Если вы видите что-то необычное, просто сообщите мне.

Docker/docker-compose

- Code-server
- Selfhosted Sentry
- GlusterFS Setup
- Gocd
- Install Fedora 37 or earlier on Windows Subsystem for Linux (WSL)
- Minio cluster s3fs
- Docker registry mirror

Code-server

code-server

взято отсюда:

https://hub.docker.com/r/codercom/code-server

docker-compose.yml - можно использовать или build или image

```
version: "2.1"
services:
  code-server:
    #image: codercom/code-server:latest
    build: .
    container_name: code-server
    #command: export PATH=$HOME/project/ghc/.cabal/bin:$PATH
    user: 1000:1000
    volumes:
      - ./project:/home/coder/project
      - ./.stack:/home/coder/.stack
      - ./config.yaml:/home/coder/.config/code-server/config.yaml
    ports:
      - 8080:8080
      - 8000:8000
```

Dockerfile - тут можно либо использовать готовый образ либо добавить в него всё что нужно, на стадии разворачивания

```
FROM codercom/code-server:latest

RUN sudo apt update && sudo apt install -y ghc libopenblas-dev
RUN curl -sSL https://get.haskellstack.org/ | sh
```

config.yaml - настройки code-server

```
bind-addr: 127.0.0.1:8080
auth: password
password: НУЖНЫЙПАРОЛЬ
cert: false
```

Selfhosted Sentry

Первый запуск:

```
docker-compose run sentry sentry upgrade
```

dokcer-compose.yml

```
version: '2'



services:
    redis:
      image: redis

    postgres:
      image: postgres
      env_file:
       -  .env
      volumes:
       - ./pgdb:/var/lib/postgresql/data

    sentry:
      image: sentry:8.10.0
      links:
       - redis
       - postgres
      ports:
       - 8010:9000
      env_file:
       -  .env

    cron:
      image: sentry:8.10.0
      links:
       - redis
       - postgres
```

```
    command: "sentry run cron"
    env_file:
     -  .env

  worker:
    image: sentry:8.10.0
    links:
     - redis
     - postgres
    command: "sentry run worker"
    env_file:
     -  .env
```

.env

```
POSTGRES_USER=sentryuser
POSTGRES_PASSWORD=!!!SETPASSWORD!!!
POSTGRES_DB=sentry
SENTRY_SECRET_KEY=!!!SETSECRETKEY!!!
SENTRY_POSTGRES_HOST=postgres
SENTRY_DB_USER=sentryuser
SENTRY_DB_PASSWORD=!!!SETPASSWORD!!!
SENTRY_REDIS_HOST=redis
SENTRY_EMAIL_HOST=smtp.yandex.ru
SENTRY_EMAIL_PORT=25
SENTRY_EMAIL_PASSWORD=!!!SETPASSWORD!!!
SENTRY_EMAIL_USER=!!!SETUSER!!
SENTRY_EMAIL_USE_TLS=true
SENTRY_SERVER_EMAIL=!!!SETUSERMAIL!!!
```

GlusterFS Setup

Требования GlusterFS поддерживает только 64bit системы, поэтому убедитесь, что хостовая машина может запустить GlusterFS и любые другие машины используют тоже 64bit системы.

Эта инструкция подоходит для Ubuntu 22.04 jammy

Инструкция Запустите следующие команды на всех системах, которые будут использоваться для распределенной файловой системы.

Добавление хостов в `/etc/hosts`

Мы хотим убедиться что наша машина может общаться друг с другом по именам, это можно сделать отредактировав следующим образом:

Редактируем `/etc/hosts`

```
sudo nano /etc/hosts
```

Добавьте в него ваши адреса которые будут использоваться машинами, ниже приведен пример для моей машины GlusterFS, но такие записи нужно будет добавить на всех машинах где будет использоваться GlusterFS

```
127.0.0.1 localhost
127.0.1.1 elzim

192.168.68.109  elzim
192.168.68.105  pi4lab01
192.168.68.114  pi4lab02
```

Установка GlusterFS

Настроим GlusterFS репозиторий. На время написания статьи GlusterFS-10 последний релиз.

```
sudo add-apt-repository ppa:gluster/glusterfs-10
```

Запускаем обновление репозиториев.

```
sudo apt update
```

```
sudo apt install glusterfs-server -y
```

Запускаем и включаем GlusterFS

```
sudo systemctl start glusterd
sudo systemctl enable glusterd
```

Связываем ноды Это команда запускается только на хостовой машине.

