作者：李晓辉

联系方式：

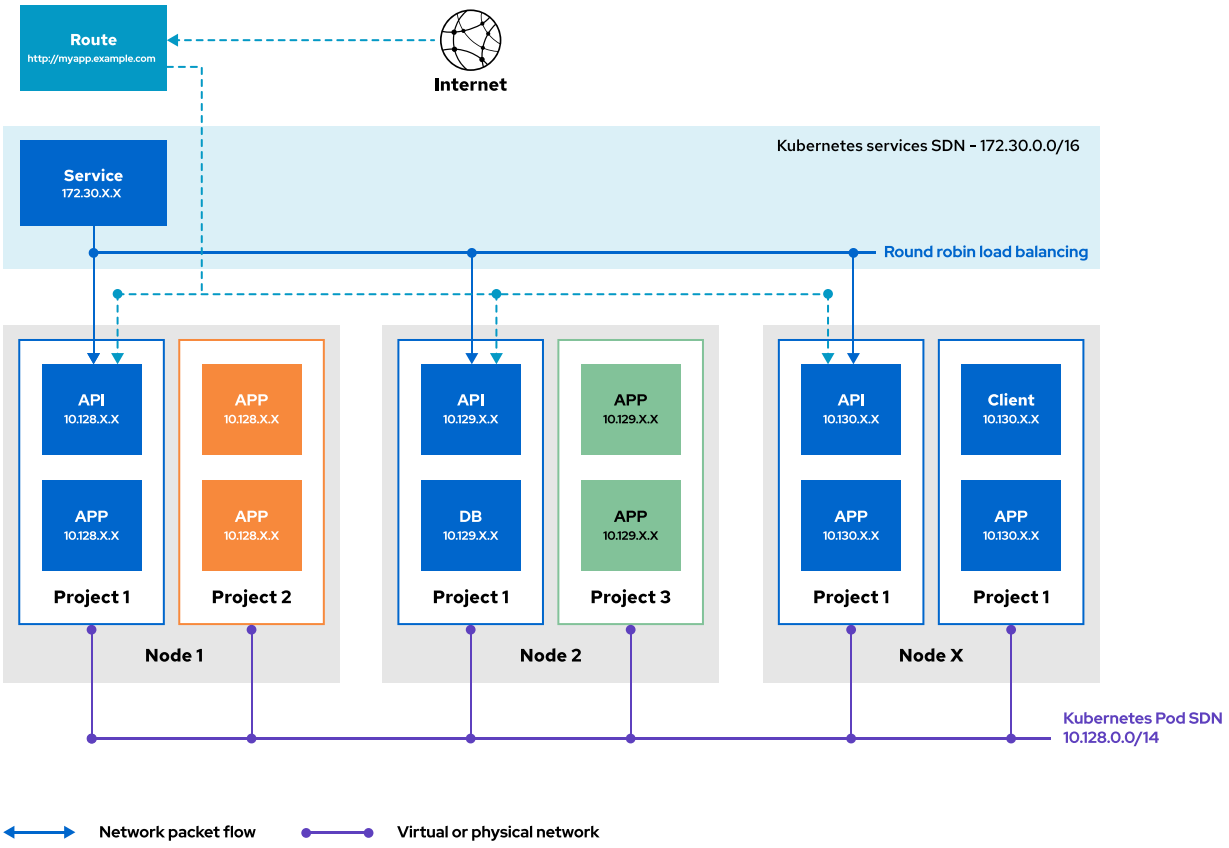
1. 微信：Lxh_Chat

2. 邮箱：939958092@qq.com

OpenShift为了让你的应用被外面的网络找到，它有好几种招数，比如能公开 HTTP、HTTPS 流量，还能搞定 TCP 应用，甚至其他非 TCP 的流量也不在话下。

