Self hosting Cookbook

I go on a lot of self hosting adventures , this book is a collection of recipes I get from all those adventures

Exposing Publically

This chapter contains ways you can expose your services publically , safely , securely and graciously

SSH Tunneling

What is SSH

SSH is a standard for secure remote logins and file transfers over untrusted networks . It is around all around the world by people for remote logins into cloud servers and their VPS.

What is SSH tunneling

SSH Tunneling is a safe and secure way to forward ports from your server to your machine locally. It is very similar to creating a VPN of some sort. Essentially can forward all the ports quickly and basically bring the other machine on your lan whereas SSH tunneling can forward one or more ports from the remote machine to yours.

How do I use it ?

The -L option can be used with your regular ssh command to create a tunnel . So for example you generally do

```
ssh -p 69 admin@blabla.com
```

Now lets say you want to get whatever is running on port 8080 of blabla.com on your machine you can do this

```
ssh -L 8080:127.0.0.1:8080 -p 69 admin@blabla.com
```

Now if you go to port 8080 on your machine you have got whatever was on the port 8080 of blabla.com

Advantages

- ## Don't have to disable or worry for firewalls

  You don't have to care about your firewall or patching holes in it whatsoever, If you can ssh into a machine - you can ssh tunnel

- ## Fast with no setup needed

  Just use the -L flag and you are gtg

- ## Secure by default

  Since it is just using the SSH protocol it doesn't have an glaring security holes in it that can be exploited

> As secure as this , it can quickly turn into something not so secure , more on that here {.is-info}

Exposing your service to the web using tunneling

As we saw that ssh tunneling can be used to direct whatever is running on the port to your local machine , you can also use this to overcome the carrier grade NAT setups that Indian ISPs use which essentially prevents you from opening any ports on your LAN to the web.

Ok so now what we are going to do is called Reverse SSH Tunneling which as the name suggests is the reverse of SSH Tunneling . In SSH Tunneling you basically get whatever is on the port of the remote machine to your machine but in Reverse Tunneling you get whatever is running on your machine to the remote machine's port

Assumptions

- Your service is running on port 8096
- You want to expose the service on subdomain.yourdomain.com
- You have an A record for subdomain.yourdomain.com pointed to your server
- Step 1 : Set up an SSH Tunnel So you can use this

```
ssh -R 8096:127.0.0.1:8096 admin@subdomain.yourdomain.com
```

or to give a more tunnely effect

```
ssh -N -R 8096:127.0.0.1:8096 admin@subdomain.yourdomain.com
```

What the second command does is that it establishes an ssh session but doesn't give you a shell
To persist this tunnel you can use several things -:

- Create a shell script that runs in the background and enable it on system startup
- Run this in a tmux or a screen sesssion in the background
- The best and the most effective way is to create a systemd service.

This is a sample systemd service that I personally use

```
[Unit]
Description=A reverse ssh tunnel

[Service]
User=ubuntu
ExecStart=ssh -N -R 8096:127.0.0.1:8096  blabla@blabla.com
Restart=always

[Install]
WantedBy=multi-user.target
```

```
```

after adding this to /etc/systemd/system as tunnel.service you can simply do

```
sudo systemctl start tunnel
```

Step 2 : Exposing on your domain

So now once you have your service on your server locally you can expose this using the reverse proxy setup guide

VPN setup

Server side

For the server side setup I recommend that people install wireguard using pivpn and not manually . The manual setup gives you bit more control but the pivpn install script has sane defaults for most basic setups.

So today we are going to be adhering to using pivpn for installation and setup of the server side

Using pivpn

Now pivpn originally is a concept that was meant for the raspberry pi but since the self hosting community has grown they have also adapted the project to fit a more wider range of servers .

> I have tested and this on both ubuntu and debian 10 machines but not on centos or any other OS . {.is-info}

So to use pivpn , you can do

```
curl -L https://install.pivpn.io > install.sh
```

This will download a script called install.sh on your machine. Now you go over this script and get a bird's eye view of what is up and what exactly is this script doing before running it . I have verified this script so I can vouch for this but you should read through it too

> Never run random scripts you got off the web directly unless it is provided by some big project that hundreds of people are using every day. Even then I would recommend atleast getting a bird's eye view of it {.is-warning}

So to run it just do

```
```

```
sh install.sh
```

Now this will ask you a bunch of questions, most times if you don't know the answer you can go with the default option and you will most probably get a working setup .