Перед запуском команды связывания, убедитесь, что вы запускаете команды от sudo.

```
sudo -s
```

Следующая команда будет связывать все ноды в кластер GlusterFS, он использует имена указанные в hosts, убедитесь, что вы внесли необходимые изменения в скрипт.

```
gluster peer probe pi4lab01;
gluster peer probe pi4lab02;
```

Запустите команды которые покажут связанные хосты в кластер.

```
sudo gluster pool list
```

Создание Gluster раздела

Давайте создадим дирректорию, которая будет использоваться в качестве раздела GlusterFS

> “ Команда ниже создается на всех нодах кластера. Note: You can name "volumes" to anything you like.

```
sudo mkdir -p /gluster/volumes
```

Теперь мы можем создать раздел на всех нодах кластера. Команда выполняется на хосте.

```
sudo gluster volume create staging-gfs replica 3 elzim:/gluster/volumes
pi4lab01:/gluster/volumes pi4lab02:/gluster/volumes force
```

Запустим раздел запустив команду ниже

```
sudo gluster volume start staging-gfs
```

Чтобы убедиться, что раздел автоматически примонтируется при перезагрузе или других обстоятельствах, нужно выполнить следующие шаги на всех машинах:

Переключитесь на супер пользователя:

```
sudo -s
```

Добаврьте следующие строки в /etc/fstab файл используя команду

```
echo 'localhost:/staging-gfs /mnt glusterfs defaults,_netdev,backupvolfile-server=localhost 0
0' >> /etc/fstab
```

Примонтируйте раздел GlusterFS к /mnt директории с помощью команды:

```
mount.glusterfs localhost:/staging-gfs /mnt
```

Установите владельца /mnt дирректории и его содержимого root:docker используя команду:

```
chown -R root:docker /mnt
```

Выйдите из рута

```
exit
```

Чтобы проверить, что GlusterFS раздел успешно смонтирован, запустите команду:

```
df -h
localhost:/staging-gfs                 15G  4.8G  9.1G  35% /mnt
```

Файлы созданные в папке /mnt будут теперь отображаться во всех машинах где запущен GlusterFS.

Gocd

```
version: "3"
services:

  server:
    image: gocd/gocd-server:v23.3.0
    ports:
      - "8153:8153"
      - "8154:8154"
    volumes:
      - ./data:/godata
      - ./data/home:/home/go
      - ./scripts/server:/docker-entrypoint.d
      - ./scripts/shared:/shared
    environment:
      - GO_SERVER_SYSTEM_PROPERTIES=-Dcruise.material.update.interval=2000 -
Dmaterial.update.idle.interval=2000
    depends_on:
      - "gitserver"

  agent:
    image: gocd/gocd-agent-docker-dind:v23.3.0
    environment:
      - GO_SERVER_URL=http://server:8153/go
      - AGENT_AUTO_REGISTER_KEY=agent-autoregister-key
    volumes:
      - ./scripts/agent:/docker-entrypoint.d
      - ./scripts/shared:/shared
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on:
      - "server"
```

Install Fedora 37 or earlier on Windows Subsystem for Linux (WSL)

Using Windows Subsystem for Linux (WSL), we have a choice of Linux distributions available in the Windows Store, such as Ubuntu, Kali, Debian, etc.

