**host-based routing** and **path-based routing in OpenShift using Routes, with YAML examples**   
  
**First, What's the Difference?**   
Type of Routing Example URL Meaning   
Host-based [http://foo.apps.example.com](http://foo.apps.example.com/) Route based on the hostname   
Path-based <http://apps.example.com/foo> Route based on the URL path (/foo)   
  
**1. Host-Based Routing in OpenShift**   
This is the default behavior in OpenShift routes.   
YAML Example (Host-Based)   
  
apiVersion: [route.openshift.io/v1](https://web.telegram.org/a/route.openshift.io/v1)   
kind: Route   
metadata:   
name: foo-route   
spec:   
host: [foo.apps.example.com](https://web.telegram.org/a/foo.apps.example.com)      # Hostname-based   
to:   
kind: Service   
name: foo-service   
port:   
targetPort: 8080   
  
What It Means:   
The app is exposed at: [http://foo.apps.example.com](http://foo.apps.example.com/)   
The router checks the Host header.   
If it matches [foo.apps.example.com](https://web.telegram.org/a/foo.apps.example.com), it forwards to foo-service.   
  
**2.Path-Based Routing in OpenShift**   
This allows you to expose multiple apps under a single domain, using paths.   
  
For example:   
/foo → App1   
/bar → App2   
  
YAML Example (Path-Based)   
apiVersion: [route.openshift.io/v1](https://web.telegram.org/a/route.openshift.io/v1)   
kind: Route   
metadata:   
name: foo-path-route   
spec:   
host: [apps.example.com](https://web.telegram.org/a/apps.example.com)           # Common host   
path: /foo # Unique path   
to:   
kind: Service   
name: foo-service   
port:   
targetPort: 8080   
  
apiVersion: [route.openshift.io/v1](https://web.telegram.org/a/route.openshift.io/v1)   
kind: Route   
metadata:   
name: bar-path-route   
spec:   
host: [apps.example.com](https://web.telegram.org/a/apps.example.com)           # Same host   
path: /bar # Different path   
to:   
kind: Service   
name: bar-service   
port:   
targetPort: 8080   
  
What It Means:   
Requests to <http://apps.example.com/foo> go to foo-service   
Requests to <http://apps.example.com/bar> go to bar-service   
OpenShift HAProxy Router checks the path prefix and routes accordingly.   
  
How Routing Actually Works Behind the Scenes   
The Router receives the incoming request.   
It checks:   
First: Does the Host match?   
Then (if path is given): Does the Path match?   
Based on that, it forwards the request to the correct Service.   
Service routes to the Pod.   
  
**Use Cases**   
Routing Type When to Use   
Host-Based When each app should have its own subdomain (e.g., [api.example.com](https://web.telegram.org/a/api.example.com))   
Path-Based When you want multiple apps under one domain ([example.com/app1](https://web.telegram.org/a/example.com/app1))   
  
  
**how the built-in HAProxy component acts both as:**   
Ingress Controller (for Ingress objects)   
Router (for Route objects)   
  
**As a Router (for Route objects)   
You create:**  
kind: Route   
spec:   
host: [myapp.apps.example.com](https://web.telegram.org/a/myapp.apps.example.com)   
to:   
kind: Service   
name: myapp-service   
  
**What happens:**   
The HAProxy Router watches for Route objects.   
It sees [myapp.apps.example.com](https://web.telegram.org/a/myapp.apps.example.com) → myapp-service   
It automatically configures itself   
When a user opens the browser and hits [myapp.apps.example.com](https://web.telegram.org/a/myapp.apps.example.com), the router:   
Accepts the traffic   
Forwards it to the correct service/pods   
This is the standard OpenShift way.   
  
