

# Marlin Network

- [Overview](#)
- [Monitoring](#)
- [Beacon](#)
- [Relay](#)
- [Instructions](#)
- [Gateway](#)

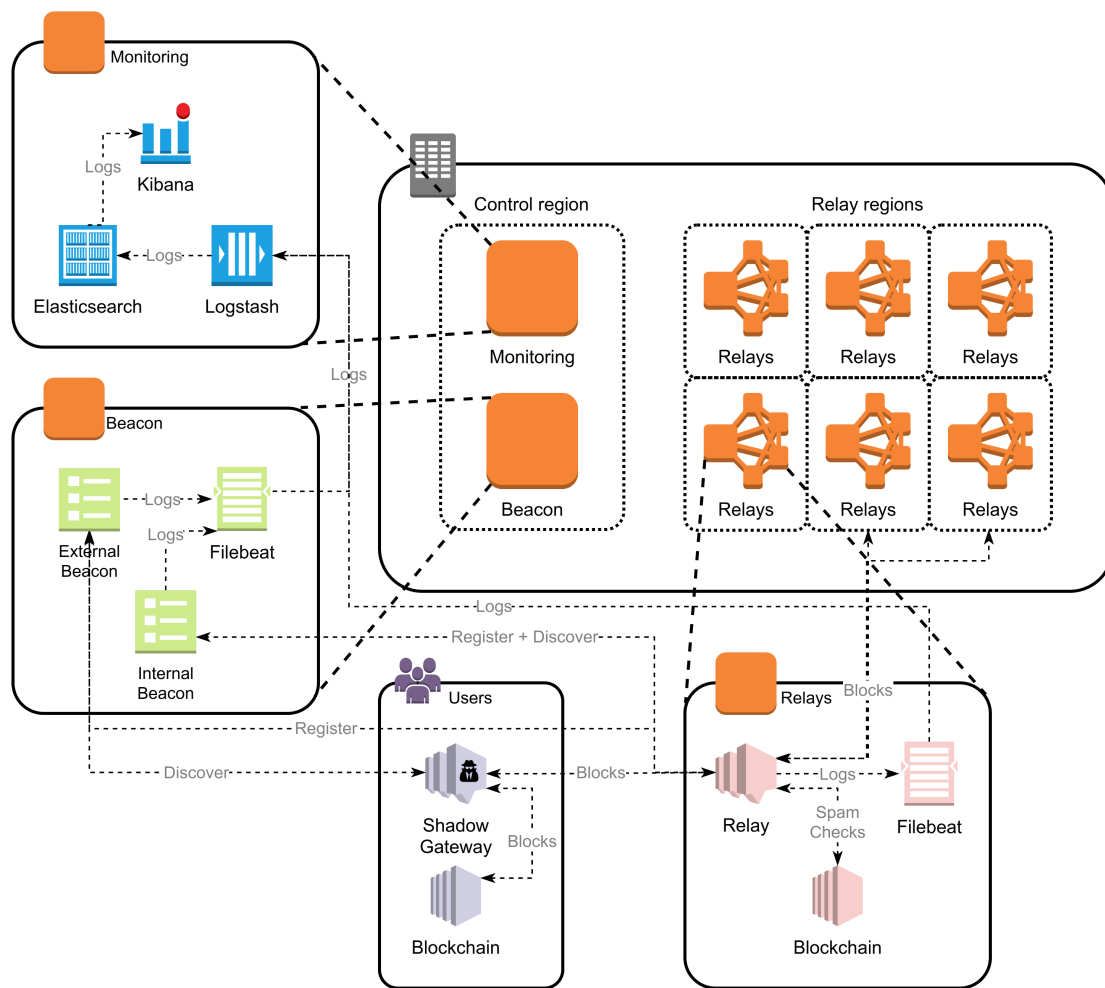
# Overview

## Architecture

The current architecture has 4 types of nodes

- Beacon
- Monitoring
- Relay
- User (Producers + Consumers)

A cluster operator is responsible for running managing the beacon, monitoring and relay nodes. Users of the network need to simply run and manage a user node.



## Beacon

The beacon node is similar to seed nodes or bootstrap nodes in other networks. It is the initial touchpoint using which a user discovers the other nodes in the network.

See [Beacon](#) for more details.

## Monitoring

The monitoring node is used to monitor the network by aggregating logs and metrics from the other nodes.