Client Side

Get yourself a client config file

Now to connect to your vpn the first most important thing is to get yourself a client-config. This config basically contains your client keys to connect and a bunch of other info to properly configure your clients. To get yourself a client config type in this command

```
pivpn add
```

This will ask you about the client name , this can be anything you are setting up so add that and it will generate a new config for you Now if you head to ~/configs you will the .conf file there with the name you typed in earlier

Now let's try to configure the three different types of clients

Winblows and MaxOs

These have GUI apps that you can get from here After you get the apps , you can then import the client configs into them and you are gtg.

Linux

Now for setting this up on a linux machine you just need to install the `wireguard-tools` package from the package manager of your choice.

After that you can copy the client config into `/etc/wireguard/wg0.conf`

Once that is done you can just start the systemd service

```
sudo systemctl start wg-quick@wg0.service
```

and if you want to persist your VPN connection across reboots you can do this

```
sudo systemctl enable wg-quick@wg0.service
```

For distros that don't use systemd,

```
wg-quick up wg0
```

Both these will basically add a new network interface called wg0 to your system and configure all the packets from that interface to head to the remote server.

Android

Now for android you can generate a client config and once you finish doing that it shows you a qr code, if you scan that using the wireguard app , you can add your tunnel painlessly.

> Please keep this config file secret and not share it with anyone. If anyone gets ahold of this file , they can connect to your VPN server and pretend that they are you. {.is-warning}

Self hosting setup

Now let's say that you have successfully setup wireguard on your linux server and connected it to the VPN (if you are running a windows server then I just have one question for you ,who hurt you sista ?), the next step is to find the ip of your machine on the wireguard network to do that , just execute

```
ip a | grep wg0
```

From this listing find out your IP address. Now you can do a standard reverse-proxy setup and the IP you just found out becomes your upstream IP and the port is the same one on which your application is running.

Things to do with a pi

Setup an HTPC server, pihole, host your cloud and a lot more

Your own Google services / Nextcloud

Now hosting nextcloud is a pretty vanilla setup , all you gotta do is get yourself a compose file and do a docker-compose up and you are pretty much good to go

Docker-compose file

```
version: "2.1"
services:
  nextcloud:
    image: linuxserver/nextcloud
    container_name: nextcloud
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Asia/Kolkata
    volumes:
      - /mnt/hdd/config:/config
      - /mnt/hdd/data:/data
    ports:
      - 993:993
      - 587:587
      - 9443:443
      - 8080:80
    restart: unless-stopped

    # PostgreSQL
  postgres:
    container_name: postgres
    environment:
      - POSTGRES_PASSWORD=PleaseChangeMeandEnterAGoodPasswordHere
```

```
      - POSTGRES_USER=nextcloud
    image: postgres:latest
    restart: always
    volumes:
      - "/mnt/hdd/postgres/init:/docker-entrypoint-initdb.d/"
      - "/mnt/hdd/postgres/data:/var/lib/postgresql/data"
      - "/etc/localtime:/etc/localtime:ro"
    # Redis
  redis:
    container_name: redis
    image: redis:latest
    restart: always
```

> This setup assumes that your disk is mounted at /mnt/hdd which you want to use for your data storage {.is-info}

- Now just do a docker-compose up -d
- While installation just select postgres as a backend with the following details

Nginx config

Here is the nginx config in case you want to make this available publically

```
server {
    listen 80;
    server_name <domain>;
    return 301 https://$host$request_uri;  # redirect http to https
}

server {
    listen 443 ssl http2;
    server_name <domain>;

    ssl_certificate     /etc/letsencrypt/live/<domain>/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/<domain>/privkey.pem;
```

```
    port_in_redirect off;
    proxy_buffering off;
    proxy_http_version 1.1;     # Properly proxy websocket connections
    proxy_read_timeout 300s;    # terminate websockets afer 5min of inactivity
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header X-Forwarded-Protocol $scheme;


    location / {
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-Proto https;
            proxy_pass  http://<ip>:<port>
    }
}
```

