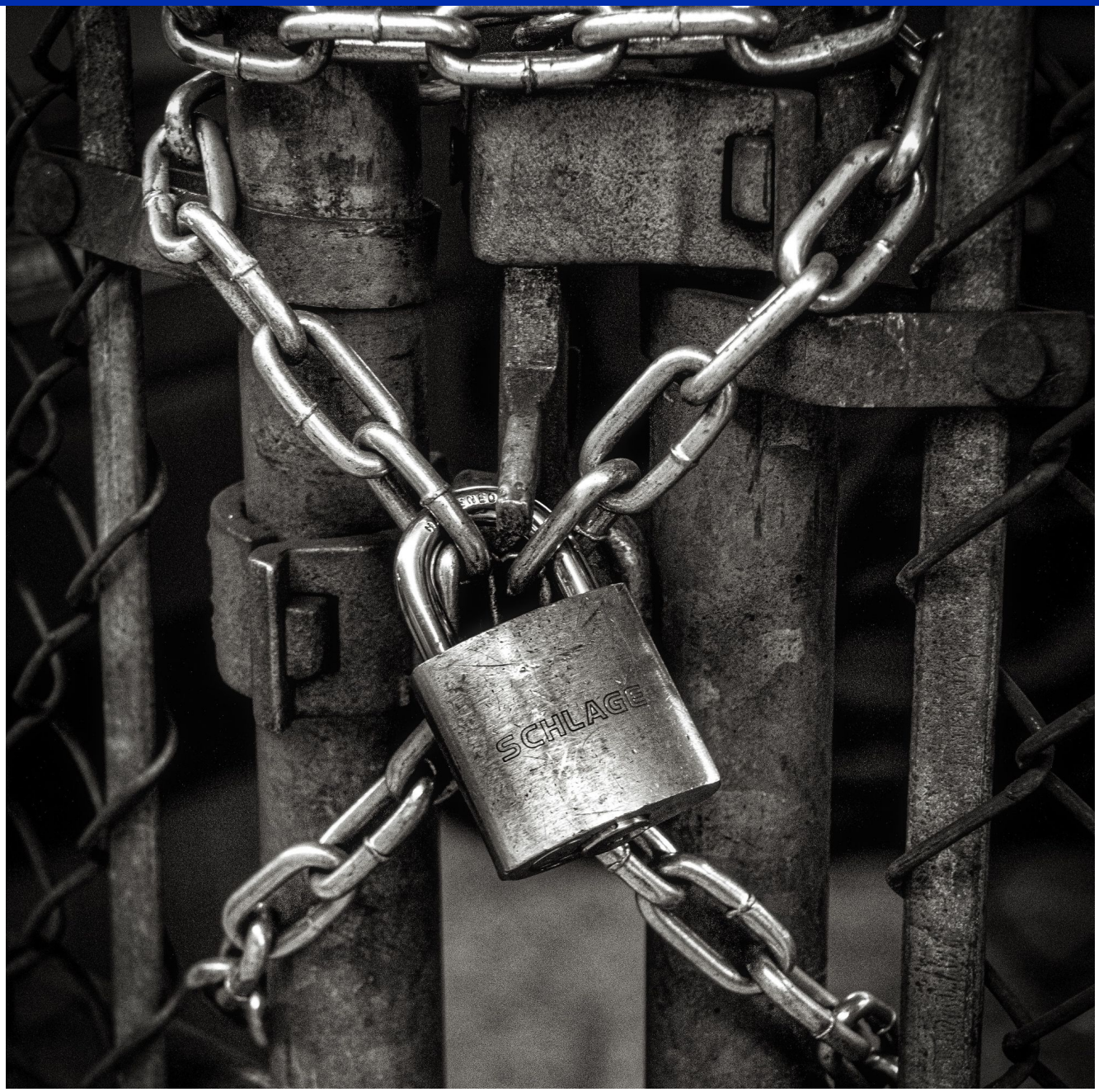
27 DE ABRIL DE 2022

Como configurar um proxy reverso de modo fácil e seguro com Docker, Nginx e Letsencrypt

Tradutor: Fernando Mota Freitas

Autor: freeCodeCamp.org (em inglês)



Artigo original: How to set up an easy and secure reverse proxy with Docker, Nginx & Letsencrypt

Escrito por: Kasper Siig

Introdução

definitivamente algo que funciona, e as pessoas fazem isso há muito tempo.

