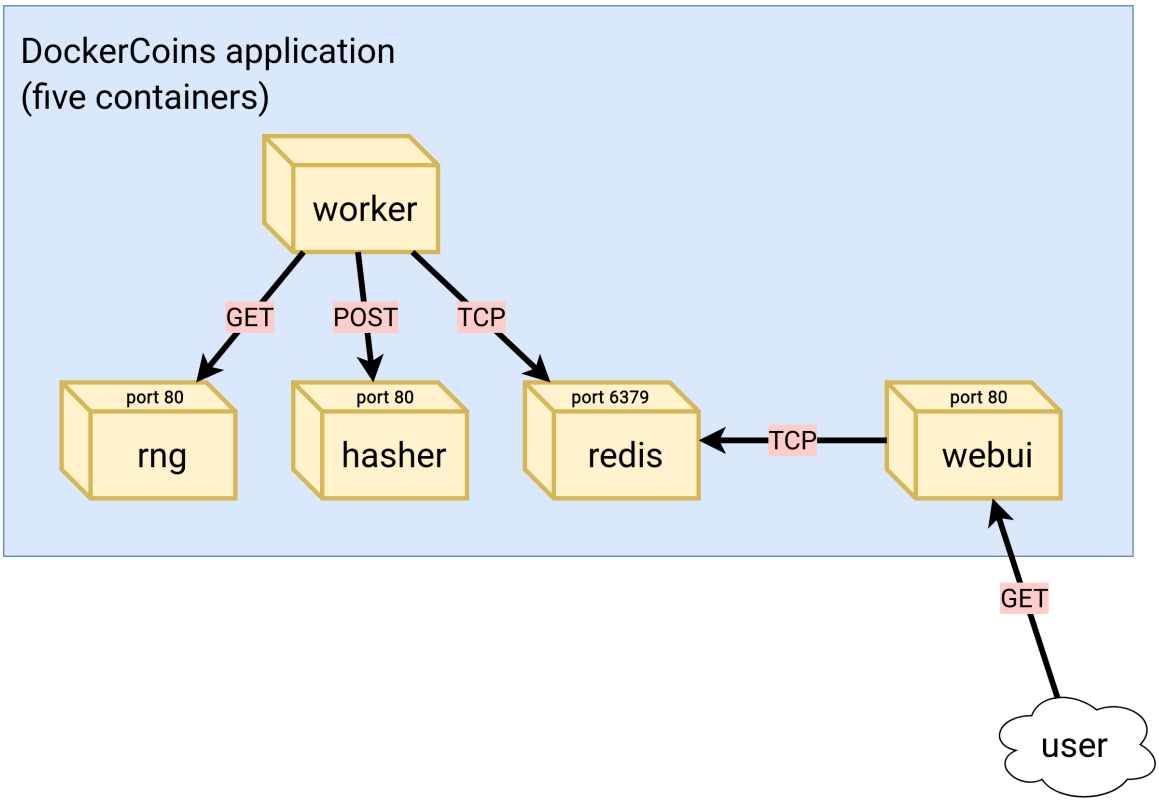
APPENDIX III. How to deploy a distributed containerized application (DockerCoins) through a VPN in a manually-managed distributed cluster with multiple nodes[1]

In our experiments, we will use a sample application called **DockerCoins**[2], which emulates part of the process to generate cryptocurrencies. Basically, DockerCoins generates a few random bytes, hashes these bytes, and increments a counter (to keep track of speed).

DockerCoins is made of 5 services, each of them deployed within a container:

- `rng` = web service generating random bytes
- `hasher` = web service computing hash of POSTed data
- `worker` = background process calling `rng` and `hasher`
- `webui` = web interface to watch progress
- `redis` = data store (holds a counter updated by `worker`)



The diagram shows how DockerCoins works, namely, 1) `worker` invokes web service `rng` to generate random bytes, 2) `worker` invokes web service `hasher` to hash these bytes, 3) `worker` does this in an infinite loop, 4) every second, `worker` updates `redis` to indicate how many loops were done, 5) `webui` queries `redis`, and computes and exposes "hashing speed" in our browser.