See [Monitoring](#) for more details.

## Relay

The relay nodes form the core of the network and are responsible for propagating information from one user to the others.

See [Relay](#) for more details.

## User

The user nodes produce and receive messages propagating through the network.

See [Gateway](#) for more details.

## Hardware Requirements

Node type	CPU	Mem	Disk space
Beacon	1	1 GB	< 1 GB
Monitoring	8	50 GB	50 GB
Relay	1	1 GB	< 1 GB

## Devops scripts

Reference scripts for provisioning and deployment of a cluster on GCP using Pulumi+Ansible are provided [here](#). While some tweaks might be needed to customize it based on your requirements and infrastructure, it should provide a good base for any customizations.

# Monitoring

The monitoring architecture collects logs and metrics from other nodes and provides dashboards for visualization. It is fairly modular since the core network produces log files which can then be consumed by any logging and analytics pipelines that support tailing log files.

Currently, we provide deployment scripts for Filebeat + the ELK stack as the default pipeline.

## Security considerations

Elasticsearch and Logstash ports must be protected from public access, otherwise malicious actors can introduce garbage logs into the pipelines.

The Kibana port also needs to be protected from public access, however, a permissioned access model might be a better fit here to selectively allow people to access logs and create visualizations. An example would be to use Nginx + TLS + HTTP Basic Auth.

# Beacon

The beacon node acts as a touchpoint for nodes to discover relay nodes in the cluster.

## Using [marlinctl](#)

### Usage

```
$ sudo ./marlinctl beacon create --help
NAME:
  marlinctl beacon create - create a new beacon

USAGE:
  marlinctl beacon create [command options] [arguments...]

OPTIONS:
  --program value                                --program <NAME> (default: "beacon")
  --discovery-addr value                        --discovery-addr <IP: PORT> (default:
127.0.0.1: 8002)
  --heartbeat-addr value                       --heartbeat-addr <IP: PORT> (default:
127.0.0.1: 8003)
  --bootstrap-addr value, --beacon-addr value  --bootstrap-addr <IP: PORT>
  --version value                              --version <NUMBER> (default: "latest")
  --help, -h                                  show help (default: false)
```

### Run

```
$ sudo ./marlinctl beacon create
```

## Manually

### Build

The beacon executable is built as part of [OpenWeaver](#). See the README for build instructions.

## Usage

```
$ ./beacon/beacon --help
```

```
USAGE: beacon [ OPTIONS]
```

```
OPTIONS:
```

```
-d, --discovery-addr <discovery_addr>
-h, --heartbeat-addr <heartbeat_addr>
-b, --beacon-addr <beacon_addr>
-h, --help <help>
-v, --version <version>
```

## Run

```
$ ./beacon/beacon
```

## Discussion

The beacon executable supports two optional command-line arguments to set the discovery and heartbeat listener addresses. The discovery address enables nodes to discover the nodes in its registry and the heartbeat address enables nodes to add themselves to the registry.

By default, `discovery-addr` is set to `127.0.0.1:8002` and `heartbeat-addr` is set to `127.0.0.1:` to make it secure by default. This means that the beacon node can **only be reached from localhost** by default. To make it available for other nodes not on localhost, it needs to listen on external interfaces as well - usually using `0.0.0.0` to bind to all interfaces:

```
# Using marlinctl
$ sudo ./marlinctl beacon create --discovery-addr "0.0.0.0:8002" --heartbeat-addr
"0.0.0.0:8003"
# Manually
$ ./beacon/beacon --discovery-addr "0.0.0.0:8002" --heartbeat-addr "0.0.0.0:8003"
```

In addition, the beacon node supports a `--beacon-addr` parameter to register the cluster with the wider Marlin Relay network.

