Session stickiness in OpenShift

2018-05-15

How OpenShift and Kubernetes handle session stickiness via HAProxy cookie-based routing.

Many enterprise applications are not yet cloud-ready or designed as microservices. For these, session stickiness is a common requirement. Let me explain how OpenShift Container Platform Enterprise (= OCP) and OKD can help you solve this.

“Management summary”

The most common question I have heard from customers, both then and now, is:

> Can I use Session stickiness in OpenShift and Kubernetes?

And the clear answer is:

> Yes.

That said, you should consider getting rid of this pattern and using a **shared** session store instead.

History

The first version of OpenShift v3 was released on 24 June 2015: OpenShift Enterprise 3: Evolving PaaS for the Future. It already included cookie-based stickiness via the following snippets in the haproxy-config.template:

```
https://github.com/openshift/origin/blob/release-1.2/images/router/
haproxy/conf/haproxy-config.template#L210-L219

  {{ if (eq $cfg.TLSTermination "") }}
    cookie OPENSHIFT_{{$cfgIdx}}_SERVERID insert indirect nocache
httponly
  {{ else }}
    cookie OPENSHIFT_EDGE_{{$cfgIdx}}_SERVERID insert indirect nocache
httponly secure
  {{ end }}
  http-request set-header Forwarded
for=%[src];host=%[req.hdr(host)];proto=%[req.hdr(X-Forwarded-Proto)]
                {{ range $idx, $endpoint := endpointsForAlias $cfg
$serviceUnit }}
  server {{$endpoint.IdHash}} {{$endpoint.IP}}:{{$endpoint.Port}} check
```

```
inter 5000ms cookie {{$endpoint.IdHash}}
                {{ end }}
{{ end }}

https://github.com/openshift/origin/blob/release-1.2/images/router/
haproxy/conf/haproxy-config.template#L237-L241
  cookie OPENSHIFT_REENCRYPT_{{$cfgIdx}}_SERVERID insert indirect nocache
httponly secure
                {{ range $idx, $endpoint := endpointsForAlias $cfg
$serviceUnit }}
  server {{$endpoint.IdHash}} {{$endpoint.IP}}:{{$endpoint.Port}} ssl
check inter 5000ms verify required ca-file {{ $workingDir }}/cacerts/
{{$cfgIdx}}.pem cookie {{$endpoint.IdHash}}
                {{ end }}
{{ end }}
```

These Go template snippets instruct HAProxy to add the cookie `OPENSHIFT_{{$cfgIdx}}_SERVERID`, `OPENSHIFT_EDGE_{{$cfgIdx}}_SERVERID`, or `OPENSHIFT_REENCRYPT_{{$cfgIdx}}_SERVERID` to the client response, and strip it again before forwarding the request to the backend server.

A more detailed description can be found in the HAProxy documentation: cookie keyword.

This has been the default configuration since the beginning. Since OpenShift 3.5, this behavior can be disabled via the route annotation `haproxy.router.openshift.io/disable_cookies`.

10,000-foot view

Let's look at how a request travels to a Pod's IP address — also known as a Kubernetes Endpoint (OCP Pod Doc, K8S Pod Doc).