As an Ingress Controller (for Ingress objects)   
OpenShift also supports standard Kubernetes Ingress objects, and here's the trick:   
OpenShift uses the same HAProxy router to also understand Ingress objects, as long as:   
The Ingress Controller is configured to support them (OpenShift enables this by default)   
kind: Ingress   
spec:   
rules:   
- host: [myapp.apps.example.com](https://web.telegram.org/a/myapp.apps.example.com)   
http:   
paths:   
- path: /foo   
backend:   
service:   
name: foo-service   
  
**What happens:**   
The same HAProxy router sees this Ingress object   
Configures itself for path /foo → foo-service   
Routes traffic just like it does for Route objects

**So in OpenShift, you don’t need to install any Ingress Controller — it’s built in!**   
  
**Traffic Flow in Simple Words**   
Client (Browser hits app URL)   
↓   
OpenShift HAProxy Router (Ingress Controller + Route Router)   
↓   
Decides based on:   
→ Host ([example.com](https://web.telegram.org/a/example.com))   
→ Path (/foo)   
↓   
Routes to → Kubernetes Service   
↓   
Service load balances to → Pods   
  
  
Host-based and Path-based routing using Ingress objects in OpenShift,   
Remember: OpenShift uses HAProxy Router as a built-in Ingress Controller, so you do not need to install anything extra.   
  
**Host-Based Routing (Ingress)**   
This means different domains (hosts) point to different services.   
  
Example YAML: Host-based Routing   
  
apiVersion: [networking.k8s.io/v1](https://web.telegram.org/a/networking.k8s.io/v1)   
kind: Ingress   
metadata:   
name: host-routing-example   
annotations:   
[nginx.ingress.kubernetes.io/rewrite-target](https://web.telegram.org/a/nginx.ingress.kubernetes.io/rewrite-target): /   
spec:   
rules:   
- host: [app1.example.com](https://web.telegram.org/a/app1.example.com)   
http:   
paths:   
- path: /   
pathType: Prefix   
backend:   
service:   
name: app1-service   
port:   
number: 80   
  
- host: [app2.example.com](https://web.telegram.org/a/app2.example.com)   
http:   
paths:   
- path: /   
pathType: Prefix   
backend:   
service:   
name: app2-service   
port:   
number: 80   
  
How it works:   
If someone hits [http://app1.example.com](http://app1.example.com/), it goes to app1-service   
If someone hits [http://app2.example.com](http://app2.example.com/), it goes to app2-service   
  
**Path-Based Routing (Ingress**)   
This means same domain, but different URL paths go to different services.   
Example YAML: Path-based Routing   
apiVersion: [networking.k8s.io/v1](https://web.telegram.org/a/networking.k8s.io/v1)   
kind: Ingress   
metadata:   
name: path-routing-example   
annotations:   
[nginx.ingress.kubernetes.io/rewrite-target](https://web.telegram.org/a/nginx.ingress.kubernetes.io/rewrite-target): /   
spec:   
rules:   
- host: [myapp.example.com](https://web.telegram.org/a/myapp.example.com)   
http:   
paths:   
- path: /app1   
pathType: Prefix   
backend:   
service:   
name: app1-service   
port:   
number: 80   
  
- path: /app2   
pathType: Prefix   
backend:   
service:   
name: app2-service   
port:   
number: 80   
  
How it works:   
<http://myapp.example.com/app1> → app1-service   
<http://myapp.example.com/app2> → app2-service   
  
Traffic Flow Summary (Same as Route)   
  
User hits browser: <http://app1.example.com/app1>   
↓   
HAProxy Router (Ingress Controller in OpenShift)   
↓   
✓ Host: [app1.example.com](https://web.telegram.org/a/app1.example.com)   
✓ Path: /app1   
↓   
Service: app1-service   
↓   
Pods with label: app1   
  
NOTE on Domain:   
OpenShift routes or Ingress will work only if:   
The external DNS (or /etc/hosts) resolves:   
[app1.example.com](https://web.telegram.org/a/app1.example.com) → OpenShift Router IP   
You can test it by editing your /etc/hosts (for testing only):   
<router-public-ip> [app1.example.com](https://web.telegram.org/a/app1.example.com)   
<router-public-ip> [myapp.example.com](https://web.telegram.org/a/myapp.example.com)