# Relay

The relay node is responsible for message propagation. It mainly comprises of 2 programs:

- Relay
- Abci

## Using [marlinctl2](#)

### Usage

```
$ sudo ./marlinctl relay create --help
NAME:
  marlinctl relay create - create a new relay

USAGE:
  marlinctl relay create [command options] [arguments...]

OPTIONS:
  --chain value           --chain "<CHAIN>"
  --discovery-addr value  --discovery-addr "<IP1: PORT1>, <IP2: PORT2>, ..." (default:
"127. 0. 0. 1: 8002")
  --heartbeat-addr value  --heartbeat-addr "<IP1: PORT1>, <IP2: PORT2>, ..." (default:
"127. 0. 0. 1: 8003")
  --datadir value         --datadir "/path/to/datadir" (default: "~/.ethereum/")
  --discovery-port value  --discovery-port <PORT> (default: 0)
  --pubsub-port value     --pubsub-port <PORT> (default: 0)
  --address value         --address "0x..."
  --name value            --name "<NAME>"
  --version value         --version <NUMBER> (default: "latest")
  --abci-version value    --abci-version <NUMBER> (default: "latest")
  --help, -h              show help (default: false)
```

### Run

```
$ sudo ./marlinctl relay create --chain "eth"
```

The abci is automatically created.

## Manually

### Build

The relay executable is built as part of [OpenWeaver](#). See the README for build instructions.

The abci is available [here](#). Use only the "Building from source" instructions (not Installation).

### Usage

```
$ ./relay/eth_relay --help

USAGE: eth_relay [ OPTIONS] discovery_addrs heartbeat_addrs datadir

OPTIONS:
  -p, --pubsub-port <pubsub_port>
  -d, --discovery-port <discovery_port>
  -a, --address <address>
  -h, --help <help>
  -v, --version <version>

ARGS:
  discovery_addrs
  heartbeat_addrs
  datadir
```

### Run

```
# Blockchain
$ ./geth --datadir /path/to/datadir/ --syncmode=light

# Relay
$ ./relay/eth_relay "127.0.0.1:8002" "127.0.0.1:8003" "/path/to/datadir/"
```

# Discussion

The relay executable takes 3 arguments:

- discovery-addr - Comma separated string of beacon addresses to initiate discovery
- heartbeat-addr - Comma separated string of beacon addresses to register with
- datadir - Path to the data directory of the blockchain node

The relay executable also supports three optional command-line arguments:

- --pubsub-port - To set the local port used for the message relay (default: 5000)
- --discovery-port - To set the local port used to initiate discovery (default: 5002)
- --address - To set the reward address for the incentivized testnet

## Why is the blockchain node needed?

The blockchain node is needed to detect spam. For supporting block propagation, the blockchain node can be a light client but to support transaction propagation, it needs to be a full node.

This is a temporary requirement and will eventually be replaced (or at the very least, decoupled) with better spam prevention measures (e.g. an onchain light client).

# Instructions

This page details the procedure to join the Marlin network.

## For clusters

### Step 1: Preliminaries

- Refer to the architecture diagram [here](#) and understand the interactions
- Preferably, set up `marlinctl2` (available [here](#)) to make the setup process smoother
- If you prefer a manual setup or `marlinctl2` doesn't cover your use case, clone and build [OpenWeaver](#)

### Step 2: Set up beacon

- Run a [beacon](#) node
- Ensure the beacon is reachable from outside
- Pass "34.82.79.68:8003" as the beacon address while starting the beacon to register it with the wider testnet

```
# Using marlinctl
$ sudo marlinctl beacon keystore create
$ sudo marlinctl beacon create --discovery-addr "0.0.0.0:8002" --heartbeat-addr
"0.0.0.0:8003" --bootstrap-addr "34.82.79.68:8003"

# Manually
$ ./beacon/beacon --discovery-addr "0.0.0.0:8002" --heartbeat-addr "0.0.0.0:8003" --beacon-
addr "34.82.79.68:8003"
```

### Step 3: Set up relays

- Run a [relay](#) node
- Set the `discovery-addrs` parameter to point to the `discovery-addr` of your beacon set up above
- Set the `heartbeat-addrs` parameter to point to the `heartbeat-addr` of your beacon set up above

```
# Using marlinctl
$ sudo marlinctl relay eth create --discovery-addrs "beaconip: 8002" --heartbeat-addrs
"beaconip: 8003"

# Manually
$ ./relay/eth_relay "beaconip: 8002" "beaconip: 8003" "/path/to/datadir/"
```

## Step 4: Set up firewalls

- Expose for public access
  - Discovery port of beacon
  - Discovery port of relay
  - Pubsub port of relay
- Expose only for relay access
  - Heartbeat port of beacon

## Private network setup

Make sure the setup above works fine before attempting the private network setup. Also ensure that your nodes can communicate with each other through private IPs.