Media streaming on your pi / Your own netflix

Basic setup

> The guide here assumes that you are running Ubuntu server 20.04.02 LTS on Raspberry pi 4 with 4gb of RAM . Though since most of this is dockerized it can be replicated on other setups as well {.is-info}

```
version: "2.1"
services:
  jellyfin:
    image: ghcr.io/linuxserver/jellyfin:nightly
    container_name: jellyfin
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Asia/Kolkata
    volumes:
      - /mnt/hdd/media/config:/config
      - /mnt/hdd/media/tvs:/data/tvshows
      - /mnt/hdd/media/movies:/data/movies
      - /mnt/hdd/media/music:/data/music
      - /opt/vc/lib:/opt/vc/lib #optional
    ports:
      - 8096:8096
      - 8920:8920 #optional
    devices:
      - /dev/dri:/dev/dri #optional
      - /dev/vchiq:/dev/vchiq #optional
```

```
      - /dev/video10:/dev/video10 #optional
      - /dev/video11:/dev/video11 #optional
      - /dev/video12:/dev/video12 #optional
    restart: unless-stopped
```

Here are some things you can change

- ghcr.io/linuxserver/jellyfin:nightly to ghcr.io/linuxserver/jellyfin:latest if you don't want to live on the bleeding edge of things like me
- volumes ofcourse need to be changed /mnt/hdd is the local volume where my hdd is mounted on my machine you need to change this to match where your music, TV shows and Movies are located accordingly
- the ports can be changed so you can change 8096:8096 to 8080:8096 that way it would be available on your machine on port 8080
- 8920 is an https port if you want https on the level of jellyfin but since this service is running on our home network it doesnt absolutely need to be on https.
- Coming to devices all the devices you can see here are used for hardware acceleration. If you want to use hardware acceleration(which I recommend using) you can mount these devices onto the container. Below I will provide the explaination for hardware acceleration in detail

Basic steps for setup

- Create a directory called jellyfin
- Paste this compose file in a file called docker-compose.yml(make sure that directory doesn't contain any other docker-compose.yml files)
- docker-compose up -d
- Go to <ip-addr of your pi>:8096 (or whatever port you set in the compose-file)
- Follow the setup

Hardware Acceleration and do you need it ?

Hardware acceleration in the most layman of terms means using something for something that was it specialized to do. For example a GPU or Graphics Procesing Unit does what it says on the box. It processes graphics . Now you can do the same thing on your CPU but you wouldn't get the same performence from it. A GPU is specefically meant to process graphics hence the perfomance.

Hardware acceleration in case of jellyfin comes in use when you want to transcode the video . Transcoding means decoding from one format into another and then encoding it again . So for example a lot of browsers dont support H.265 encoded video and if you ask jellyfin to play that directly it will refuse . It will instead transcode it into another format and then send the video to your browser.

The transcoding part is where you need to Hardware acceleration and trust me when you are trying to transcode a big video hardware acceleration really really helps.

How to enable hardware acceleration on Ubuntu 20 on raspi

Hardware acceleration is disabled by default since the pi isn't really meant to be used as a device which might be used for graphics intensive purposes such as playing games.

> Try doing and ls /dev/dri if you see some devices here and then hardware acceleration is already enabled and you don't need to do anything else {.is-info}

Add this to `/boot/firmware/usercfg.txt`

```
# Place "config.txt" changes (dtparam, dtoverlay, disable_overscan, etc.) in
# this file. Please refer to the README file for a description of the various
# configuration files on the boot partition.
dtoverlay=vc4-fkms-v3d
max_framebuffers=2
gpu_mem=128
hdmi_enable_4kp60=1
```

Now reboot your pi and you should be good to go

Enable hardware accel. in Jellyfin

To do that you shoudl go to Dashboard -> Playback and then select Video Acceleration API from the dropdown under hardware acceleration Rest everything can be default.

