Chapter 1. Using the EchoMix Docker test network

Table of Contents

EchoMix provides a ready-to-deploy Docker image [https://github.com/katzenpost/katzenpost/tree/main/docker] for developers who need a non-production test environment for developing and testing client applications. By running this image on a single computer, you avoid the need to build and manage a complex multi-node mix net. The image can also be run using Podman [https://podman.io/]

The test mix network includes the following components:

- Three directory authority (PKI [https://katzenpost.network/docs/specs/pki/]) nodes
- Six mix [https://katzenpost.network/docs/specs/mixnet/] nodes, including one node serving also as both gateway and service provider
- A ping utility

Requirements

Before running the EchoMix docker image, make sure that the following software is installed.

- A Debian GNU Linux [https://debian.org] or Ubuntu [https://ubuntu.com] system
- Git [https://git-scm.com/]
- Go [https://go.dev/]
- GNU Make [https://www.gnu.org/software/make/]
- Docker [https://www.docker.com], Docker Compose [https://docs.docker.com/compose/], and (optionally) Podman [https://podman.io]

Note

If both Docker and Podman are present on your system, EchoMix uses Podman. Podman is a drop-in daemonless equivalent to Docker that does not require superuser privileges to run.

On Debian, these software requirements can be installed with the following commands (running as superuser). **Apt** will pull in the needed dependencies.

```
# apt update
            # apt install git golang make docker docker-compose podman
```

Preparing to run the container image

Complete the following procedure to obtain, build, and deploy the EchoMix test network.

1. Install the EchoMix code repository, hosted at https://github.com/katzenpost. The main EchoMix repository contains code for the server components as well as the docker image. Clone the repository with the following command (your directory location may vary):

   ```
   ~$ git clone https://github.com/katzenpost/katzenpost.git
   ```

2. Navigate to the new `katzenpost` subdirectory and ensure that the code is up to date.

   ```
   ~$ cd katzenpost
                              ~/katzenpost$ git checkout main
                              ~/katzenpost$ git pull
   ```

3. (Optional) Create a development branch and check it out.

   ```
   ~/katzenpost$ git checkout -b devel
   ```

4. (Optional) If you are using Podman, complete the following steps:

   1. Point the DOCKER_HOST environment variable at the Podman process.

      ```
      $ export DOCKER_HOST=unix:///var/run/user/$(id -u)/podman/podman.sock
      ```

   2. Set up and start the Podman server (as superuser).

      ```
      $ podman system service -t 0 $DOCKER_HOST &
                                      $ systemctl --user enable --now podman.socket
      ```

Operating the test mixnet

Navigate to `katzenpost/docker`. The `Makefile` contains target operations to create, manage, and test the self-contained EchoMix container network. To invoke a target, run a command with the using the following pattern:

```
~/katzenpost/docker$ make target
```

Running **make** with no target specified returns a list of available targets.:

Table 1.1. Makefile targets

[none]	Display this list of targets.
run	Run the test network in the background.
start	Run the test network in the foreground until **Ctrl-C**.
stop	Stop the test network.
wait	Wait for the test network to have consensus.
watch	Display live log entries until **Ctrl-C**.

status	Show test network consensus status.
show-latest-vote	Show latest consensus vote.
run-ping	Send a ping over the test network.
clean-bin	Stop all components and delete binaries.
clean-local	Stop all components, delete binaries, and delete data..
clean-local-dryrun	Show what clean-local would delete.
clean	The above, plus cleans includes go_deps image.

Starting and monitoring the mixnet

Either of two command targets, **run** and **start**, can be used to start the mix network. The first They differ only in that **start** quickly detaches and runs the network in the background, while **run** runs the network in the foreground.

Note

When running **run** or **start** , be aware of the following considerations:

- If you intend to use Docker, you need to run **make** as superuser. If you are using **sudo** to elevate your privileges, you need to edit `katzenpost/docker/Makefile` to prepend **sudo** to each command contained in it.
- If you have Podman installed on your system and you nonetheless want to run Docker, you can override the default behavior by adding the argument **docker=docker** to the command as in the following:

  ```
  ~/katzenpost/docker$ make run docker=docker
  ```

The first time that you use the **start** target, the docker image will be downloaded, built, and installed. This takes several minutes.

```
~/katzenpost/docker$ make start
                     ...
                     ~/katzenpost/docker$ make watch
                     ...
                     <output>
                     ...
```

Once installation is complete, there is a further delay as the mix servers vote and reach a consensus. You can use the **wait** target to wait for the mixnet to get consensus and be ready to use. This can also take several minutes:

```
~/katzenpost/docker$ make wait
                     ...
                     <output>
                     ...
```