## Replace step 2:

- Run `_two_` beacons, one public facing and one private facing

```
# Using marlinctl
$ sudo marlinctl beacon keystore create
# Public beacon with bootstrap address
$ sudo marlinctl beacon create --discovery-addr "0.0.0.0: 8002" --heartbeat-addr
"0.0.0.0: 8003" --bootstrap-addr "34.82.79.68: 8003"
# Private beacon without bootstrap address
$ sudo marlinctl beacon create --discovery-addr "0.0.0.0: 9002" --heartbeat-addr "0.0.0.0: 9003"
```

```
# Manually
$ ./beacon/beacon --discovery-addr "0.0.0.0: 8002" --heartbeat-addr "0.0.0.0: 8003" --beacon-addr "34.82.79.68: 8003"
$ ./beacon/beacon --discovery-addr "0.0.0.0: 9002" --heartbeat-addr "0.0.0.0: 9003"
```

## Replace step 3:

- Set the `discovery-addrs` parameter to point to the `discovery-addr` of your `_private_` beacon set up above
- Set the `heartbeat-addrs` parameter to point to the `heartbeat-addr` of both your `_public_` and `_private_` beacons set up above

```
# Using marlinctl
$ sudo marlinctl relay eth create --discovery-addrs "privateip: 9002" --heartbeat-addrs "privateip: 9003,publicip: 8003"

# Manually
$ ./relay/eth_relay "privateip: 9002" "privateip: 9003,publicip: 8003" ""
```

At the end of this, you should see your private beacon getting heartbeats from the `_private_` IPs of the relays and the public beacon getting heartbeats from the `_public_` IPs of the relays.

## Tuning

The linux UDP stack needs to be tuned for optimal performance. Here's a good reference for the parameters - <https://gist.github.com/voluntas/bc54c60aaa7ad6856e6f6a928b79ab6c>.

Note: Try to understand what you're changing and why before you change it, since it depends on your system characteristics if any of the commands there would improve or worsen performance (and stability, be wary of memory exhaustion).

The important bits seem to be:

- Run `watch netstat -us` - Check if send or receive errors are increasing with time, indicates buffer overflows
- If yes, increase buffer size (replace the numbers below as you please)

```
# Per-socket read buffers
net.core.rmem_default = 8192000
net.core.rmem_max = 8192000

# Per-socket write buffers
net.core.wmem_default = 8192000
net.core.wmem_max = 8192000

# Option memory
net.core.optmem_max = 8192000

# Global tuning (multiplied by 4KB)
net.ipv4.udp_mem = 64000    64000    64000
```

- Restart the relays and check again after some time

# Gateway

Gateways enable people to connect their blockchain nodes to the Marlin network to send and receive messages.

## Polkadot

Documentation for the polkadot gateway is available [here](#).

## NEAR Protocol

### Usage (using [marlinctl2](#))

```
$ sudo marlinctl gateway near create --help
NAME:
  marlinctl gateway near create - create a new gateway

USAGE:
  marlinctl gateway near create [command options] [arguments...]

OPTIONS:
  --bootstrap-addr value  --bootstrap-addr "<IP:PORT>" (default: "127.0.0.1:8002")
  --version value         --version <NUMBER> (default: "latest")
  --help, -h              show help (default: false)
```

## Run

```
$ sudo marlinctl gateway near create --bootstrap-addr "54.219.22.51:8002"
```

## Connect

You need to connect your blockchain node to the gateway to send and receive data from it.

### Step 1: Get the gateway identity

```
$ sudo supervisorctl tail near_gateway
```

```
...  
[ 2020-12-06 16:20:45.648] [info] [OnRampNear.hpp:63] Node identity:  
Ar5vnFLiYeX8jHTNCKBJuTND1ez85fHZN5Q4ikanFtU  
...
```

The above command prints logs which contain the identity key of the gateway. This step only needs to be performed once since the key is stored locally for future runs.

For advanced users, the key is stored as a file (`.marlin/keys/near_gateway`). You can supply your own pre-generated key files to make the above step deterministic (useful for automation scripts).

## Step 2: Add as a peer

The gateway now needs to be added as a peer. There are a variety of ways to do this, here's an example using the commandline while starting the near node:

```
# Use the key obtained above  
$ ./target/release/neard run --boot-nodes  
"Ar5vnFLiYeX8jHTNCKBJuTND1ez85fHZN5Q4ikanFtU@<gateway_ip>: 21400"
```

That's it!