We aim to distribute DockerCoins services among two nodes:

- Node 1 will host the `rng`, `worker` and `redis` services.
- Node 2 will host the `hasher` and `webui` services.

By default, a container can only connect to other containers in the same node. Now, there are multiple nodes, hence we must first enable the communication among containers executing in different physical hosts. There are two main approaches to achieve this: *underlay* or *overlay*.

[1] A node refers to a machine — physical or virtual — in our cluster.
[2] https://github.com/jpetazzo/dockercoins

In the ‘underlay’ network approach, containers are directly exposed to the host network. Macvlan network driver is an example of this approach. In the ‘overlay’ network approach, there is an additional level of encapsulation like VXLAN or NVGRE between the container network and the underlay network. We have tested several VPN technologies that provide ‘overlay’ networking. Detailed instructions to set up networking among containers through each of them are provided below:

1. Clone the repository that contains the code
   `git clone https://github.com/jpetazzo/container.training`
2. Go to the `dockercoins` directory in the cloned repository
   `cd ~/container.training/dockercoins`
   - The Compose file in this directory (`docker-compose.yml`) lists all 5 services:
     i. redis is using an official image from the Docker Hub
     ii. `hasher`, `rng`, `worker`, `webui` are each built from a Dockerfile. You can check each one at `<SERVICE-NAME>/Dockerfile`
3. Create Dockerfile to install the client daemon of the VPN (to be placed in 'sidecar' folder)

[ZEROTIER]

```
ARG ALPINE_IMAGE=alpine:3.15
ARG ZT_COMMIT=eac56a2e25bbd27f77505cbd0c21b86abdfbd36b
ARG ZT_VERSION=1.8.4

FROM ${ALPINE_IMAGE} as builder

ARG ZT_COMMIT

RUN apk add --update alpine-sdk linux-headers \
  && git clone --quiet https://github.com/zerotier/ZeroTierOne.git /src \
  && git -C src reset --quiet --hard ${ZT_COMMIT} \
  && cd /src \
  && make -f make-linux.mk

FROM ${ALPINE_IMAGE}

COPY --from=builder /src/zerotier-one /usr/sbin/

RUN apk add iproute2

RUN apk add --no-cache --purge --clean-protected libc6-compat libstdc++ \
  && mkdir -p /var/lib/zerotier-one \
  && ln -s /usr/sbin/zerotier-one /usr/sbin/zerotier-idtool \
  && ln -s /usr/sbin/zerotier-one /usr/sbin/zerotier-cli \
  && rm -rf /var/cache/apk/*

ENTRYPOINT ["zerotier-one"]

EXPOSE 9993/udp
```

[TAILSCALE]

```
FROM golang:1.17-alpine3.15 AS build-env

ARG VERSION_LONG="1"
ENV VERSION_LONG=$VERSION_LONG
ARG VERSION_SHORT="22"
ENV VERSION_SHORT=$VERSION_SHORT
ARG VERSION_GIT_HASH="4e0b00ad830e656b1d76f1c5194520469ab0ff92"
ENV VERSION_GIT_HASH=$VERSION_GIT_HASH
ARG TARGETARCH

WORKDIR /go/src/tailscale

RUN apk add --update git \
```

```
 && git clone --quiet https://github.com/tailscale/tailscale.git . \
 && git reset --quiet --hard ${VERSION_GIT_HASH}

RUN go mod download

RUN GOARCH=$TARGETARCH go install -ldflags="\
      -X tailscale.com/version.Long=$VERSION_LONG \
      -X tailscale.com/version.Short=$VERSION_SHORT \
      -X tailscale.com/version.GitCommit=$VERSION_GIT_HASH" \
      -v ./cmd/tailscale ./cmd/tailscaled

FROM alpine:3.15

COPY --from=build-env /go/bin/* /usr/local/bin/

RUN apk add iptables

ENTRYPOINT ["tailscaled"]
```

