SIMPLY COMPLEX TASK OF KUBERNETES INGRESS

Richard Li
WHAT IS INGRESS?
Pod
Pod
Pod

Static IP address

External Load Balancer
Get traffic into cluster

Kubernetes routing service
Route traffic inside your cluster

Pod
Pod
Pod
kind: Service
type: LoadBalancer

Support for LoadBalancer type dependent on cloud provider.
kind: Service

type: NodePort

Static IP address

Bare metal load balancer

Kubernetes service

Pod

Pod

Pod
TL; DR. Create a Service of type LoadBalancer if you’re using AWS, GKE, etc. Otherwise, use type NodePort.
This is all Layer 4. What about Layer 7?
Static IP address

External Load Balancer (normally, L4)

Kubernetes routing service (L7)
1. Configuration changes are sent to the control plane.

2. Control plane computes the differences and creates an updated proxy configuration.

3. New configuration is passed to proxy.
CONFIGURING L7
Decentralized, declarative configuration.

SILOED TEAMS

PM  Dev  QA  Ops

Imperative, API-driven configuration

DECENTRALIZED, FULL-LIFECYCLE TEAMS

Users  Ads  Catalog  Search

Declarative configuration
You can configure routing via ingress resources (e.g., use an ingress controller).

```yaml
apiVersion: extensions/v1beta1
class: Ingress

metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /

spec:
  rules:
  - http:
      paths:
        - path: /testpath
      backend:
        serviceName: test
        servicePort: 80
```
You can configure routing via annotations (e.g., Ambassador API Gateway).

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: httpbin
annotations:
  getambassador.io/config: |
  ---
    apiVersion: ambassador/v0
    kind: Mapping
    name: httpbin_mapping
    prefix: /httpbin/
    service: httpbin.org:80
    host_rewrite: httpbin.org
spec:
  ports:
    - name: httpbin
      port: 80
---
```
Ingress provides portability between different controllers …

How many NGINX ingress controllers exist on Kubernetes?
Ingress provides portability between different controllers …

- ingress-nginx (Google)
- kubernetes-ingress (NGINX)
- kubernetes-ingress with NGINX Plus (NGINX)
- …
Except ingress isn’t actually portable.

<table>
<thead>
<tr>
<th>Aspect or Feature</th>
<th>kubernetes/ingress-nginx</th>
<th>nginxinc/kubernetes-ingress with NGINX</th>
<th>nginxinc/kubernetes-ingress with NGINX Plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>** Fundamental **</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authors</td>
<td>Kubernetes community</td>
<td>NGINX Inc and community</td>
<td>NGINX Inc and community</td>
</tr>
<tr>
<td>NGINX version</td>
<td>Custom NGINX build that includes several third-party modules</td>
<td>NGINX official mainline build</td>
<td>NGINX Plus</td>
</tr>
<tr>
<td>Commercial support</td>
<td>N/A</td>
<td>N/A</td>
<td>Included</td>
</tr>
<tr>
<td>Load balancing configuration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Merging Ingress rules with the same host</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>HTTP load balancing extensions - Annotations</td>
<td>See the supported annotations</td>
<td>See the supported annotations</td>
<td>See the supported annotations</td>
</tr>
<tr>
<td>HTTP load balancing extensions -- ConfigMap</td>
<td>See the supported ConfigMap keys</td>
<td>See the supported ConfigMap keys</td>
<td>See the supported ConfigMap keys</td>
</tr>
<tr>
<td>TCP/UDP</td>
<td>Supported via a ConfigMap</td>
<td>Supported via a ConfigMap with native NGINX configuration</td>
<td>Supported via a ConfigMap with native NGINX configuration</td>
</tr>
<tr>
<td>Websocket</td>
<td>Supported</td>
<td>Supported via an annotation</td>
<td>Supported via an annotation</td>
</tr>
</tbody>
</table>

→ Each controller does custom extensions to the ingress specification for features.
→ Each controller has different features.
→ The “solution” to this conundrum is to keep Ingress in beta (since Kube 1.1)

https://github.com/nginxinc/kubernetes-ingress/blob/master/docs/nginx-ingress-controllers.md
Ingress controller / resources ≠ ingress
Ingress controller / resources = routing
The NGINX ingress controller is a Service of type LoadBalancer!