No entanto, não seria bom digitar *plex.example.com* e ter acesso instantâneo ao seu servidor de mídia? É exatamente isso que um proxy reverso fará por você, e combiná-lo com o Docker está mais fácil do que nunca.

Pré-requisitos

Docker e Docker-Compose

Você deve ter o Docker versão 17.12.0+ e o docker-compose versão 1.21.0+.

Domínio

Você deve ter um domínio configurado e ter um certificado SSL associado a ele. Se você não tiver um, siga meu guia aqui (texto em inglês) sobre como obter um gratuitamente com o LetsEncrypt.

O que este artigo abordará

Acredito firmemente na importância de se entender o que você está fazendo. Houve um tempo em que eu seguia guias e não fazia ideia de como solucionar falhas. Se é assim que você quer fazer isso, aqui está um ótimo tutorial (em inglês), que aborda como configurá-lo. Embora meus artigos sejam longos, você deve acabar entendendo como tudo funciona.

cabeçalhos. Assim, sinta-se à vontade para pular uma seção, se quiser. Eu recomendo ler o artigo inteiro uma vez primeiro, antes de começar a configuração.

O que é um proxy reverso?

Proxy comum

Vamos começar com o conceito de um proxy comum. Embora este seja um termo muito prevalente na comunidade de tecnologia, não é o único lugar em que é usado. Um proxy significa que a informação está passando por um terceiro, antes de chegar ao local.

Digamos que você não queira que um serviço saiba seu IP. Você pode, nesse caso, usar um proxy. Um proxy é um servidor que foi configurado especificamente para essa finalidade. Se o servidor proxy que você está usando estiver localizado, por exemplo, em Amsterdã, o IP que será mostrado para o mundo externo será o IP do servidor em Amsterdã. Os únicos que saberão seu IP são os que controlam o servidor proxy.

Proxy reverso

Para simplificar, um proxy adicionará uma camada de mascaramento. É o mesmo conceito em um proxy reverso, excetuando o fato de que, em vez de mascarar as conexões de saída (você acessando um servidor web), são as conexões de entrada (pessoas acessando seu servidor web) que serão mascaradas. Você simplesmente fornece um URL como *exemplo.com* e, sempre que as pessoas acessarem esse URL, seu proxy reverso cuidará para onde vai essa solicitação.

solicitações provenientes de *exemplo.com* para Server1. Um dia, você tem algumas atualizações para a página da web. Em vez de tirar o site do ar para manutenção, basta fazer a nova configuração no Server2. Feito isso, basta alterar uma única linha em seu proxy reverso e agora as solicitações são enviadas para o Server2. Supondo que o proxy reverso esteja configurado corretamente, você não deve ter absolutamente nenhum tempo de inatividade.

Talvez a maior vantagem de se ter um proxy reverso seja que você pode ter serviços rodando em várias portas, mas só precisa abrir as portas 80 e 443, HTTP e HTTPS, respectivamente. Todas as solicitações chegarão à sua rede nessas duas portas e o proxy reverso cuidará do resto. Tudo isso fará sentido quando começarmos a configurar o proxy.

Configurando o contêiner

O que fazer

`docker-compose.yaml`:

```
version: '3'

services:
  reverse:
    container_name: reverse
    hostname: reverse
    image: nginx
    ports:
      - 80:80
      - 443:443
    volumes:
```

Antes de tudo, você deve adicionar um novo serviço ao seu arquivo docker-compose. Você pode chamá-lo como preferir, neste caso eu escolhi "*reverse*". Aqui, escolhi *nginx* como a imagem. No entanto, em um ambiente de produção, geralmente é uma boa ideia especificar uma versão caso haja alguma alteração importante em atualizações futuras.