Exposing Jellyfin to the web

The not so secure way

- The easiest but not so secure way to expose jellyfin to the web is to open ports in your router and get a DDNS setup . Then you can access jellyfin over your domain:8096 , which btw looks straight up ugly
- You can also open port 80 and 443 and use nginx and certbot to use as reverse proxy and then expose jellyfin to the web over ssl using certbot(letsencrypt) (Only applicable in case your ISP allows you to do that which most ISPs don't)

The more secure way

- Patching holes in the firewall that protects your home network isn't considered very secure but there is another way
- Get a small cloud VM from whatever provider and host a VPN on it and then that way you will be easily able to connect your pi to this VPN and serve whatever content onto the dark place we call internet
- When you have a VPN configured you can use nginx as a reverse proxy to serve that via ssl

Sample configuration for nginx

Here is a sample configuration for nginx if you are using the VPN route

```
server {
    listen 443 ssl;
    listen [::]:443 ssl;

    server_name <website domain here>;

    #include /config/nginx/ssl.conf;
    ssl_trusted_certificate /etc/letsencrypt/live/<domainhere>/chain.pem;
    ssl_stapling on;
    ssl_stapling_verify on;
```

```
    ssl_certificate /etc/letsencrypt/live/<domainhere>/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/<domainhere>/privkey.pem; # managed by
Certbot

    client_max_body_size 0;

    location / {
        include /etc/nginx/proxy.conf;
        #resolver 127.0.0.11 valid=30s;
        set $upstream_app <ip of your jellyfin server>;
        set $upstream_port 8096;
        set $upstream_proto http;
        proxy_pass $upstream_proto://$upstream_app:$upstream_port;

        proxy_set_header Range $http_range;
        proxy_set_header If-Range $http_if_range;
    }
    # to be able to use syncplay over nginx
    location ~ (/jellyfin)?/socket {
        include /etc/nginx/proxy.conf;
        #resolver 127.0.0.11 valid=30s;
        set $upstream_app <ip of your jellyfin server>;
        set $upstream_port 8096;
        set $upstream_proto http;
        proxy_pass $upstream_proto://$upstream_app:$upstream_port;

    }
}
```

> Go through the nginx config file and change whatever needs to be changed this will not work as is {.is-warning}

Here is the proxy.conf file save this in `/etc/nginx/proxy.conf`

```
proxy_next_upstream error timeout invalid_header http_500 http_502 http_503;

# Proxy Connection Settings
proxy_buffers 32 4k;
proxy_connect_timeout 240;
```

```
proxy_headers_hash_bucket_size 128;
proxy_headers_hash_max_size 1024;
proxy_http_version 1.1;
proxy_read_timeout 240;
proxy_redirect  http://  $scheme://;
proxy_send_timeout 240;

# Proxy Cache and Cookie Settings
proxy_cache_bypass $cookie_session;
#proxy_cookie_path / "/; Secure"; # enable at your own risk, may break certain apps
proxy_no_cache $cookie_session;

# Proxy Header Settings
proxy_set_header Host $host;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Host $host;
proxy_set_header X-Forwarded-Proto https;
proxy_set_header X-Forwarded-Ssl on;
proxy_set_header X-Real-IP $remote_addr;
```

A gentle introduction to docker and docker-compose

Containerize everything !!!

Misc Recipies

Recipies for other things I have self hosted in the past

Your own VPN

- This guide will help you set up wireguard VPN with pihole as the DNS server.

What is pi-hole

Pi-hole is basically a DNS level adblocker. Now for people who are new to this,

What is DNS

- DNS stands for Domain Name Server and is basically the backbone of the modern internet. The internet works on IP addresses ie if you go 172.217.167.238 using your browser , you would be directed to google.com. Beautiful right ?
- But as humans we can't be expected to remember these ip addresses, This is where DNS comes in .So when you go to google.com on your browser , your browser asks the DNS server you have set - What is the Ip of google.com and it gets a response and hence it takes you to that ip.
- Here is a more detailed guide on how DNS works

What is Pi-hole then ?

- When you load up a site with ads, it phones home on several different domains for ads and when you use regular DNS servers, they just allow those domains to resolve .
- Now when you change your DNS to pi-hole , as soon as a site phones home for ads pi-hole blocks that DNS request and that's how the ads get blocked.