```yaml
kind: Service
apiVersion: v1
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
annotations:
spec:
type: LoadBalancer
selector:
  app.kubernetes.io/name: ingress-nginx
  app.kubernetes.io/part-of: ingress-nginx
ports:
- name: http
  port: 80
  targetPort: http
- name: https
  port: 443
  targetPort: http
```
TL; DR.

- Most ingress models on Kubernetes use a decentralized, declarative configuration model.
- This configuration occurs through Kubernetes manifests.
- Ingress is one format for routing configuration, but there are others.
- Decide on ingress solutions based on features/functionality/robustness.
REAL-WORLD INGRESS
Ingress isn’t just about routing.

- **Protocols.** gRPC, HTTP/2, WebSockets.
- **Resilience.** Timeouts, rate limiting, circuit breakers.
- **Testing.** Canary releases, traffic shadowing.
- **Observability.** Distributed tracing, metrics.
- **TLS.** Redirect from cleartext, SNI.
- **Load balancing.** Round robin, sticky sessions, maglev …
And there are operational concerns, too!

➔ **Upgrades.** How do you upgrade and test your ingress solution?

➔ **Hitless reloads.** How do you avoid impacting your users during configuration changes?

➔ **Endpoint vs service routing.** Do you need to route to Kubernetes services or pods?
Upgrades

- Ingress has a new release (v0.35 —> v0.36)
- Run a “full stack canary” of new ingress versions
  - Route most of traffic through v0.35
  - Route some traffic through v0.36 (for some services, for 1% traffic, …)
Configuration changes are sent to the control plane.

Control plane computes the differences and creates an updated proxy configuration.

New configuration is passed to proxy.
Configuration changes are sent to the control plane.

Control plane computes the differences and creates an updated proxy configuration.

New configuration is passed to proxy.

Proxy reloads configuration.
In Kubernetes, configurations can change frequently ("microservices"), triggering proxy reloads.

- Existing connections can drop
- Response latency increases
- Load balancing quality goes down
Strategies for mitigating reloads

- Don’t trigger reload when there is no change in state
- Do “hot reloads” (aka “hitless reloads” aka “hot restart”): HAProxy 1.8, NGINX, Envoy Proxy
- Use APIs to manage configuration (if available): NGINX Plus, Envoy Proxy, NGINX Lua handler, HAProxy 1.8
Netflix announces Hystrix

2016

2017

• NGINX Plus R13 (Aug 2017)
  • Runtime API
  • Shadowing

2018

• 1.8 released
  • Finally supports hitless reloads, runtime API

• Istio announces in May; will use Envoy
Pod

Static IP address

External Load Balancer
Get traffic into cluster

Kubernetes routing service
Route traffic inside your cluster

Pod

Pod

Pod
By default, Kubernetes does round-robin load balancing.

Kube-proxy maintains the mapping between a Kubernetes service and its pods.
You can bypass Kubernetes default with endpoint routing.

(This is how you get sticky sessions, fancier load balancing, etc.)
HERE’S YOUR “BUZZWORD BINGO” CARD FOR THE MEETING.

IF THE BOSS USES A BUZZWORD ON YOUR CARD, YOU CHECK IT OFF. THE OBJECTIVE IS TO FILL A ROW.

YOU’RE ALL VERY ATTENTIVE TODAY. MY PROACTIVE LEADERSHIP MUST BE WORKING! BINGO, SIR.
SERVICE MESH.

→ Service mesh facilitates service-to-service communication
  • Routing
  • Resilience
  • Observability
  • Security (end-to-end encryption)
→ Grows more important as your topology gets deeper / more complex
“Sidecar” deployment model

Service B
Sidecar proxy

Service C
Sidecar proxy

Service A
Sidecar proxy

Service D
Sidecar proxy
Ingress versus service mesh

→ Service meshes frequently include an ingress (e.g., Istio has a “gateway” abstraction)
→ Service meshes assume you have control of the client
→ Ingress assumes you have no control of the client
  • HTTP → HTTPS redirect
  • OAuth / OIDC
TL; DR.

→ Think about the functional aspects of ingress, as well as the operational aspects when choosing an ingress.

→ The edge and service mesh are different but related use cases.

→ If you’re looking for a new Kubernetes job, add service mesh to your LinkedIn profile (you’ll know more than your hiring manager, anyway).
INGRESS CAN HELP YOU GO FASTER
Scenario

Shadow & Routing: Route 100% of prod traffic to 1.0 and 1.1.

Metrics: Compare latency on requests to v1.0 vs v1.1
Summary

→ To get traffic into a cluster, you need a service of LoadBalancer or NodePort (and probably a LoadBalancer)

→ This service is implemented as a combination of a control plane and a L7 proxy such as NGINX or Envoy Proxy

→ Ingress resources are a specific way of controlling routing into your cluster, but not the only way

→ When choosing ingress, think about your protocol, resilience, observability, and other requirements
Thank you!

➔ richard@datawire.io
➔ Twitter: @rdli
➔ Slack: @rdl (On Kubernetes, Envoy, and Ambassador Slack channels)