Então você deve vincular duas pastas. */etc/nginx* é onde todos os seus arquivos de configuração são armazenados e */etc/ssl/private* é onde seus certificados SSL são armazenados. É MUITO importante que sua pasta de configuração NÃO exista em seu host na primeira vez em que você iniciar o contêiner. Quando você inicia seu contêiner por meio do docker-compose, ele criará automaticamente a pasta e a preencherá com o conteúdo do contêiner. Se você criou uma pasta de configuração vazia em seu host, ela será montada e a pasta dentro do contêiner estará vazia.

Por que funciona

Não há muito nesta parte. Principalmente, é como iniciar qualquer outro contêiner com docker-compose. O que você deve observar aqui é que você está vinculando as portas 80 e 443. É aqui que todas as solicitações entrarão e serão encaminhadas para qualquer serviço que você especificar.

Configurando o Nginx

O que fazer

Agora, você deve ter uma pasta de configuração em seu host. Mudando para esse diretório, você deve ver vários arquivos

sem problemas.

Ainda dentro de `conf.d`, crie duas pastas: `sites-available` e `sites-enabled`. Navegue até `sites-available` e crie seu primeiro arquivo de configuração. Aqui, vamos configurar uma entrada para o Plex, mas sinta-se à vontade para usar outro serviço que você configurou, se desejar. Realmente não importa como o arquivo é chamado. Porém, prefiro chamá-lo de `plex.conf`.

Abra o arquivo e digite o seguinte:

```
upstream plex {
  server        plex:32400;
}

server {
  listen        80;
  server_name   plex.exemplo.com;

  location / {
    proxy_pass  http://plex;
  }
}
```

Entre no diretório `sites-enabled` e digite o seguinte comando:

```
ln -s ../sites-available/plex.conf .
```

Isso criará um link simbólico para o arquivo na outra pasta. Resta apenas uma coisa, que é mudar o arquivo `nginx.conf` na pasta

```
include /etc/nginx/conf.d/*.conf;
```

Altere isso para:

```
include /etc/nginx/conf.d/sites-enabled/*.conf;
```

Para que o proxy reverso realmente funcione, precisamos recarregar o serviço nginx dentro do contêiner. No host, execute `docker exec <nome-do-contêiner> nginx -t`. Isso executará um verificador de sintaxe em seus arquivos de configuração. Isso deve mostrar que a sintaxe está correta. Agora, execute `docker exec <nome-do-contêiner> nginx -s reload`. Isso enviará um sinal para o processo nginx de que ele deve recarregar. Parabéns! Agora você tem um proxy reverso em execução e deve conseguir acessar seu servidor em *plex.exemplo.com* (assumindo que você encaminhou a porta 80 para seu host em seu roteador).

Mesmo que seu proxy reverso esteja funcionando, você está executando em HTTP, que não fornece criptografia alguma. A próxima parte será como proteger seu proxy e obter uma pontuação perfeita no SSL Labs.

Por que funciona

O arquivo de configuração

as solicitações de entrada, qual domínio essa configuração deve corresponder e para onde deve ser enviada.

Da forma como este servidor está sendo configurado, você deve criar um arquivo para cada serviço para o qual deseja fazer solicitações de proxy. Então, obviamente, você precisa de alguma maneira de distinguir qual arquivo receberá cada solicitação. É isso que a diretiva `server-name` faz. Abaixo disso, temos a diretiva `location`.

No nosso caso, só precisamos de uma `location`, mas você pode ter quantas diretivas `location` quiser. Imagine que você tenha um site com um front-end e um back-end. Dependendo da infraestrutura que estiver usando, você terá o front-end como um contêiner e o back-end como outro contêiner. Você poderia então ter `location / {}`, que enviará solicitações para o front-end, e `location /api/ {}`, que enviará solicitações para o back-end. De repente, você tem vários serviços em execução em um único domínio memorável.