有些招数是靠服务类型，像 `NodePort` 或者负载平衡器这种；还有些是靠自己的 API 资源，比如 `Ingress` 和 `Route`。特别是 OpenShift 的路由，特别方便，它能让你的应用通过一个独一无二的公开主机名被访问。这背后靠的是路由器插件，能把从公共 IP 来的流量精准地重定向到对应的 pod 上。

下图显示了路由如何公开在集群中作为 pod 运行的应用：



保护route的方案一般有三种，主要用前两种

边缘终止

简单来说，就是路由器先“拆开”加密的流量，然后再把流量转发给后端的 Pod。因为路由器负责处理 TLS 证书，所以你得把证书配置到路由里。不然的话，OpenShift 会用自己的证书来搞定。不过，一旦流量过了路由器，到 Pod 的这段路就是明文传输了，不会被加密。

直通

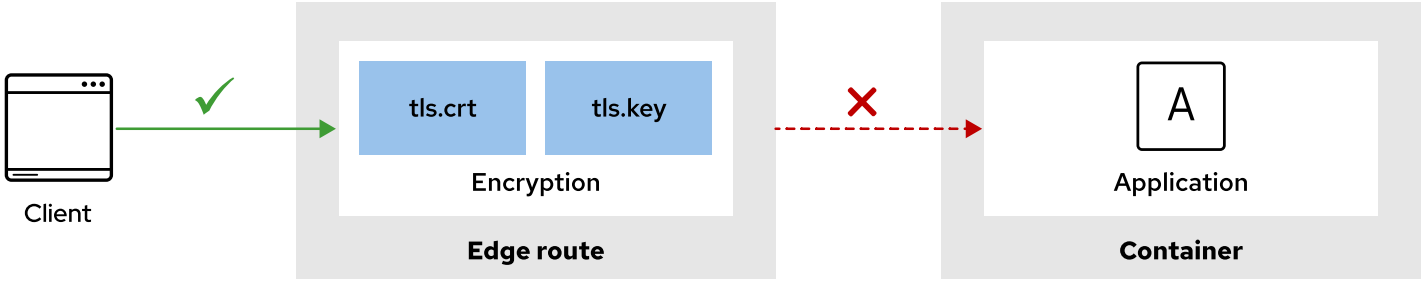
这种模式下，加密的流量一路直达目标 Pod，路由器不会去“拆开”它。所以，应用自己得搞定证书的事儿。如果你的应用需要和客户端互相验证身份，那直通是目前唯一的选择。

再加密

这可以看作是边缘终止的“升级版”。路由器先是像边缘终止那样终止 TLS，然后还会重新加密从路由器到 Pod 的连接，而且这个连接可以用不同的证书。这样一来，整个连接过程都是加密的，哪怕是在内部网络里。路由器还会通过健康检查来确认 Pod 的身份。

边缘卸载

在边缘模式中使用路由时，客户端和路由器之间的流量会加密，但路由器和应用之间的流量则不会加密：



这里需要用到HTTPS证书，我们在workstation上生成证书

证书生成

先生成root证书

```
[student@workstation ~]$ su -
```

```
openssl genrsa -out /etc/pki/tls/private/xiaohuiroot.key 4096
openssl req -x509 -new -nodes -sha512 -days 3650 -subj "/C=CN/ST=Shanghai/L=Shanghai/O=Comp
-key /etc/pki/tls/private/xiaohuiroot.key \
-out /etc/pki/ca-trust/source/anchors/xiaohuiroot.crt
```

再生成证书请求，本次申请为*.apps.ocp4.example.com

```
openssl genrsa -out /etc/pki/tls/private/xiaohui.cn.key 4096
openssl req -sha512 -new \
-subj "/C=CN/ST=Shanghai/L=Shanghai/O=Company/OU=SH/CN=*.apps.ocp4.example.com" \
-key /etc/pki/tls/private/xiaohui.cn.key \
-out xiaohui.cn.csr
```