[NEOROUTER] (client side)

```
FROM ubuntu:20.04

ENV NRCLIENT_DEB_NAME "nrclient-2.3.1.4360-free-ubuntu-amd64.deb"
ENV NRCLIENT_DEB_URL "http://download.neorouter.com/Downloads/NRFree/Update_2.3.1.
4360/Linux/Ubuntu"

ADD . /
RUN apt-get update && apt-get install -y wget iproute2 && \
    wget -P / $NRCLIENT_DEB_URL/$NRCLIENT_DEB_NAME && \
    dpkg -i /$NRCLIENT_DEB_NAME && \
    apt-get -y clean && \
    rm /$NRCLIENT_DEB_NAME

ENTRYPOINT ["/usr/bin/nrservice"]
```

[NEOROUTER] (server side)

a. Choose a server node with a public IP address which can be accessed from all the client nodes.
b. Download and install server

```
wget -c
http://download.neorouter.com/Downloads/NRFree/Update_2.3.1.4360/Linux/Ub
untu/nrserver-2.3.1.4360-free-ubuntu-amd64.deb
sudo dpkg -i nrserver-2.3.1.4360-free-ubuntu-amd64.deb
```

c. Configure server
   i. set domain

```
nrserver -setdomain <DOMAINNAME> <DOMAINPASSWORD>
```

   ii. If you are running dockercoins in VirtualBox VMs configured in 'NAT' mode, you must restrict the address range of the Neorouter server (10.10.0.0 instead of 10.0.0.0; netmask=255.255.0.0 instead of 255.0.0.0) so that it does not overlap with VirtualBox NAT

```
nrserver -dhcp 10.10.0.0 255.255.0.0
```

   iii. add users

```
nrserver -adduser <USERNAME> <PASSWORD> [admin|user]
```

   iv. show settings / users / computers

```
nrserver -showsettings
nrserver -showusers
nrserver -showcomputers
```

4. We must split `docker_compose.yml` in different files that reflect the distribution of services described above, install the VPN containers as sidecar containers, provide the needed capabilities and devices to the sidecars, and link the original services (in particular, its network) with the sidecar containers

[ZEROTIER] (example `docker_compose.xml` with services `hasher` and `webui`)

```
version: "2"

x-zerotier: &zerotier
    build: sidecar
    image: zerotier-sidecar
    devices:
    - /dev/net/tun:/dev/net/tun
    cap_add:
    - NET_ADMIN
    - SYS_ADMIN

services:
  hasher:
    <<: *zerotier
    ports:
    - "8002:80"

  webui:
    <<: *zerotier
    ports:
    - "8000:80"

  hasher-orig:
    build: hasher
    network_mode: service:hasher
    depends_on:
      - hasher

  webui-orig:
    build: webui
    volumes:
    - "./webui/files/:/files/"
    network_mode: service:webui
    depends_on:
      - webui
```

[TAILSCALE] (example `docker_compose.xml` with services `rng`, `redis`, and `worker`)

```
version: "2"

x-tailscale: &tailscale
    build: sidecar
    image: tailscale-sidecar
    devices:
    - /dev/net/tun:/dev/net/tun
    cap_add:
    - NET_ADMIN
    - SYS_MODULE

services:
  rng:
    <<: *tailscale
    ports:
    - "8001:80"

  redis:
    <<: *tailscale

  worker:
    <<: *tailscale

  rng-orig:
    build: rng
    network_mode: service:rng
```

```
    depends_on:
      - rng

  redis-orig:
    image: redis
    network_mode: service:redis
    depends_on:
      - redis

  worker-orig:
    build: worker
    network_mode: service:worker
    depends_on:
      - worker
```