Quanto à parte de `upstream`, ela pode ser usada para balanceamento de carga. Se você estiver interessado em saber mais sobre como isso funciona, consulte a documentação oficial aqui. Para o nosso caso simples, você apenas define o nome do host ou endereço IP do serviço para o qual deseja fazer proxy, e qual porta deve ser a proxy. Em seguida, direcione ao nome do upstream na diretiva `location`.

Nome do host x endereço IP

Para entender o que é um hostname, vamos pensar em um exemplo. Digamos que você esteja em sua rede doméstica *192.168.1.0*. Você,

um nome de host. Nesse caso, daremos o nome de host *plex*. Agora, você pode acessar o Plex digitando *plex:32400* no seu navegador!

Esse mesmo conceito foi introduzido no docker-compose na versão 3. Se você observar o arquivo docker-compose anteriormente neste artigo, notará que dei a ele uma diretiva `hostname: reverse`. Agora todos os outros contêineres podem acessar meu proxy reverso por seu nome de host. Uma coisa que é muito importante observar é que o nome do serviço deve ser o mesmo que o nome do host. Isso é algo que os criadores do docker-compose escolheram impor.

Outra coisa realmente importante a ser lembrada é que, por padrão, os contêineres do docker são colocados em sua própria rede. Isso significa que você não poderá acessar seu contêiner pelo nome do host, se estiver usando o laptop na rede do host. São apenas os contêineres que podem acessar uns aos outros por meio de seu nome de host.

Então, para resumir e deixar bem claro. Em seu arquivo docker-compose, adicione a diretiva `hostname` aos seus serviços. Na maioria das vezes, seus contêineres receberão um novo IP toda vez que você reiniciar o contêiner. Portanto, referir-se a ele por meio do nome do host significa que não importa qual IP seu contêiner está recebendo.

sites-available e sites-enabled

Por que estamos criando os diretórios `sites-available` e `sites-enabled`? Isso não é algo da minha criação. Se você instalar o Nginx em um servidor, verá que ele vem com essas pastas. No entanto, como o Docker é construído com microsserviços em mente, onde

E, sim, você definitivamente pode criar uma pasta `sites-enabled` ou hospedar diretamente seus arquivos de configuração em `conf.d`. Fazendo desta forma, você pode ter uma configuração passiva. Digamos que você está fazendo a manutenção e não quer ter o serviço ativo; você simplesmente remove o link simbólico e o coloca de volta quando quiser que o serviço fique ativo novamente.

Links simbólicos

Links simbólicos são um recurso muito poderoso do sistema operacional. Eu pessoalmente nunca os usei antes de configurar um servidor Nginx, mas desde então os tenho usado em todos os lugares que posso. Digamos que você esteja trabalhando em 5 projetos diferentes, mas todos esses projetos usam o mesmo arquivo de alguma forma. Você pode copiar o arquivo em cada projeto e consultá-lo diretamente, ou pode colocar o arquivo em um local e, nesses 5 projetos, criar links simbólicos para esse arquivo.

Isso oferece duas vantagens: você ocupa 4 vezes menos espaço do que teria de outra forma e - o motivo mais poderoso de todos: muda o arquivo em um só lugar e ele muda em todos os 5 projetos de uma vez! Isso não tem muito a ver com o tópico do artigo, mas acho que vale a pena mencionar.