In addition to these, with WSL 2, installing custom distributions is fairly straightforward, even if they are not in the Windows Store:

- find a rootfs for the distro
- install with wsl --import in Windows Powershell or Command Prompt Lately I rely on Fedora in WSL 2, and have been thoroughly pleased with that distribution. This article details the steps I use to get up and running with Fedora on WSL. By the way, if you prefer a nice out-of-box experience, take a look at the fine work by Whitewater Foundry with their Fedora Remix for WSL. But if you see "some assembly required" as an enticing invitation, read on.

> “ Already have a previous version of Fedora on WSL 2, and just want to upgrade to the latest? I documented the steps involved in another article; feel free to take a look.

Prerequisite: WSL 2

Please note that these steps require WSL 2 (not version 1).

> To run WSL 2, Windows version 1903 or higher is needed, with Build 18362 or higher. Most likely, you have a much later version than this already. To tell what version you are running, run winver in Powershell or CMD, or just type Win key and R (⊞-r) to open the Run dialog and then enter winver. Hopefully you will see something like "Version 21H2. OS Build 19044.1503". If on Windows 11, you needn't worry at all anyway, but the version should be something like "Version 21H2. OS Build 22000.1165".

To check if you are running WSL 2, try this command:

```
wsl --set-default-version 2
```

This will set the default version to WSL 2, or fail if you are still on the first version.

If it does neither and instead offers installation instructions, this may be your first time using WSL on this system.

Microsoft offers helpful installation instructions, including step-by-step instructions on how to upgrade to WSL 2.

In short, if you are comfortable installing another Linux distribution, running wsl --install will install the default latest Ubuntu, or you can pick from a list by using wsl -l -o then installing one with wsl --install -d Debian or, if not Debian, whichever distribution you would like.

If you are on an older version of Windows 10 or using WSL version 1, then the following Powershell command should get you to where you need (a reboot will likely be necessary).

```
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-
Linux,VirtualMachinePlatform -All
```

Most likely, the wsl command will then instruct you to install a new kernel, so proceed to https://aka.ms/wsl2kernel and follow the instructions there.

Again, see Microsoft's instructions for a current and detailed version.

Obtain a rootfs image of Fedora

We first need a rootfs (the base filesystem for the Linux distro in question, in this case Fedora). There a few ways to get this.

Obtain rootfs from Fedora Container Base

The rootfs embedded in the Fedora Container Base project seems to work well, and is my preferred method, as it offers a lot of flexibility. I start with the most recent stable version of Fedora that auto-built successfully. If you want to be bleeding edge, you can download the latest Rawhide image that built successfully, but I am using the latest Fedora 37 for now. The "State" column indicates build success. Pick the latest with a green checkmark.

Find the right "xz" file for your platform (likely x86_64). Such as:

- Fedora 37, built on October 29, 2022.
- Fedora 36, built on October 29, 2022.
- Fedora 35 built on October 29, 2022.

Unpack the Fedora-Container-Base-*.tar.xz file in your preferred manner. You might try 7-zip for this, extracting the .tar file, then extracting the contents of the .tar file. This, however, is not your rootfs. Further work is needed.

Once unpacked you will see a folder with a long hexadecimal name. Within that folder, there should be a layer.tar file. This is your rootfs. Copy the layer.tar file to a logical location, such as your Downloads folder. You may even want to rename it to something like fedora-33-rootfs.tar.

Alternative: download rootfs from docker-brew-fedora project

The docker-brew-fedora project imports "the official Fedora Docker Base Images built in Fedora Koji (Fedora's Build System) so that they may be submitted to the official-images repository for the Docker Hub."

To download, first pick your desired Fedora version from the active branches. For example, you might choose Fedora 37 and there find a file with a name like fedora-37-x86_64.tar.xz. Or use Fedora 36 and look for fedora-36-x86_64.tar.xz. Or go for Fedora 35 and locate fedora-35-x86_64.tar.xz. Download that file.

Unpack the fedora-3?-x86_64.tar.xz file in your preferred manner. You might try 7-zip for this, extracting the .tar file, then, if desired, renaming it to something like fedora-36-rootfs.tar.

