channable Signup

< Overview

Tech

Setting up a private Nix cache for fun and profit

November 7, 2022

all point to the same origin server. This way you can horizontally scale out the number of caching servers. Another neat consequence is that you can have a *hierarchy* of caching servers: One caching server can point to another caching server as its upstream. And the latter caching server can then point to the origin server. The key property that makes this design possible is the immutability of the URLs. What we have just described is how e.g. content distribution networks (CDNs) work. However, it is *also* how Nix caches work. And in this blog post we will explain how you can set up a private Nix cache in your office. This will speed up the downloading of build dependencies and minimize the amount of traffic that needs to be downloaded from outside the office network.

Overview

Our private Nix cache is implemented as an Nginx server that is running on a box in our office. We have three upstream servers that we want to cache:

1. `https://cache.nixos.org` - the official Nix binary cache providing prebuilt binaries for Nixpkgs and NixOS. It is used automatically by the Nix package manager to speed up builds.
2. `https://channable-public.cachix.org` - a public Nix cache for our own open-source projects, hosted by https://www.cachix.org/

3. We must support authentication for our private cache
4. We must configure Nix to skip querying the cache if it is unreachable, e.g. if a developer is using a laptop and working from home

Let's look at each of these steps in turn.

Configuring Nix to use a private cache

The configuration in this section will have to be done on each developer machine (not on the server running Nginx).

First, we will have to tell Nix where to find our local caching server. We will assume that the local cache server is called `cachixcache` and will have IP `10.0.0.10`. We are going to point three different domain names to this IP address by editing `/etc/hosts` and adding for example:

```
10.0.0.10       channable-cachix-org.cachixcache channable-public-cachix-org.cac
```

Note, that you can pick any domain name here, as long as you pick different ones for each upstream server. Having these three distinct domain names is required to be able to route to different upstream servers in Nginx, as we will explain in the next section. If you have an internal DNS server in your office, you can also configure the domain names there, and avoid having to edit `/etc/hosts` on each dev machine.

to the substituter URL, a feature seemingly only noted in the release notes for Nix 2.4 (link).

If you only use public upstream caches, you are now done with the local Nix configuration and can continue reading the next section. However, if you use a private cache, with authentication, then you still need to configure Nix to know about that. This works as follows:

Open `~/.config/nix/netrc` and add a line like this:

```
machine channable-cachix-org.cachixcache password __your_password__
```

With this configuration in place Nix will always first try to contact the local cache before falling back to the global caches.

Next, we will see how we need to configure Nginx to serve as a local caching proxy.

Configuring Nginx as a caching proxy

The configuration in this section will have to be done on the local caching server running Nginx.

```
# First off, some common configuration options for all vhosts in this file.
# Note that because these options are in the `http` context these options will
# also apply to any other `proxy_pass` directives for this server!
# If you apply this to an existing Nginx server you'll want to move these
# directives into the `server` blocks.

# Tell Nginx to set up a response cache at `/data/nginx/cache`, with a maximum
# size of 800 GB or 400_000 stored files.
# Also set `inactive` to tell Nginx to keep files until they haven't been
# accessed for a year, instead of the default of removing files after they
# haven't been accessed for 10 minutes.
# See: https://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_cache_pat
```

```
proxy_cache_path /data/nginx/cache max_size=800G keys_zone=cache_zone:50m inacti

# Tell Nginx to actually use the response cache to cache requests.
# See: https://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_cache
proxy_cache cache_zone;

# Since Nix store paths are immutable, we can cache successful responses for a
# long time.
# We only want to cache successful responses: if we get a 404 error or a 401
# error, we want the request to be retried the next time a client asks for it.
# This is the default behaviour of `proxy_cache_valid` if no specific response
# codes are specified.
```

```
# log.
# You'll probably want to remove this option if your network does support IPv6
# or you use a different DNS server.
# See: https://nginx.org/en/docs/http/ngx_http_core_module.html#resolver
resolver 8.8.8.8 ipv6=off;

# When connecting to an upstream server, do use TLS SNI to indicate which server
# to connect to. Without this option Nginx fails to connect to Cachix upstreams.
# See: https://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_ssl_serve
proxy_ssl_server_name on;

# When connecting to an upstream server, do verify the TLS certificate since
# this is outside of our network. Nginx defaults to not verifying certificates.
# See: https://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_ssl_trust
proxy_ssl_verify on;
proxy_ssl_trusted_certificate /etc/ssl/certs/ca-certificates.crt;

# === PER-UPSTREAM CONFIGURATION ===
# With the above common configuration options, we can now define the upstream
# substituters to connect to.

# For each upstream substituter we define a separate `server` block that
# forwards traffic to the actual server. Nginx will determine based on the Host
# header which upstream server is to be used, and will check if the requested
```