签发证书

```
openssl x509 -req -in xiaohui.cn.csr \
-CA /etc/pki/ca-trust/source/anchors/xiaohuiroot.crt \
-CAkey /etc/pki/tls/private/xiaohuiroot.key -CAcreateserial \
-out /etc/pki/tls/certs/xiaohui.cn.crt \
-days 3650
```

```
chmod +r /etc/pki/tls/certs/xiaohui.cn.crt
chmod +r /etc/pki/tls/private/xiaohui.cn.key
```

本地信任根证书

```
update-ca-trust
```

创建明文服务

先创建一个不加密的后端服务，此服务名为no-tls并工作在80端口

```
cat > no-tls.yml <<-EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-http
  labels:
    app: todo-http
    name: todo-http
spec:
  replicas: 1
  selector:
    matchLabels:
      app: todo-http
      name: todo-http
  template:
    metadata:
      labels:
        app: todo-http
        name: todo-http
    spec:
      containers:
      - resources:
          limits:
            cpu: '0.5'
        image: registry.ocp4.example.com:8443/redhattraining/todo-angular:v1.1
        name: todo-http
        ports:
        - containerPort: 8080
          name: todo-http
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: todo-http
    name: todo-http
  name: no-tls
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    name: todo-http
EOF
```

```
[root@workstation ~]# oc create -f no-tls.yml
```

将服务暴露出来

```
oc expose svc no-tls --hostname no-tls.apps.ocp4.example.com
```

确认可以用不加密的方式访问

```
[student@workstation ~]$ oc get route todo-http
NAME        HOST/PORT                          PATH   SERVICES    PORT   TERMINATION   WILDC
todo-http   todo-http.apps.ocp4.example.com           todo-http   8080                 None
[student@workstation ~]$ curl -s no-tls.apps.ocp4.example.com | grep -i todo
<html lang="en" ng-app="todoItemsApp" ng-controller="appCtl">
    <title>ToDo app</title>
    <script type="text/javascript" src="assets/js/app/domain/todoitems.js"></script>
        <a class="navbar-brand" href="/">ToDo App</a>
```

创建TLS边缘卸载服务

这次创建了一个主机为tls-only.apps.ocp4.example.com的服务地址

```
oc create route edge --service no-tls --hostname tls-only.apps.ocp4.example.com --key /etc/
```

访问一下看看

```
curl -s https://tls-only.apps.ocp4.example.com | grep -i todo
<html lang="en" ng-app="todoItemsApp" ng-controller="appCtl">
    <title>ToDo app</title>
    <script type="text/javascript" src="assets/js/app/domain/todoitems.js"></script>
        <a class="navbar-brand" href="/">ToDo App</a>
```