Another rootfs alternative: use docker or podman and export

While this requires an extra tool, if you already have docker or podman available, then you can pull the Fedora image of your choosing and export the rootfs. In the following, if you have a Linux distro with podman available, you can substitute podman in place of docker.

```
docker run --name fedora37 fedora:37
docker export -o fedora-37-rootfs.tar fedora37
```

First we create the container, name it "fedora37", then export the rootfs as "fedora-37-rootfs.tar". Afterward, you can certainly docker rm fedora37 to clean up.

You may be interested in my articles for configuring podman on WSL, or setting up Docker on WSL.

Make a folder for your WSL distro.

Once we have the rootfs, we can prepare to import it.

I like to use wsl in my home directory, so in this case I create that folder and a fedora folder within it. In Powershell, that's:

```
mkdir $HOME\wsl\fedora
```

Install a new WSL Fedora distro

In Powershell, assuming you want to name the distro "fedora" and the folder is $HOME\wsl\fedora and the rootfs is in Downloads, named "fedora-37-rootfs.tar":

```
wsl --import fedora $HOME\wsl\fedora $HOME\Downloads\fedora-37-rootfs.tar
```

View installed distros

If this is the only WSL distro you now have, executing wsl -l should look something like this:

```
PS C:\Users\me> wsl -l
Windows Subsystem for Linux Distributions:
fedora (Default)
```

Launch Fedora as root

```
wsl -d fedora
```

Or, if Fedora is the default, simply wsl should result in a BASH prompt.

If you have multiple distros installed, and want Fedora to be set as the default, something like this should work:

```
wsl -s fedora
```

Ensure DNS is functioning (skip this section if network functionality is good)

DNS in WSL is interesting. By default, WSL will set the DNS server automatically, dynamically configuring /etc/resolv.conf. If your dns is resolving fine (does sudo dnf upgrade work for you?) then you can skip to the next section. It seems that it just works for most people.

For me, however, the dynamic /etc/resolv.conf has not worked consistently. I need to turn it off and configure resolv.conf manually.

Conveniently, WSL provides a means in /etc/wsl.conf to set some configuration settings specific to WSL.

In order for DNS to work, we will create our own resolv.conf, but first we create a new wsl.conf file and disable auto-generation of resolv.conf:

```
echo -e "[network]\ngenerateResolvConf = false" > /etc/wsl.conf
```

Now exit WSL, then terminate it with

```
wsl -t fedora
```

Then enter it again with

```
wsl -d fedora
```

Now we can persist our custom DNS configuration. First, unlink /etc/resolv.conf. This covers cases in which /etc/resolv.conf is linked to systemd-resolved or resolvconf generated files. Then create a new /etc/resolv.conf with the nameserver(s) of your choice:

```
unlink /etc/resolv.conf
echo nameserver 1.1.1.1 > /etc/resolv.conf
```

Why not test network settings now with a system upgrade:

```
dnf upgrade
```

If repositories are synced, you have success!

Missing the `mount` command?

If, at first entry to your new Fedora instance, you are greeted with An error occurred mounting one of your file systems, then that is a sign you are missing the util-linux package, which includes mount as well as other essential commands. Depending on the rootfs you installed, you may not receive any error, in which case you can proceed to the next section.

But if you do need mount, install util-linux (or, if you are going minimalist, just install util-linux-core).

```
dnf install -y util-linux
```

Then exit, and terminate your fedora instance (this, in effect, causes a restart):

```
wsl -t fedora
```

Launch Fedora as an unprivileged user

Just sayin': root should never be your default user. Let's create a normal user.

We are going to need passwd for this, and we might as well get the cracklib dicts for password checking, too. Install both, once you have launched Fedora:

```
dnf install -y passwd cracklib-dicts
```

Now, create a new user; while we are at it, let's add them to the wheel group so the user can sudo (don't forget to replace "myusername" with something that makes sense for you):

```
useradd -G wheel myusername
```

Then create a password for that user: passwd myusername Now, exit WSL or launch a new Powershell window, then re-launch WSL with the new username:

```
wsl -d fedora -u myusername
```

Success?

```
$ whoami
myusername
```

Does sudo work?

```
sudo cat /etc/shadow
```

If you see the list of users, including, toward the bottom, the one you recently added, then all is well!

