:
      namespaces:
        from: Same  # Ограничивает маршруты только текущим пространством
имен, что обеспечивает дополнительную безопасность, разрешая подключение
только сервисов из одного неймспейса.  # Ограничивает маршруты только текущим
пространством имен, что обеспечивает дополнительную безопасность, разрешая
подключение только сервисов из одного неймспейса  # Ограничивает маршруты
только текущим пространством имен
```

2. Применяем манифест:

```
kubectl apply -f gateway.yaml
kubectl get gateway
```

12.6. Создание HTTPRoute

1. Создаем файл `httproute.yaml`:

```
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: HTTPRoute
metadata:
  name: echo-route
```

```
spec:
  parentRefs:
  - name: echo-gateway  # Связывает маршрут с Gateway, который будет
обрабатывать входящие запросы
  rules:
  - matches:
    - path:
        type: PathPrefix  # Тип маршрутизации, который сопоставляет все
запросы с указанным префиксом пути
        value: /  # В данном примере обрабатываются все пути, начиная с /
    backendRefs:
    - name: echo-service  # Название сервиса, на который будут перенаправлены
запросы
      port: 80  # Порт, на котором сервис принимает запросы  # Связывает
маршрут с сервисом echo-service на порту 80
```

2. Применяем манифест:

```
kubectl apply -f httproute.yaml
kubectl get httproute
```

12.7. Настройка hosts файла

1. Открываем файл `/etc/hosts` на локальном компьютере и добавляем строку:

```
127.0.0.1 gateway.local
```

12.8. HTTPS с самоподписанным сертификатом

1. Создаем самоподписанный сертификат:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout tls.key -out
tls.crt -subj "/CN=gateway.local/O=gateway.local"
# Команда `openssl req` используется для создания самоподписанного
сертификата.
# - `-x509` указывает, что создается самоподписанный сертификат X.509.
# - `-nodes` означает, что приватный ключ не будет защищен паролем.
# - `-days 365` устанавливает срок действия сертификата на 365 дней.
# - `-newkey rsa:2048` создает новый ключ RSA длиной 2048 бит.
# - `-subj` определяет данные субъекта, включая CN (Common Name) — имя хоста,
для которого создается сертификат.
```

2. Создаем Secret с сертификатом:

```
kubectl create secret tls gateway-tls --cert=tls.crt --key=tls.key
```

3. Обновляем файл `gateway.yaml` для использования HTTPS:

```
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: Gateway
metadata:
  name: echo-gateway-https
spec:
  gatewayClassName: nginx
  listeners:
  - protocol: HTTPS  # Указывает, что Gateway будет принимать HTTPS-трафик.
    port: 443  # Стандартный порт для HTTPS-соединений.
    name: https  # Имя слушателя для идентификации.
    tls:
```

```
      certificateRefs:
      - name: gateway-tls  # Ссылается на Secret, содержащий сертификаты для
шифрования трафика.
    allowedRoutes:
      namespaces:
        from: Same  # Ограничивает маршруты только текущим пространством
имен.

# TLS обеспечивает шифрование трафика между клиентом и сервером, что
предотвращает перехват данных.
# Secret используется для хранения сертификатов, чтобы Gateway API мог
устанавливать защищенные соединения.
```

4. Применяем манифест:

```
kubectl apply -f gateway.yaml
kubectl get gateway
```

5. Проверяем доступ через curl:

```
curl -k https://gateway.local:8080
```

12.9. Проверка работы

1. Перенаправляем порт для тестирования локально:

```
kubectl port-forward svc/echo-service 8080:80
```

2. Проверяем доступ через `curl`:

```
curl http://gateway.local:8080
curl -k https://gateway.local:8080
```

Ожидается ответ: `Hello, Gateway!`

12.10. Удаление ресурсов

1. Удаляем все созданные ресурсы:

```
kubectl delete -f deploy-gateway.yaml
kubectl delete -f gateway.yaml
kubectl delete -f httproute.yaml
kubectl delete secret gateway-tls
kubectl delete crd gateways.gateway.networking.k8s.io
httproutes.gateway.networking.k8s.io
```