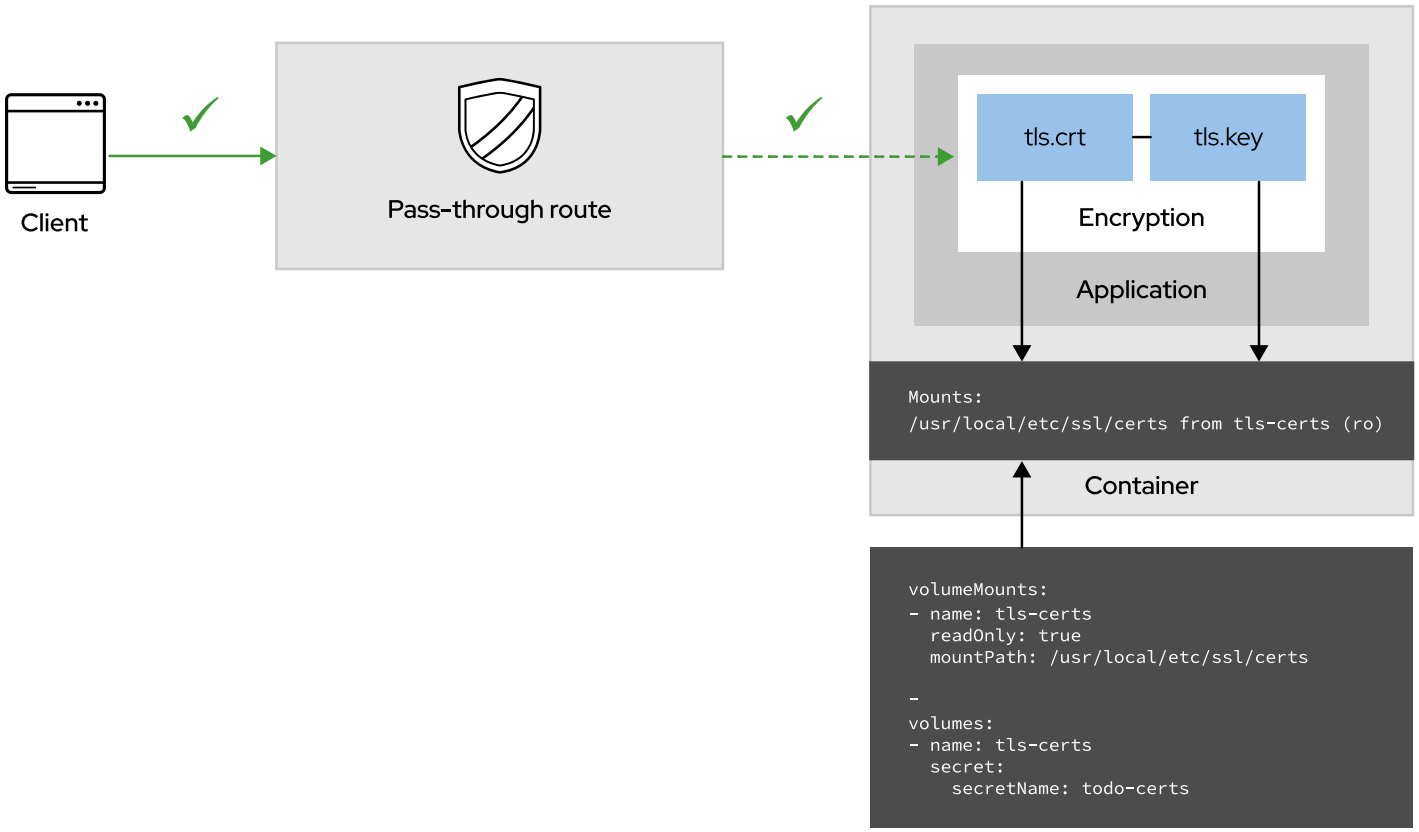
直通路由

直通路由是个很安全的选择，因为应用自己会“亮出”它的 TLS 证书。这样一来，从客户端到应用之间的流量都是加密的，不用担心被人偷窥。

提供证书的最好方法

搞定证书的最好办法是用 OpenShift 的 TLS 机密。你可以把机密通过挂载点“塞进”容器里，应用就能直接用上这些证书了，很方便。

下图显示了如何在容器中挂载 secret 资源。然后，应用可以访问你的证书。



创建tls机密

再生成证书请求，本次申请为tls-pass.apps.ocp4.example.com

我这里复用代码，所以会覆盖前面的证书和私钥，如果需要请备份前面的

```
su -
```

```
openssl genrsa -out /etc/pki/tls/private/xiaohui.cn.key 4096
openssl req -sha512 -new \
-subj "/C=CN/ST=Shanghai/L=Shanghai/O=Company/OU=SH/CN=tls-pass.apps.ocp4.example.com" \
-key /etc/pki/tls/private/xiaohui.cn.key \
-out xiaohui.cn.csr
```