Set the default user

It would be nice to enter your Fedora instance as the user you just created, rather than as root.

To enable this, assuming you have Windows build 18980 or later: simply add a user section to /etc/wsl.conf.

Something like this will work well if you do not already have that file, or a [user] section in it:

```
printf "\n[user]\ndefault = myusername\n" | sudo tee -a /etc/wsl.conf
```

Then exit, and terminate your fedora instance, so that it starts fresh next time.

```
wsl -t fedora
```

Launch WSL again, without specifying a user, and you should be that user, not root.

If that succeeded, then proceed to the next section.

But if on a version of Windows before build 18980, then you will instead need to edit the registry to set a default user.

In Fedora, the user you created likely has a user id of 1000, but you can check with id -u. Remember that number.

Back in Powershell you can set the default user by editing the Windows registry, changing "fedora" and "1000" to what you named your WSL distro and the user id, respectively:

```
Get-ItemProperty Registry::HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Lxss\*\
DistributionName | Where-Object -Property DistributionName -eq fedora  | Set-ItemProperty -
Name DefaultUid -Value 1000
```

Fine tuning

If you do container work, especially in userspace, you will likely want to reinstall shadow-utils, in order to fix sticky bits that weren't set properly in the rootfs:

```
sudo dnf reinstall -y shadow-utils
```

If you like to ping servers to see if they are up, then these two steps may be necessary:

```
sudo dnf install -y procps-ng iputils
sudo sysctl -w net.ipv4.ping_group_range="0 2000"
```

The second one allows group IDs all the through 2000 to be able to ping. You can check group IDs with getent group or see your primary group ID with id -g and make sure it is included in the range

above.

To make the above permanent, however, it is necessary to create or alter the $HOME.wslconfig file in Windows, not Linux. Placing the following in that file will allow ping to work even after restarts:

```
[wsl2]
kernelCommandLine = sysctl.net.ipv4.ping_group_range=\"0 2000\"
```

You may also find other commands you are used to are missing from the sparse rootfs we installed. You may want to install iproute, findutils, ncurses, and others, like so:

```
sudo dnf -y install iproute findutils ncurses
```

While the included vi is a useful text editor, you will likely want one that is either more robust or has a user interface that suits you. Good options include vim, micro, or nano. Pick one and install it with something like:

```
sudo dnf install -y micro
```

From here, you can proceed to install packages, edit configurations, and customize your new distro to your heart's content!

A few good man pages

You want the docs? You can't handle the docs!

But if you can handle them, and can handle the extra storage space they will occupy, then you likely want man pages. Thanks to Martin Hinze for his good suggestions on how best to add man page functionality. This is an enhancement, rather than baked in, because we obtained a slimmed-down rootfs earlier.

First, ensure the nodocs option is not set in /etc/dnf/dnf.conf. You may edit out the tsflags=nodocs line yourself, or use the following:

```
grep -v nodocs /etc/dnf/dnf.conf | sudo tee /etc/dnf/dnf.conf
```

Then install man and man-pages:

```
sudo dnf install -y man man-pages
```

This will ensure you get man pages on every future dnf install; however, to add them in retroactively, you will want to dnf reinstall any package for which you want man pages. For instance, man dnf will yield nothing now. But try it again after sudo dnf reinstall -y dnf and you should have good results.

To reinstall all installed packages, try the following:

```
for pkg in $(dnf repoquery --installed --qf "%{name}"); do sudo dnf reinstall -qy $pkg; done
```

Don't repeat yourself

Once you have a pristine base system the way you want it, why not export a tarball that you can import later. I mean, I am honored if you want to read this article again and follow it step by step. But if you want life to be a little easier, you might try the following.