You can confirm that installation and configuration are complete by issuing the **status** command from the same or another terminal. When the network is ready for use, **status** begins returning consensus information similar to the following:

```
~/katzenpost/docker$ make status
```

```
...
00:15:15.003 NOTI state: Consensus made for epoch 1851128 with
...
```

Testing the mixnet

At this point, you should have a locally running mix network. You can test whether it is working correctly by using **ping**, which launches a packet into the network and watches for a successful reply. Run the following command:

```
~/katzenpost/docker$ make run-ping
```

If the network is functioning properly, the resulting output contains lines similar to the following:

```
19:29:53.541 INFO gateway1_client: sending loop decoy
                !19:29:54.108 INFO gateway1_client: sending loop decoy
                19:29:54.632 INFO gateway1_client: sending loop decoy
                19:29:55.160 INFO gateway1_client: sending loop decoy
                !19:29:56.071 INFO gateway1_client: sending loop decoy
                !19:29:59.173 INFO gateway1_client: sending loop decoy
                !Success rate is 100.000000 percent 10/10)
```

lf **ping** fails to receive a reply, it eventually times out with an error message. If this happens, try the command again.

Note

If you attempt use **ping** too quickly after starting the mixnet, and consensus has not been reached, the utility may crash with an error message or hang indefinitely. If this happens, issue (if necessary) a **Ctrl-C** key sequence to abort, check the consensus status with the **status** command, and then retry **ping**.

Shutting down the mixnet

The mix network continues to run in the terminal where you started it until you issue a **Ctrl-C** key sequence, or until you issue the following command in another terminal:

```
~/katzenpost/docker$ make stop
```

When you stop the network, the binaries and data are left in place. This allows for a quick restart.

Uninstalling and cleaning up

Several command targets can be used to uninstall the Docker image and restore your system to a clean state. The following examples demonstrate the commands and their output.

- **clean-bin**

  To stop the network and delete the compiled binaries, run the following command:

  ```
  ~/katzenpost/docker$ make clean-bin
  ```

  ```
  [ -e voting_mixnet ] && cd voting_mixnet && DOCKER_HOST=u
  Stopping voting_mixnet_auth3_1        ... done
  ```

```
Stopping voting_mixnet_servicenode1_1 ... done
Stopping voting_mixnet_metrics_1      ... done
Stopping voting_mixnet_mix3_1         ... done
Stopping voting_mixnet_auth2_1        ... done
Stopping voting_mixnet_mix2_1         ... done
Stopping voting_mixnet_gateway1_1     ... done
Stopping voting_mixnet_auth1_1        ... done
Stopping voting_mixnet_mix1_1         ... done
Removing voting_mixnet_auth3_1        ... done
Removing voting_mixnet_servicenode1_1 ... done
Removing voting_mixnet_metrics_1      ... done
Removing voting_mixnet_mix3_1         ... done
Removing voting_mixnet_auth2_1        ... done
Removing voting_mixnet_mix2_1         ... done
Removing voting_mixnet_gateway1_1     ... done
Removing voting_mixnet_auth1_1        ... done
Removing voting_mixnet_mix1_1         ... done
removed 'running.stamp'
rm -vf ./voting_mixnet/*.alpine
removed './voting_mixnet/echo_server.alpine'
removed './voting_mixnet/fetch.alpine'
removed './voting_mixnet/memspool.alpine'
removed './voting_mixnet/panda_server.alpine'
removed './voting_mixnet/pigeonhole.alpine'
removed './voting_mixnet/ping.alpine'
removed './voting_mixnet/reunion_katzenpost_server.alpine
removed './voting_mixnet/server.alpine'
removed './voting_mixnet/voting.alpine'
```

This command leaves in place the cryptographic keys, the state data, and the logs.

- **clean-local**

  To delete both compiled binaries and data, run the following command:

  ```
  ~/katzenpost/docker$ make clean-local
  ```

  ```
  [ -e voting_mixnet ] && cd voting_mixnet && DOCKER_HOST=u
  Removing voting_mixnet_mix2_1         ... done
  Removing voting_mixnet_auth1_1        ... done
  Removing voting_mixnet_auth2_1        ... done
  Removing voting_mixnet_gateway1_1     ... done
  Removing voting_mixnet_mix1_1         ... done
  Removing voting_mixnet_auth3_1        ... done
  Removing voting_mixnet_mix3_1         ... done
  Removing voting_mixnet_servicenode1_1 ... done
  Removing voting_mixnet_metrics_1      ... done
  removed 'running.stamp'
  rm -vf ./voting_mixnet/*.alpine
  removed './voting_mixnet/echo_server.alpine'
  removed './voting_mixnet/fetch.alpine'
  removed './voting_mixnet/memspool.alpine'
  removed './voting_mixnet/panda_server.alpine'
  removed './voting_mixnet/pigeonhole.alpine'
  ```

```
removed './voting_mixnet/reunion_katzenpost_server.alpine
removed './voting_mixnet/server.alpine'
removed './voting_mixnet/voting.alpine'
git clean -f -x voting_mixnet
Removing voting_mixnet/
git status .
On branch main
Your branch is up to date with 'origin/main'.
```