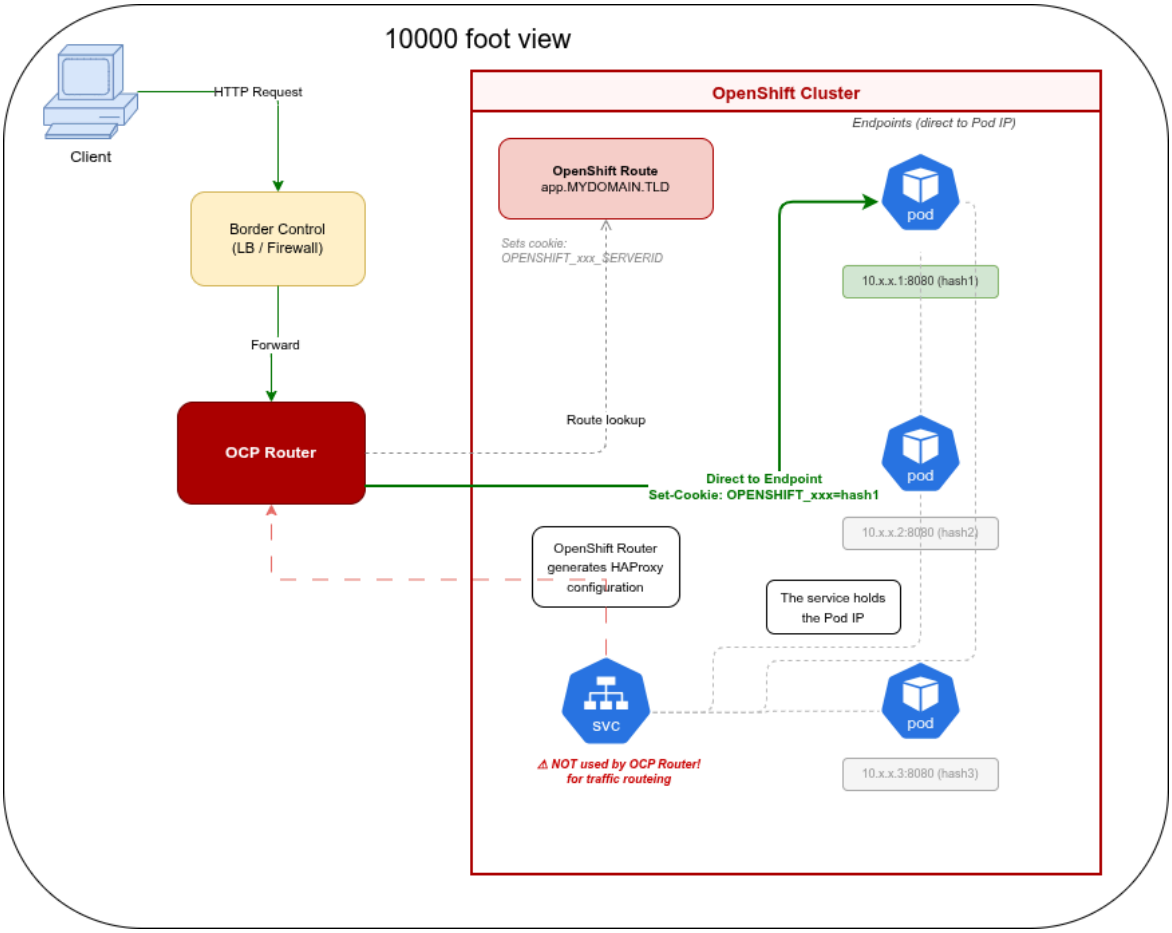

Figure 1: Session stickiness flow: client request goes through border control device to OCP Router, which sets a cookie and routes directly to a specific pod endpoint, bypassing the Kubernetes Service

Flow description

- The user requests `app.MYDOMAIN.TLD` and the traffic terminates at the **border control device(s)**.

The border control device can be almost anything — from a Raspberry Pi to a full-blown highly available network farm. Regardless of what sits in front of the OCP Router, there is always a Route configured in OpenShift.

The flow is always the same:

- The OCP Router looks up the Route configuration and selects the right backend *Pod*.
- The OCP Router **DOES NOT** route via the Kubernetes Service!
- The request is forwarded directly to the Kubernetes Endpoint — the Pod's IP and port.

What's a "route"

A "route" is the **external entrypoint** to a Kubernetes Service.

This is one of the biggest differences between Kubernetes and OpenShift Enterprise (= OCP) and okd.

The OpenShift Router is built in, whereas Kubernetes Ingress is an additional component you need to install separately. Both approaches have their pros and cons.

OpenShift Router

Until 9 May 2018, the HAProxy-based router and the F5-based router were the only supported router options.
Since 9 May 2018, NGINX has also been available as a router.

The Router Overview and Routes describe the concept and the setup in OpenShift.

The reason stickiness works is that the OpenShift router targets Endpoints directly — and therefore individual application pods.

Kubernetes Ingress

The Kubernetes Ingress resource is the standard way to handle external HTTP(S) traffic in a Kubernetes cluster, routing it to a Service.

There are several ingress controllers available:

- haproxy-ingress
- nginx-ingress
- istio-ingress
- caddyserver-ingress
- traefik ingress
- and so on. Search for kubernetes ingress

Since there are several options to choose from, you can pick the one that fits your setup best.

2023 update: Since the Kubernetes Gateway API reached v1.0 GA in October 2023, it is the recommended path for new deployments. The Ingress API is now considered feature-frozen — it receives bug fixes only. All major ingress controllers (NGINX, Traefik, Istio, HAProxy) have added Gateway API support. Cookie-based session persistence is supported via HTTPRoute with the `SessionPersistence` extension (v1.2+).

The stickiness

If you ask yourself:

> After all this router, ingress, gateway, and load balancer stuff — what is the actual solution for my stickiness?

The answer is, as so often in IT, a multi-layered one.

The following conditions must be met to enable session stickiness:

1. You **must** have an HTTP/HTTPS endpoint.
2. Session handling must be cookie-based.

OpenShift solution

In OpenShift, cookie-based stickiness is active by default.

You can only use this stickiness with the following Route Types

- plain http
- edge
- Re-encryption

> **i SNI**
>
> TLS termination in OpenShift Container Platform relies on SNI for serving custom certificates. Any non-SNI traffic received on port 443 is handled with TLS termination and a default certificate (which may not match the requested host name, resulting in validation errors).

Kubernetes solution

Ingress (pre-2023)

The solution depends on which ingress controller you choose.

With haproxy-ingress you get full HAProxy feature support, so cookie-based stickiness works out of the box via annotations.

For other controllers, stickiness support varies or may not be available at all.

Gateway API (2023+)

The Kubernetes Gateway API introduces native session persistence support via the `SessionPersistence` policy (stable since v1.2). It can be applied directly to an `HTTPRoute`:

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: my-app
spec:
  rules:
    - backendRefs:
        - name: my-service
          port: 8080
      sessionPersistence:
        type: Cookie
        cookieName: MY_SESSION_COOKIE
        absoluteTimeout: 1h
```

This is controller-independent — any Gateway API-conformant implementation (Envoy Gateway, Traefik, NGINX Gateway Fabric, HAProxy, Istio) will honor the `SessionPersistence` spec. No controller-specific annotations needed.

My personal opinion/conclusion

If you needed cookie-based session stickiness out of the box in 2018, OpenShift was the obvious choice — it worked with zero configuration.

Today the landscape has changed. With the Kubernetes Gateway API and its `SessionPersistence` support, plain Kubernetes deployments are equally capable without relying on controller-specific annotations or enterprise features. If you are starting fresh, the Gateway API is the right path.

That said, if you are already running OpenShift, the built-in Route stickiness still works perfectly well.

Even if it sounds like I am a Red Hat sales rep — I am not.
I am just a partner who likes OpenShift :satisfied:

Updates

- 2018-05-27 Add Træfik ingress
- 2023-10 Kubernetes Gateway API v1.0 GA — recommended over Ingress for new deployments
- 2026-04-06 Update to current state of the tools and migrate to privat zola blog