[NEOROUTER] (client side) (example `docker_compose.xml` with services hasher and webui)
We must also assign a different subnet to the *dockercoins_default* network on each host, because neorouter uses the MAC address (which is generated from the IP of this network) to assign the VPN address.

```
version: "2"

x-neorouter: &neorouter
    build: sidecar
    image: neorouter-sidecar
    devices:
    - /dev/net/tun:/dev/net/tun
    cap_add:
    - NET_ADMIN

services:
  hasher:
    <<: *neorouter
    ports:
    - "8002:80"

  webui:
    <<: *neorouter
    ports:
    - "8000:80"

  hasher-orig:
    build: hasher
    network_mode: service:hasher
    depends_on:
      - hasher

  webui-orig:
    build: webui
    volumes:
    - "./webui/files/:/files/"
    network_mode: service:webui
    depends_on:
      - webui

networks:
  default:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.19.20.0/24
```

5. Now, we can start the different services in the corresponding nodes by running the next command on each of them:

```
docker-compose up
```

6. Join each container to the VPN network

   [ZEROTIER]

   a. Create network at https://my.zerotier.com/ : `<NETWORK-ID>` is returned
   b. Configure network as public at https://my.zerotier.com/ (otherwise you will need to authorize manually the peers)
   c. Join each container to the Zerotier network created before, and check that it joined successfully[3]

```
docker exec <CONTAINER-NAME> zerotier-cli join <NETWORK-ID>
docker exec <CONTAINER-NAME> zerotier-cli listnetworks
```

   [TAILSCALE]

   a. Create Ephemeral Auth Key for each container at https://login.tailscale.com/admin/settings/keys : `[TSKEY-ID-X]` is returned
   b. Join each container to the tailscale network, and check that it joined successfully[4]

```
docker exec <CONTAINER-NAME> tailscale up --authkey=[TSKEY-ID-X]
docker exec <CONTAINER-NAME> tailscale status
```

   [NEOROUTER]

   a. Join each container to the Neorouter server (or domain) created before, and check (in the server) that it joined successfully[4]

```
docker exec <CONTAINER-NAME> nrclientcmd -d <NRserverIP> -u <USER> -p <PASS>
```

7. Does the system work correctly? We must do some more work so that the service name defined in the *Compose* file can be used to reach the corresponding container, as Docker Compose is no longer managing this. For each service that needs to connect to other services **running in different hosts**, we need to set the mappings of the service names it connects to with their corresponding VPN IP addresses by modifying the `/etc/hosts` file in the container. In particular, the container running the `worker` service must know the name-address mapping of services `rng`, `hasher` and `redis`, and the container running the `webui` service must know the mapping of the `redis` service.

   a. We can obtain the IP address of a container by using the following command:

```
docker exec <CONTAINER-NAME> ip addr
```

   b. We can add a name-address mapping to the `/etc/hosts` file of a container by running:

```
docker exec <CONTAINER-NAME> sh -c 'echo "<SERVICE-VPN-IP> <SERVICE-NAME>" >> /etc/hosts'
```

8. We can view the web dashboard exposed by the `webui` container by using a web browser to connect to port 8000 of Node 2. A drawing area should show up, and after a few seconds, a blue graph will appear.

---

[3] If you are running dockercoins in a VM, its network adapter must be configured in 'Bridged' mode. In the 'Advanced' settings, the 'Adapter Type' should be "PCnet-FAST III", and 'Promiscuous Mode' should be set to "Allow All". ZEROTIER MIGHT NOT WORK IF VMs ARE NOT ASSIGNED THEIR OWN IPs, AS HAPPENS IN THE eduroam NETWORK

[4] If you are running dockercoins in a VM, Tailscale and Neorouter seem to work also when the network adapter is configured in 'NAT' mode. THUS, THEY SHOULD ALSO WORK IF VMs ARE NOT ASSIGNED THEIR OWN IPs, AS HAPPENS IN THE eduroam NETWORK.