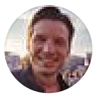
Using Nginx-Ingress as a Static Cache for Assets Inside Kubernetes

Optimizing Nginx on Kubernetes Without a Adding a Cloud CDN.



Diederik van der Boor
6 min read · Jul 27, 2019

Kubernetes clusters often use the NGINX Ingress Controller. This provides a nice solution when hosting many websites. Yet all contents is retrieved from the back-end pods at every request. Would it be possible to cache that?

One of the wonderful features of Nginx is content caching. So can we apply this inside a Kubernetes Cluster? Certainly!

> This avoids continuous fetching of static HTTP resources from backend pods. Instead, the Nginx server directly returns them, as if it were hosting static web sites!

For GKE Users: What About Google's Cloud CDN?

When hosting inside Google Kubernetes Engine (GKE), there is the option for Cloud CDN. Sadly, this is only supported for native HTTP load balancers — not with the TCP load balancer that the NGINX Ingress controller uses.

For large websites, Cloud CDN is most likely the best cost effective approach. When running many small websites, the billing costs become quite high. GKE spawns a new HTTP load balancer for each Ingress resource — hence charging you per domain name. For such scenario, this solution improves performance without additional charges.

Preparation

Configuring the Web Application

To leverage caching, the backend web application should send the proper HTTP headers to indicate the resources can be cached. Most important are:

- **Cache-Control: public, max-age=...** tells the upstream proxy it may cache this URL, and how long.
- **Expires** also works instead of **max-age**, but this is the legacy header.
- **Content-Length** is typically required by various caches.

When those headers are set, Nginx or any upstream (like Google Cloud CDN) is able to cache the resource.

> *Tip for Python projects: use whitenoise and all these headers will be properly set. It also generates unique URL's with hashed suffixes. Changing a file therefore changes it's URL, so site changes or upgrades won't serve older cached versions.*

Understanding the Nginx Configuration

The documentation on content caching, clarifies most options. This is a reasonable configuration for `nginx.conf` to apply basic caching:

```
http {
  ...

  # Declare a cache named static-cache
  proxy_cache_path     /tmp/nginx-cache levels=1:2 keys_zone=static-
cache:2m max_size=100m inactive=7d use_temp_path=off;
  proxy_cache_key       $scheme$proxy_host$request_uri;
  proxy_cache_lock      on;
  proxy_cache_use_stale updating;

  server {
    listen      80;
    server_name example.com;
    ...

    location / {

      proxy_buffering       on;
      proxy_cache           static-cache;
      proxy_cache_valid     404 1m;
      proxy_cache_use_stale error timeout updating http_404 http_500
http_502 http_503 http_504;
      proxy_cache_bypass    $http_x_purge;
      add_header            X-Cache-Status $upstream_cache_status;

      ...
      proxy_pass http://example-backend.internal;
    }
    }
  }
```

> *The **proxy_buffering** setting is important: Nginx can only cache resources that are first fully received (=buffered), before they are sent to the client.*

Configuring Nginx-Ingress Inside Kubernetes

Inside the NGINX controller, we don't have full control over the configuration file. The ingress controller generates it's own `nginx.conf.` Still, raw configuration can be added using the http-snippet option inside the configmap.

The snippet contents is inserted directly at the top-level `http { ... }` block of the generated `nginx.conf` file — right where we need it.

When using the stable/nginx-ingress Helm chart, use the following values:

```
controller:
  config:
    http-snippet: |
      proxy_cache_path /tmp/nginx-cache levels=1:2 keys_zone=static-
cache:2m max_size=100m inactive=7d use_temp_path=off;
      proxy_cache_key $scheme$proxy_host$request_uri;
      proxy_cache_lock on;
      proxy_cache_use_stale updating;
```