What is wireguard

- Wireguard is this new age VPN technology that consumes less battery, processing power than openvpn and is just plain faster.

What are we going to setup today ?

We are basically going to setup your own VPN server which will have it's dns set as pihole and hence give you adblocking whenever you connect to your VPN

Docker-compose file

For this setup we are going to use docker-compose paste the below into a docker-compose.yml file and then run

```
docker-compose up -d
```

in the same directory where you have saved this file

docker-compose.yml

```
version: "3"

networks:
  private_network:
    ipam:
      driver: default
      config:
        - subnet: 10.2.0.0/24

services:
  unbound:
    image: "klutchell/unbound:latest"
    container_name: unbound
    restart: unless-stopped
    hostname: "unbound"
    volumes:
      - "./unbound:/opt/unbound/etc/unbound/"
    networks:
```

```
    private_network:
      ipv4_address: 10.2.0.200

wireguard:
  depends_on: [unbound, pihole]
  image: linuxserver/wireguard
  container_name: wireguard
  cap_add:
    - NET_ADMIN
    - SYS_MODULE
  environment:
    - PUID=1000
    - PGID=1000
    - TZ=America/Los_Angeles # Change to your timezone
    - SERVERPORT=51820
    #- SERVERURL=my.ddns.net #optional - For use with DDNS (Uncomment to use)
    - PEERS=1 # How many peers to generate for you (clients)
    - PEERDNS=10.2.0.100 # Set it to point to pihole
    - INTERNAL_SUBNET=10.6.0.0

  volumes:
    - ./wireguard:/config
    - /lib/modules:/lib/modules
  ports:
    - "51820:51820/udp"
  dns:
    - 10.2.0.100 # Points to pihole
    - 10.2.0.200 # Points to unbound
  sysctls:
    - net.ipv4.conf.all.src_valid_mark=1

  restart: unless-stopped
  networks:
    private_network:
      ipv4_address: 10.2.0.3

pihole:
  depends_on: [unbound]
  container_name: pihole
```

```
    image: pihole/pihole:latest
    restart: unless-stopped
    hostname: pihole
    dns:
      - 127.0.0.1
      - 8.8.8.8 # Points to unbound
    environment:
      TZ: "America/Los_Angeles"
      WEBPASSWORD: "" # Blank password - Can be whatever you want.
      ServerIP: 10.1.0.100 # Internal IP of pihole
      DNS1: 8.8.8.8 # Unbound IP
      DNS2: 8.8.8.8 # If we don't specify two, it will auto pick google.
    # Volumes store your data between container upgrades
    volumes:
      - "./etc-pihole/:/etc/pihole/"
      - "./etc-dnsmasq.d/:/etc/dnsmasq.d/"
    # Recommended but not required (DHCP needs NET_ADMIN)
    #   https://github.com/pi-hole/docker-pi-hole#note-on-capabilities
    cap_add:
      - NET_ADMIN
    networks:
      private_network:
        ipv4_address: 10.2.0.100
```

Step 2

- Now that you have the containers running , just do a docker-compose logs -f and you will see a qr code.
- Install the wireguard android app and scan this qr code to add this tunnel
- Now try connecting to the tunnel
- Congrats you have a vpn server now which has adblocking built into it

Configuring a split tunnel

- Wireguard also lets you configure something called a split tunnel which in my opinion is an amazing feature.
- So when you connect to your vpn server, your internet speed is bottlenecked by the speed of

the VPN server.
- To get fast speeds and adblocking as well you can configure what is called a split tunnel, which means that only your dns queries will be sent to the server and all the other queries will be routed directly
- To do that change the AllowedIPs in the wireguard config to `10.2.0.0/24`.

Troubleshooting

- For troubleshooting the first step is to check if all containers are still alive `docker-compose logs -f`
- Now if the containers are running properly, check if port 51820/udp is accessible over the internet.