Protegendo o proxy Nginx

O que fazer

Vá para sua pasta de configuração, crie 3 arquivos e preencha-os com o seguinte :

```
add_header Strict-Transport-Security    "max-age=31536000; inclu
add_header X-Frame-Options              SAMEORIGIN;
add_header X-Content-Type-Options       nosniff;
add_header X-XSS-Protection             "1; mode=block";
```

`common_location.conf`:

```
proxy_set_header    X-Real-IP           $remote_addr;
proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_f
proxy_set_header    X-Forwarded-Proto   $scheme;
proxy_set_header    Host                $host;
proxy_set_header    X-Forwarded-Host    $host;
proxy_set_header    X-Forwarded-Port    $server_port;
```

`ssl.conf`:

```
ssl_protocols               TLSv1 TLSv1.1 TLSv1.2;
ssl_ecdh_curve              secp384r1;
ssl_ciphers                 "ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA
ssl_prefer_server_ciphers   on;
ssl_dhparam                 /etc/nginx/dhparams.pem;
ssl_certificate             /etc/ssl/private/fullchain.pem;
ssl_certificate_key         /etc/ssl/private/privkey.pem;
ssl_session_timeout         10m;
ssl_session_cache           shared:SSL:10m;
ssl_session_tickets         off;
ssl_stapling                on;
ssl_stapling_verify         on;
```

```
upstream plex {
  server        plex:32400;
}

server {
  listen        443 ssl;
  server_name   plex.example.com;

  include       common.conf;
  include       /etc/nginx/ssl.conf;

  location / {
    proxy_pass  http://plex;
    include     common_location.conf;
  }
}
```

Agora, volte para a raiz da sua pasta de configuração e execute o seguinte comando:

```
openssl dhparam -out dhparams.pem 4096
```

Isso levará muito tempo para ser concluído (até uma hora, em alguns casos).

Se você seguiu meu artigo sobre como obter um certificado SSL LetsEncrypt, seus certificados devem estar localizados em `</caminho/para/seu/letsencrypt/config>/etc/letsencrypt/live/<domínio>/`.

Quando ajudei um amigo a configurar isso em seu sistema, tivemos alguns problemas em que não foi possível abrir os arquivos quando eles estavam localizados nesse diretório. Provavelmente, a causa

`/etc/ssl/private` do contêiner Nginx. Na pasta recém-criada, você deve criar links simbólicos para os certificados na pasta de configuração do LetsEncrypt.

Quando o comando `openssl` terminar de ser executado, você deve executar o `docker exec <nome-do-contêiner> nginx -t` para se certificar de que toda a sintaxe está correta e, em seguida, recarregá-lo executando `docker exec <nome-do-contêiner> nginx -s reload`. Neste ponto, tudo deve estar funcionando e, agora, você tem um proxy reverso funcionando e perfeitamente seguro!

Por que funciona

Olhando no arquivo `plex.conf`, há apenas uma grande mudança - em qual porta o proxy reverso está escutando e informando que é uma conexão SSL. Depois, há 3 lugares onde estamos incluindo os outros 3 arquivos que criamos. Embora o SSL seja, de certo modo, seguro por si só, esses outros arquivos o tornam ainda mais seguro. No entanto, se por algum motivo você não quiser incluir esses arquivos, será necessário mover o `ssl-certificate` e `ssl-certificate-key` dentro do arquivo `.conf`. Eles são necessários para que uma conexão HTTPS funcione.

Common.conf

Olhando no arquivo `common.conf`, adicionamos 4 cabeçalhos diferentes. Os cabeçalhos são algo que o servidor envia ao navegador em cada resposta. Esses cabeçalhos dizem ao navegador para agir de uma determinada maneira e, então, cabe ao navegador impor esses cabeçalhos.

Strict-Transport-Security (HSTS)

navegador não permitirá que você faça uma conexão HTTP simples com o servidor, garantindo que toda a comunicação seja segura.

X-Frame-Options

Ao especificar esse cabeçalho, você especifica se outros sites podem ou não incorporar seu conteúdo em seus sites. Isso pode ajudar a evitar ataques de clickjacking.

X-Content-Type-Options

Digamos que você tenha um site onde os usuários possam fazer upload de arquivos. Não há validação suficiente nos arquivos, então um usuário carrega com sucesso um arquivo `php` para o servidor, onde o servidor espera que uma imagem seja carregada. O invasor pode então acessar o arquivo carregado. Agora, o servidor responde com uma imagem, porém o tipo MIME do arquivo é `text/plain`. O navegador fará o 'sniffing' do arquivo e renderizará o script php, permitindo que o invasor faça RCE (Remote Code Execution).