The ingress annotations activate caching for the selected web site:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: mywebsite
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/proxy-buffering: "on"  # Important!
    nginx.ingress.kubernetes.io/service-upstream: "true"
    nginx.ingress.kubernetes.io/configuration-snippet: |
      proxy_cache static-cache;
      proxy_cache_valid 404 1m;
      proxy_cache_use_stale error timeout updating http_404 http_500
http_502 http_503 http_504;
      proxy_cache_bypass $http_x_purge;
      add_header X-Cache-Status $upstream_cache_status;
```

(mentioned in the comments: The **nginx.ingress.kubernetes.io/service-upstream** annotation lets Nginx use the ClusterIP of the service instead of the Pods. This is needed for reliable 503-handling).

There is still one other minor problem: all resources are buffered inside Nginx before streaming them to the client. Let's configure the caching on a single path.

Configuring a Sub Path for Caching

The easiest way is apply settings to a single path (e.g. `/static/` and `/media/` for Django websites), is the create another ingress for that path.

> *The nginx-ingress-controller generates a single* `server { ... }` *block for all ingresses of the same domain name inside* `nginx.conf`.

Each ingress becomes a `location` block within that `server` block, with all required upstream proxy parameters in place. As such, some global options like TLS configuration don't really need to be repeated:

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: mysite
  annotations:
    kubernetes.io/ingress.class: nginx
    ingress.kubernetes.io/force-ssl-redirect: "true"
    nginx.ingress.kubernetes.io/proxy-body-size: 10m
spec:
  tls:
    - secretName: mysite-ssl
      hosts:
        - mysite.example.com
  rules:
    - host: mysite.example.com
      http:
        paths:
          - path: /
            backend:
              serviceName: mysite
              servicePort: http
---
# Leverage nginx-ingress cache for /static/
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: mysite-static
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/proxy-buffering: "on"
    nginx.ingress.kubernetes.io/service-upstream: "true"
    nginx.ingress.kubernetes.io/configuration-snippet: |
      proxy_cache static-cache;
      proxy_cache_valid 404 10m;
      proxy_cache_use_stale error timeout updating http_404 http_500
http_502 http_503 http_504;
      proxy_cache_bypass $http_x_purge;
      add_header X-Cache-Status $upstream_cache_status;
spec:
  rules:
    - host: mysite.example.com
      http:
        paths:
          - path: /static/
            backend:
              serviceName: mysite
              servicePort: http
```

One benefit of this approach is that the **proxy_buffering** parameter is only enabled for the sub path that uses caching. The remaining URLs are still streamed directly to the client without buffering them entirely first.

Testing That It Works

The easiest way to test, is by requesting the same URL twice:

```
curl --head https://mysite.example.com/static/logo.png
```

This should show **x-cache-status: HIT** for the second attempt.

> *While the browser also shows HTTP headers in the network panel, it keeps showing **x-cache-status: MISS**. That's because it also caches the entire response when the Expires and Cache-Control headers are set. Only when the cache is entirely disabled, the browser will actually refresh and show the server response.*

Debugging nginx.conf

In case this configuration doesn't seem to work, it's useful to retrieve the generated `nginx.conf`. This reveals how the *nginx-ingress-controller* generates it's configuration using our Ingress resources:

```
KUBE_NAMESPACE=infra
NGINX_POD_NAME=$(kubectl get pods -n $KUBE_NAMESPACE --
selector=app=nginx-ingress,component=controller -o name | cut -f1 -
d' ')

kubectl exec -n $KUBE_NAMESPACE $NGINX_POD_NAME -- cat
/etc/nginx/nginx.conf | less
```

The generated `nginx.conf` should look something like:

```
server {
  server_name mysite.example.org
  ...

  location /static/ {
    # Contents of our mysite-static ingress

    # All our proxy settings:
    proxy_cache static-cache;
    proxy_cache_valid 404 10m;
    proxy_cache_use_stale ...
    # ...

    proxy_pass http://upstream_balancer;
  }

  location / {
    # Contents of the top-level ingress
    # ...

    proxy_pass http://upstream_balancer;
  }
}
```

The following commands also help debugging:

```
kubectl logs -f --since=5m -n $KUBE_NAMESPACE $NGINX_POD_NAME

kubectl describe ingress mysite-static
```

Clearing Cached Resource

While caching was easy, how about invalidating the cache on updates? There are a few ways to deal with this issue:

- Generate unique URL names per release (e.g. an md5 hash). This avoids the whole problem, as each new release changes the URL of the resource.
- Move the Nginx cache to memcache or redis, which is easier to purge.
- Restart the Nginx-Ingress container to flush the cache.
- Using the commercial Nginx Plus offers better ways to purge the cache, e.g. by requesting the URL with a HTTP `PURGE` method. The community edition only supports the **proxy_bypass_cache** setting.

. . .

Final Words

Would this be suited for everyone? Definitely not. Larger web sites would be better served with a full CDN that persists the cache across regions. However, the advantages for smaller sites are:

- ✅ Cheap performance wins.
- ✅ Reduces load on backend pods.
- ✅ No reliance on external services (like memcache).
- ✅ Ideal for small clusters and low-traffic sites.

Disadvantages:

- ❌ Running many Nginx replicas spreads the cache.
- ❌ Restarting Nginx pods clears the cache.
- ❌ High traffic sites are better served with a full CDN.

The performance gain is significant. In my case, it allows me to host multiple web sites on GKE's **n1-standard-1** machine with a decent performance.

I'm curious to hear how this works for your situation!

Nginx Google Cloud Platform Kubernetes Cdn Cache