First, clean up downloaded packages, etc. within Fedora:

```
sudo dnf clean all
```

Then, exit WSL and export the whole installation to a tarball:

```
wsl --export fedora $HOME\Downloads\fedora-wsl.tar
```

You may want a different folder than Downloads; specify the location you desire.

Depending on what packages you installed, it may be as small as a quarter GB. You could gzip it if you want the storage size to be even smaller. Next time you want to start fresh, you can do something like this:

```
mkdir $HOME\wsl\freshfedora
wsl --import freshfedora $HOME\wsl\freshfedora $HOME\Downloads\fedora-wsl.tar
```

Keep upgrading

Even if you have a prerelease, there should be no need to reinstall. Just keep upgrading; the process is pretty seamless:

```
sudo dnf upgrade
```

For instance, if you decided to use Fedora 37 (you adventurer, you!), upgrade as often as you like with the above command, and you will eventually (by November of 2022, presumably) be at release.

Minio cluster s3fs

```
#compose.yaml
version: '3.8'

services:
  minio1:
    image: quay.io/minio/minio:latest
    hostname: minio1
    volumes:
      - ./minio1-data:/data
    environment:
      MINIO_ROOT_USER: minioadmin
      MINIO_ROOT_PASSWORD: minioadmin
    command: server http://minio{1...3}/data --console-address ":9001"

  minio2:
    image: quay.io/minio/minio:latest
    hostname: minio2
    volumes:
      - ./minio2-data:/data
    environment:
      MINIO_ROOT_USER: minioadmin
      MINIO_ROOT_PASSWORD: minioadmin
    command: server http://minio{1...3}/data --console-address ":9001"

  minio3:
    image: quay.io/minio/minio:latest
    hostname: minio3
    volumes:
      - ./minio3-data:/data
    environment:
      MINIO_ROOT_USER: minioadmin
      MINIO_ROOT_PASSWORD: minioadmin
    command: server http://minio{1...3}/data --console-address ":9001"

  nginx:
```

```
    image: nginx:alpine
    ports:
      - "9000:9000"  # MinIO API
      - "9001:9001"  # MinIO Console (UI)
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - minio1
      - minio2
      - minio3
```

```
#nginx.conf
events {
    worker_connections 1024;
}

http {
    upstream minio_servers {
        server minio1:9000;
        server minio2:9000;
        server minio3:9000;
    }

    upstream console_servers {
        server minio1:9001;
        server minio2:9001;
        server minio3:9001;
    }

    server {
        listen 9000;
        location / {
            proxy_pass http://minio_servers;
            proxy_set_header Host $host;
        }
    }

    server {
        listen 9001;
        location / {
```

```
            proxy_pass http://console_servers;
            proxy_set_header Host $host;
        }
    }
}
```

использовать s3fs

```
sudo yum install s3fs
echo "minioadmin:minioadmin" > ~/.passwd-minio
chmod 600 ~/.passwd-minio
mkdir /mnt/minio
sudo s3fs test /mnt/minio -o passwd_file=/home/elama/.passwd-
minio,use_path_request_style,url=http://localhost:9000,umask=0007,allow_other
```

Docker registry mirror

```
mkdir -p ./containers-registry-proxy/cache
mkdir -p ./containers-registry-proxy/certs

podman run --rm --detach --name containers-registry-proxy \
  --publish 0.0.0.0:3128:3128 \
  --env ENABLE_MANIFEST_CACHE=true \
  --env REGISTRIES="quay.io gcr.io k8s.gcr.io ghcr.io mcr.microsoft.com registry.gitlab.com" \
  --volume "$(pwd)/containers-registry-proxy/cache":/docker_mirror_cache \
  --volume "$(pwd)/containers-registry-proxy/certs":/ca \
  rpardini/docker-registry-proxy:0.6.4
```

- docker.io
- quay.io
- gcr.io
- k8s.gcr.io
- ghcr.io
- mcr.microsoft.com
- registry.gitlab.com