Discourse

Step 1: Getting and building the docker image

- `git clone https://github.com/discourse/discourse_docker.git`
- Now the official way is to run ./discourse-setup which will ask you a bunch of questions and then run discourse for you . But the very basic requirement for that is that ports 80 and 443 should be open and thats not the case in this guid
- Now to change how the official docker image builds you need to copy samples/standalone.yml to containers/app.yml
- So `cp samples/standalone.yml containers/app.yml`
- Go into the app.yml file and change the expose statement to `- "3001:80   # http`
- Scroll down and change the SMTP settings

> If you do not add the SMTP settings the container will not build {.is-warning}

- Now run the launcher script and rebuild the app container `./launcher rebuild app`

Sample app.yml file

```
## this is the all-in-one, standalone Discourse Docker container template
##
## After making changes to this file, you MUST rebuild
## /var/discourse/launcher rebuild app
##
## BE *VERY* CAREFUL WHEN EDITING!
## YAML FILES ARE SUPER SUPER SENSITIVE TO MISTAKES IN WHITESPACE OR ALIGNMENT!
## visit http://www.yamllint.com/ to validate this file as needed
```

```
templates:
  - "templates/postgres.template.yml"
  - "templates/redis.template.yml"
  - "templates/web.template.yml"
  - "templates/web.ratelimited.template.yml"
## Uncomment these two lines if you wish to add Lets Encrypt (https)
  #- "templates/web.ssl.template.yml"
  #- "templates/web.letsencrypt.ssl.template.yml"

## which TCP/IP ports should this container expose?
## If you want Discourse to share a port with another webserver like Apache or nginx,
## see https://meta.discourse.org/t/17247 for details
expose:
  - "3001:80"   # http


params:
  db_default_text_search_config: "pg_catalog.english"

  ## Set db_shared_buffers to a max of 25% of the total memory.
  ## will be set automatically by bootstrap based on detected RAM, or you can override
  #db_shared_buffers: "256MB"

  ## can improve sorting performance, but adds memory usage per-connection
  #db_work_mem: "40MB"

  ## Which Git revision should this container use? (default: tests-passed)
  #version: tests-passed

env:
  LC_ALL: en_US.UTF-8
  LANG: en_US.UTF-8
  LANGUAGE: en_US.UTF-8
  # DISCOURSE_DEFAULT_LOCALE: en

  ## How many concurrent web requests are supported? Depends on memory and CPU cores.
  ## will be set automatically by bootstrap based on detected CPUs, or you can override
  #UNICORN_WORKERS: 3

  ## TODO: The domain name this Discourse instance will respond to
```

```
  ## Required. Discourse will not work with a bare IP number.
  DISCOURSE_HOSTNAME: 'discourse.example.com'

  ## Uncomment if you want the container to be started with the same
  ## hostname (-h option) as specified above (default "$hostname-$config")
  #DOCKER_USE_HOSTNAME: true

  ## TODO: List of comma delimited emails that will be made admin and developer
  ## on initial signup example 'user1@example.com,user2@example.com'
  DISCOURSE_DEVELOPER_EMAILS: 'me@example.com,you@example.com'

  ## TODO: The SMTP mail server used to validate new accounts and send notifications
  # SMTP ADDRESS, username, and password are required
  # WARNING the char '#' in SMTP password can cause problems!
  DISCOURSE_SMTP_ADDRESS: smtp.example.com
  #DISCOURSE_SMTP_PORT: 587
  DISCOURSE_SMTP_USER_NAME: user@example.com
  DISCOURSE_SMTP_PASSWORD: pa$$word
  #DISCOURSE_SMTP_ENABLE_START_TLS: true           # (optional, default true)
  #DISCOURSE_SMTP_DOMAIN: discourse.example.com    # (required by some providers)
  #DISCOURSE_NOTIFICATION_EMAIL: noreply@discourse.example.com    # (address to send
notifications from)

  ## If you added the Lets Encrypt template, uncomment below to get a free SSL certificate
  #LETSENCRYPT_ACCOUNT_EMAIL: me@example.com

  ## The http or https CDN address for this Discourse instance (configured to pull)
  ## see https://meta.discourse.org/t/14857 for details
  #DISCOURSE_CDN_URL: https://discourse-cdn.example.com

  ## The maxmind geolocation IP address key for IP address lookup
  ## see https://meta.discourse.org/t/-/137387/23 for details
  #DISCOURSE_MAXMIND_LICENSE_KEY: 1234567890123456

## The Docker container is stateless; all data is stored in /shared
volumes:
  - volume:
      host: /var/discourse/shared/standalone
      guest: /shared
  - volume:
```

```
      host: /var/discourse/shared/standalone/log/var-log
      guest: /var/log

## Plugins go here
## see https://meta.discourse.org/t/19157 for details
hooks:
  after_code:
    - exec:
        cd: $home/plugins
        cmd:
          - git clone https://github.com/discourse/docker_manager.git

## Any custom commands to run after building
run:
  - exec: echo "Beginning of custom commands"
  ## If you want to set the 'From' email address for your first registration, uncomment and
change:
  ## After getting the first signup email, re-comment the line. It only needs to run once.
  #- exec: rails r "SiteSetting.notification_email='info@unconfigured.discourse.org'"
  - exec: echo "End of custom commands"
```