- **clean**

  To stop the the network and delete the binaries, the data, and the go_deps image, run the following command as superuser:

  ```
  ~/katzenpost/docker$ sudo make clean
  ```

- **clean-local-dryrun**

  ```
  ~/katzenpost/docker$ make clean-local-dryrun
                       git clean -n -x voting_mixnet
                       Would remove voting_mixnet/
  ```

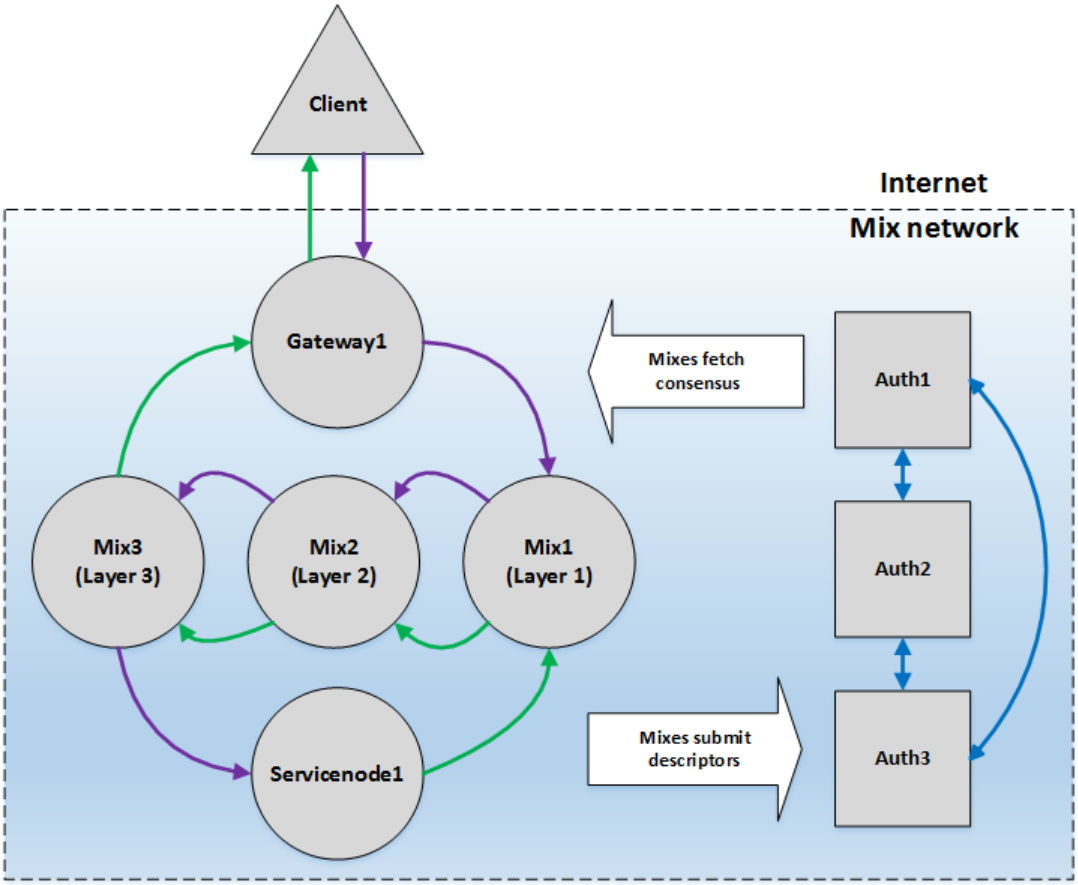
```
~/katzenpost/docker$ make clean
```

For a preview of the components that **clean-local** would remove, without actually deleting anything, running **clean-local-dryrun** generates output as follows:

Network components and topology

There needs to be an interpretation of this diagram. including the ways that the testnet differs from production network.

Figure 1.1. Test network topology



Discuss how to view these components, where their configuration files are, etc.

Table 1.2. Network hosts

Host type	Identifier	IP	Port	Panda
Directory authority	auth1	127.0.0.1	30001	
Directory authority	auth2	127.0.0.1	30002	
Directory authority	auth3	127.0.0.1	30003	
Gateway node	gateway1	127.0.0.1	30004	
Service node	servicenode1	127.0.0.1	30006	✓
Mix node	mix1	127.0.0.1	30008	
Mix node	mix2	127.0.0.1	30010	
Mix node	mix3	127.0.0.1	30012	