签发证书

```
openssl x509 -req -in xiaohui.cn.csr \
-CA /etc/pki/ca-trust/source/anchors/xiaohuiroot.crt \
-CAkey /etc/pki/tls/private/xiaohuiroot.key -CAcreateserial \
-out /etc/pki/tls/certs/xiaohui.cn.crt \
-days 3650
```

```
chmod +r /etc/pki/tls/certs/xiaohui.cn.crt
chmod +r /etc/pki/tls/private/xiaohui.cn.key
```

```
oc create secret tls tls-only --key /etc/pki/tls/private/xiaohui.cn.key --cert /etc/pki/tls
```

创建加密的后端服务

创建一个名为todo-https-pass且工作在8443和80的端口上的服务

在这个服务中，我们引用了上面的tls机密

```
cat > tls-only-pass.yml <<-EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-https
  labels:
    app: todo-https
    name: todo-https
spec:
  replicas: 1
  selector:
    matchLabels:
      app: todo-https
      name: todo-https
  template:
    metadata:
      labels:
        app: todo-https
        name: todo-https
    spec:
      containers:
      - resources:
          limits:
            cpu: '0.5'
        image: registry.ocp4.example.com:8443/redhattraining/todo-angular:v1.2
        name: todo-https
        ports:
        - containerPort: 8080
          name: todo-http
        - containerPort: 8443
          name: todo-https
        volumeMounts:
        - name: tls-only
          readOnly: true
          mountPath: /usr/local/etc/ssl/certs
      resources:
        limits:
          memory: 64Mi
      volumes:
      - name: tls-only
        secret:
          secretName: tls-only
---
apiVersion: v1
kind: Service
metadata:
  labels:
```

```
    app: todo-https
    name: todo-https
  name: todo-https-pass
spec:
  ports:
  - name: https
    port: 8443
    protocol: TCP
    targetPort: 8443
  - name: http
    port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    name: todo-https
EOF
```

创建出服务

```
oc create -f tls-only-pass.yml
```

看看pod是否正常运行

```
[student@workstation ~]$ oc get -f tls-only-pass.yml
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/todo-https    0/1     1            0           7s

NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
service/todo-https-pass    ClusterIP   172.30.64.247   <none>        8443/TCP,80/TCP   7s
[student@workstation ~]$ oc get pod
NAME                           READY   STATUS        RESTARTS   AGE
todo-https-69b956b947-5zp89    1/1     Running       0          32s
```

看看证书是否如期挂载到pod中

```
[student@workstation ~]$ oc describe pod todo-https-69b956b947-5zp89 | grep -A2 Mounts
    Mounts:
      /usr/local/etc/ssl/certs from tls-only (ro)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-7nxs7 (ro)
```

创建直通安全路由

```
[student@workstation ~]$ oc create route passthrough tls-pass --service todo-https-pass --p
route.route.openshift.io/tls-pass created
[student@workstation ~]$ oc get route
NAME       HOST/PORT                         PATH   SERVICES          PORT   TERMINATION   W
tls-pass   tls-pass.apps.ocp4.example.com           todo-https-pass   8443   passthrough   N
```

访问https看看是否成功

```
[student@workstation ~]$ curl -vv -I  https://tls-pass.apps.ocp4.example.com
*   Trying 192.168.50.254:443...
* Connected to tls-pass.apps.ocp4.example.com (192.168.50.254) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
*  CAfile: /etc/pki/tls/certs/ca-bundle.crt
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS header, Finished (20):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS header, Finished (20):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* ALPN, server accepted to use h2
* Server certificate:
*  subject: C=CN; ST=Shanghai; L=Shanghai; O=Company; OU=SH; CN=tls-pass.apps.ocp4.example.
*  start date: Dec 19 14:02:06 2024 GMT
*  expire date: Dec 17 14:02:06 2034 GMT
*  common name: tls-pass.apps.ocp4.example.com (matched)
*  issuer: C=CN; ST=Shanghai; L=Shanghai; O=Company; OU=SH; CN=*.apps.ocp4.example.com
*  SSL certificate verify ok.
```