Com este cabeçalho definido como 'nosniff', o navegador não examinará o arquivo e simplesmente o renderizará como o que o servidor disser ao navegador que ele é.

X-XSS-Protection

Embora esse cabeçalho fosse mais necessário em navegadores mais antigos, é tão fácil de adicionar que você também pode. Alguns ataques XSS (Cross-site Scripting) podem ser muito inteligentes, enquanto alguns são muito rudimentares. Este cabeçalho dirá aos navegadores para verificar as vulnerabilidades simples e bloqueá-las.

X-Real-IP

Como seus servidores estão atrás de um proxy reverso, se você tentar ver o IP solicitante, sempre verá o IP do proxy reverso. Este cabeçalho é adicionado para que você possa ver qual IP está realmente solicitando seu serviço.

X-Forwarded-For

Às vezes, uma solicitação de usuário passará por vários clients antes de chegar ao seu servidor. Este cabeçalho inclui um array de todos esses clients.

X-Forwarded-Proto

Este cabeçalho mostrará qual protocolo está sendo usado entre client e servidor.

Host

Isso garante que seja possível fazer uma pesquisa de DNS reversa no nome de domínio. É usado quando a diretiva `server_name` é diferente daquela para a qual você está fazendo proxy.

X-Forwarded-Host

Mostra qual é o host real da solicitação em vez do proxy reverso.

X-Forwarded-Port

Ajuda a identificar em qual porta o client solicitou o servidor.

explicar neste artigo. Existem muitos tutoriais excelentes sobre como os handshakes de SSL funcionam e assim por diante. Se você quiser examinar esse arquivo específico, sugiro examinar os protocolos e cifras que estão sendo usados e qual a diferença que eles fazem.

Redirecionando HTTP para HTTPS

Os observadores talvez tenham notado que estamos ouvindo apenas na porta 443 nesta versão segura. Isso significaria que qualquer pessoa que tentasse acessar o site via *https://** passaria, mas tentar se conectar através de *http://** receberia apenas um erro. Felizmente, há uma solução muito fácil para isso. Crie um arquivo `redirect.conf` com o seguinte conteúdo:

```
server {
  listen        80;

  server_name   _;

  return 301 https://$host$request_uri;
}
```

Agora, apenas certifique-se de que ele apareça em sua pasta `sites-enabled` e, quando você recarregar o processo Nginx no contêiner, todas as solicitações para a porta 80 serão redirecionadas para a porta 443 (HTTPS).

fazer um teste para ver o nível de segurança de seu site. No momento em que escrevo isso, você deve obter uma pontuação perfeita. No entanto, há algo importante a se notar sobre isso.

Sempre haverá um equilíbrio entre segurança e conveniência. Neste caso, os pesos estão muito do lado da segurança. Se você executar o teste no SSL Labs e rolar para baixo, verá que há vários dispositivos que não poderão se conectar ao seu site porque não suportam novos padrões.

Portanto, tenha isso em mente quando estiver fazendo essa configuração. No momento, estou apenas executando um servidor em casa, onde não preciso me preocupar com tantas pessoas podendo acessá-lo. Mas se você fizer uma varredura no Facebook, verá que o site não terá uma pontuação tão boa, porém pode ser acessado por mais dispositivos.

Tradutor: Fernando Mota Freitas

https://github.com/f3rn4nd0000

Autor: freeCodeCamp.org (em inglês)

Ler mais publicações.

Excluir branches
Clonar branches
Date now em JavaScript
Media queries do CSS
Var, let e const em JavaScript
Fix do Live Server no VS Code
Axios em React
SQL em Python
ForEach em JavaScript
Interpretadas x compiladas
Fotos do Instagram
Imagens SVG em HTML e CSS