Configuring Nginx

Sample reverse proxy file

Assuming discourse is running on port 3001. Here is a sample nginx reverse proxy file

```
server {
    listen 80;
    server_name ;
    return 301 https://$host$request_uri;  # redirect http to https
}

server {
    listen 443 ssl http2;
    server_name ;
```

```
    port_in_redirect off;
    proxy_buffering off;
    proxy_http_version 1.1;     # Properly proxy websocket connections
    proxy_read_timeout 300s;    # terminate websockets afer 5min of inactivity
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header X-Forwarded-Protocol $scheme;



    location / {
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-Proto https;
            proxy_pass  http://0.0.0.0:3001;
        }
}
```

Make sure to add the ssl configuration to this or else it won't work

Now just reload nginx and you are good to go

```
sudo nginx -s reload
```

Gitea

To run gitea , you need to have the gitea container itself and then you need to have a database container alongside, you can use sqlite but as usual it is not recommended so here in this we are going to use mysql , you can also use postgresql if you want

Docker-compose file

```
version: '2'
volumes:
  app:
  db:
services:
  app:
    container_name: gitea-app
    restart: always
    image: gitea/gitea:${GITEA_VERSION}
    links:
      - db:mysql
    volumes:
      - ./volumes/gitea_app:/data
      - /home/git/.ssh/:/data/git/.ssh
    ports:
      - "${GITEA_WEB_PORT}:3000"
      - "127.0.0.1:2222:22"
    environment:
      - VIRTUAL_PORT=3000
      - VIRTUAL_HOST=${GITEA_HOSTNAME}
    networks:
      - backend
      - frontend
  db:
    container_name: gitea-db
    restart: always
```

```
    image:  mysql:5.6
    volumes:
      - ./volumes/gitea_db:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
      - MYSQL_DATABASE=${MYSQL_DATABASE}
      - MYSQL_USER=${MYSQL_USER}
      - MYSQL_PASSWORD=${MYSQL_PASSWORD}
    networks:
      - backend

networks:
  frontend:
  backend:
```

apart from this you need to have a .env file to store all your secrets etc

```
GITEA_VERSION=latest
GITEA_HOSTNAME=<domain>
GITEA_WEB_PORT=300
GITEA_SSH_PORT=2222
MYSQL_ROOT_PASSWORD=<choose a good password>
MYSQL_DATABASE=gitea
MYSQL_USER=gitea
MYSQL_PASSWORD=<choose a good password>
```

Now once you do a docker-compose up -d you will have a working gitea setup. To expose this to the web you can do two things

- Open port 3000 and expose it over http
- Use nginx to serve it over https

One of the above is not a good thing to do . can you guess which one ?