This website uses cookies

We use cookies to personalise content and ads, to provide social media features and to analyse our traffic. We also share information about your use of our site with our social media, advertising and analytics partners who may combine it with other information that you've provided to them or that they've collected from your use of their services.

Consent Details About

Allow all

Manage preferences

```
    location / {
        # This cache requires authentication, so forward the Authorization
        # header to Cachix.
        proxy_set_header Authorization $http_authorization;
        proxy_set_header Host $proxy_host;

        proxy_pass https://channable.cachix.org;
    }
}

server {
    listen 80;
    server_name channable-public-cachix-org.cachixcache;

    location / {
        proxy_set_header Host $proxy_host;
        proxy_pass https://channable-public.cachix.org;
    }
}
```

A few notes on this config file:

- make sure to only pass the `Authorization` header to the private upstream to avoid

from home

However, there was still one remaining problem that prevented us from rolling this out to the whole dev team: most of our devs work on laptops and are only in the office on some days, while working from home on other days. This is problematic because the private cache is only available when people are in the office, not when they are at home. However, Nix will now always try to contact the private cache first, even when people are at home. This will make Nix *slower* at home, since Nix will retry for a few seconds before falling back to the upstream binary caches. This was clearly not acceptable, so we needed to find a solution for this, so that Nix would be faster in the office, while not being slower when working from home.

To work around this issue we installed another Nginx process in the pipeline, but this time on our developers' devmachines. We configure this local Nginx process to do two things:

1. Always respond to `/nix-cache-info` with a fixed response, which is required for Nix to consider it a binary cache, and
2. Forward any other requests to the local cache server if it is available, and to respond with a 404 error otherwise.

A response with status code 404 causes Nix to not retry querying the substituter for the path, and Nix will instead immediately query the next substituter, if any.

```
        proxy_send_timeout 100ms;
        proxy_connect_timeout 100ms;

        # Serve a 404 response if the cache server cannot be reached:
        error_page 502 504 =404 @fallback;

        # Forward to the actual cache server:
        proxy_set_header Host $http_host;
        proxy_set_header Authorization $http_authorization;
        proxy_pass https://10.0.3.10;
    }
}
```

After this Nginx configuration is running locally it can be used by adjusting your `/etc/hosts` to have the `*.cachixcache` hosts resolve to localhost instead:

```
127.0.0.1    cache-nixos-org.cachixcache channable-cachix-org.cachixcache channa
```

Once this configuration is in place Nix should automatically fall back to the upstream substituters if the local cache server is unreachable.

How Nix uses compression

is available, we will look into upgrading to that, since it offers very fast (de)compression speeds for only slightly bigger storage sizes.

Conclusion

This project started out as a fun Hackathon project with the goal to speed up our builds. We are happy to report that our local downloads are now significantly faster when served from the local Nix cache as when they are served from the internet. We also save a lot of bandwidth, since we have many developers working in the office on any day, and many of them will have shared build artifacts, which can now be served from the cache (after the first download).

This project has sped up our local builds considerably as we are now able to get prebuilt binaries at 1 Gbit/s and are only limited by our internal network. The latency for cache hits is also considerably lower now, leading to happy devs.

Robert Kreuzer

Co-founder & CTO

see if we have an open position that suits you!

Apply now

Solutions

For retailers

For agencies

Products

Feed Management

PPC Optimization

Marketplace Integration

Insights & Analytics

Integrations

Pricing

Consent | Details | About

This website uses cookies

We use cookies to personalise content and ads, to provide social media features and to analyse our traffic. We also share information about your use of our site with our social media, advertising and analytics partners who may combine it with other information that you've provided to them or that they've collected from your use of their services.

Allow all

Manage preferences

Bug bounty

Cookie policy

Recruitment Policy

Capterra

Trustpilot