Reverse proxy setup

```
server {
    listen 80;
```

```
    server_name <domain-here>;
    return 301 https://$host$request_uri;  # redirect http to https
}

server {
    listen 443 ssl http2;
    server_name <domain-here>;

    ssl_certificate     /etc/letsencrypt/live/<domain-here>/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/<domain-here>/privkey.pem;

    port_in_redirect off;
    proxy_buffering off;
    proxy_http_version 1.1;     # Properly proxy websocket connections
    proxy_read_timeout 300s;    # terminate websockets afer 5min of inactivity
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header X-Forwarded-Protocol $scheme;


    location / {
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-Proto https;
            proxy_pass  http://localhost:3000;
    }
}
```

Now if you want to use ssh inside the container , you can do any of the three things below -:

- Run your local ssh on some other port and open port 22
- Run the containers ssh on some other port
  - Even though this looks tempting to do , it really messes up the ssh clone url by adding an additional port to it . Not to mention you have to open another port to the web
- Setup SSH container passthrough. I would strongly recommend setting up the passthrough since you don't have to do any of the above

SSH passthrough

> Here I am assuming that you have a user name git locally which has the same PUID and GUID inside the container {.is-info}

> It is strongly recommended to have this git user for ssh since having any other username would also mess up the domain name you get for cloning and adding remote etc {.is-warning}

1. Now the first step is to mount `/home/git/,ssh/` inside the container Add this to the docker-compose

```
volumes:
- /home/git/.ssh/:/data/git/.ssh
```

2. Next get yourself an ssh key pair this will be used to authenticate the git user on the host to the container

```
sudo -u git ssh-keygen -t rsa -b 4096 -C "Gitea Host Key"
```

3. Now create a file name `/app/gitea/gitea` on the host . This is basically going to issue ssh forwarding from the container . Add the following to this file and make it executable

```
ssh -p 2222 -o StrictHostKeyChecking=no git@127.0.0.1
"SSH_ORIGINAL_COMMAND=\"$SSH_ORIGINAL_COMMAND\" $0 $@"
```

Now to make ssh forwarding work port 22 on the container needs to be mapped to the host port 2222. Since this doesn't need to be exposed outside , you can map this to the localhost of the container

```
ports:
  # [...]
  - "127.0.0.1:2222:22"
```

Also now the /home/git/,ssh/authorized_keys on the host needs to be modified in order to act the same way as the authorized_keys on the container . To do that , just do

```

```

```
echo "$(cat /home/git/.ssh/id_rsa.pub)" >> /home/git/.ssh/authorized_keys
```

The file should look like

```
ssh-rsa <Gitea Host Key>

# other keys from users
command="/app/gitea/gitea --config=/data/gitea/conf/app.ini serv key-1",no-port-forwarding,no-X11-forwarding,no-agent-forwarding,no-pty <user pubkey>
```

Introduction to self-hosting

What is this ?

- Self hosting is the art of hosting all your SaaS services on your own since huge corporations can't really be trusted.
- What follows in this section of the wiki are some not-so-comprehensive guides to self hosting your own SaaS at home or with some cloud provider

Why I insist on calling these recipes

For me self-hosting something is like a dish made in the kitchen. Every dish can have good or bad recipes, and your cooking skills/recipes get better when you spend more time in the kitchen. For example Sanjeev Kapoor wouldn't see a generic recipe for making butter chicken would he ? Similarly for self hosting you have the generic recipes ie the documentation for the project but sometimes you need something that is tailored to your style of self-hosting (ie recipes made by other people). Hence this is a book of recipes.

The point of self hosting

- A web server just has one purpose to serve static content . It's not really meant to process stuff for you, but with the advent of the web we have also seen the advancement of something called Software as a Service or SaaS.
- When you consume a SaaS , the server typically asks for some data , processes it for you and then gives you back. Some examples of SaaS include facebook,instagram,github,gitlab,basically all things hosted by google etc.
- The problem with proprietary SaaS is that you have no idea what is running on the server and what it is doing to your data, atleast if the SaaS is Free software, you have some idea what the server is doing to your data so you can have some re-assurance
- The best way hence in my opinion is to self host your SaaS as much as you can. Maybe if you have enough funds host some Services for your friends too. :)

A short list of SaaS providers I trust

- nixnet.services
- fsci.in
- disroot.org
- autistici.org
- jit.si
- Friends I personally know
- Myself

> I am a just a student of life documenting what I learn . I am not responsible for south park not releasing season 26 or your server burning down. Please copy stuff at your own risk {.is-warning}