## IRISnet

The IRISnet gateway comprises of two programs:

- Gateway
- Bridge

## Usage (using [marlinctl2](#))

```
$ sudo marlinctl gateway iris create --help  
NAME:  
    marlinctl gateway iris create - create a new gateway  
USAGE:  
    marlinctl gateway iris create [command options] [arguments...]  
OPTIONS:  
    --bootstrapaddr value    --bootstrapaddr "<IP1:PORT1>" (default: "127.0.0.1:8002")  
    --listenportpeer value   --listenportpeer "PORT" (default: "59001")
```

```
--peerip value      --peerip "IP" (default: "127.0.0.1")
--peerport value    --peerport "PORT" (default: "26656")
--rpcport value     --rpcport "PORT" (default: "26657")
--version value     --version <NUMBER> (default: "latest")
--help, -h          show help (default: false)
```

## Run

```
sudo marlinctl gateway iris create --bootstrapaddr "52.8.52.100:8002"
```

The bridge is automatically setup for you.

## Connect

You need to connect your blockchain node to the gateway to send and receive data from it.

### Step 1: Get the gateway identity

```
$ ls ~/.marlin/ctl/configs/iris_keyfile*
/home/exampleuser/.marlin/ctl/configs/iris_keyfile-0.0.1.json
$ grep IdString /home/exampleuser/.marlin/ctl/configs/iris_keyfile-0.0.1.json
  "IdString": "f4d35da5490d9e5962b0b3041ccc2980b0dec5dd",
```

The above command prints the node ID of the gateway. This keyfile is initially pulled from remote to help get the system up and running as soon as possible for end user. However, you can also generate your own keyfile and save it at the same place (`/home/exampleuser/.marlin/ctl/configs/iris_keyfile-0.0.1.json`) for gateway to use for future runs.

New keys can be generated using the following:

```
$ cd ~/.marlin/ctl/bin/
$ ./iris_gateway-0.0.1 keyfile -g -f /home/exampleuser/.marlin/ctl/configs/iris_keyfile-0.0.1.json -c irisnet
```

This would respond with:

```
[INFO]: 2020-12-05 11:29:10 - Generating KeyPair for irisnet-0.16.3-mainnet
[INFO]: 2020-12-05 11:29:10 - ID for node after generating KeyPair:
6f5c8faeb8d14bb0c28e9dda22cc2d580e7af929
[INFO]: 2020-12-05 11:29:10 - Successfully written keyfile
```

```
/home/exampleuser/.marlin/ctl/configs/iris_keyfile-0.0.1.json
```

Hence, `6f5c8faeb8d14bb0c28e9dda22cc2d580e7af929` is the new nodeID. Verify the same using:

```
$ grep IdString /home/exampleuser/.marlin/ctl/configs/iris_keyfile-0.0.1.json
```

## Step 2: Restart IRISnet node with gateway as persistent peer

Kill your iris instance running with `iris start` command. Then, open the config file for iris using:

```
$ vim ~/.iris/config/config.toml
```

Change persistent peer configuration from

```
# Comma separated list of nodes to keep persistent connections to
persistent_peers = ""
```

to:

```
# Comma separated list of nodes to keep persistent connections to
persistent_peers = "f4d35da5490d9e5962b0b3041ccc2980b0dec5dd@127.0.0.1:59001"
```

Make note: Here, correct node ID which your system has should be entered along with IP and Port in the format `nodeID@ip:port`. Failure to do so could lead to gateway not being able to connect.

Save the configuration file and exit using `ESC -> :wq -> RETURN` and begin iris using `iris start`

## Step 3: Run the gateway

Run the gateway using run command given above.

**Note:** Run the gateway post running the iris node only, since the gateway needs to do RPC calls to the iris node to find the chain and verify compatibility before running.

That's it!

## Fantom

The gateway for Fantom is undergoing testing. Please feel free to register at <https://form.typeform.com/to/mx9BbUM0>

. You will receive a notification once the gateway is ready. Your delegators will be made eligible for FlowMint immediately for a grace period till the gateway is not ready and installed.

## Matic

The gateway for Matic is undergoing testing. Please feel free to register at <https://form.typeform.com/to/mx9BbUM0>. You will receive a notification once the gateway is ready. Your delegators will be made eligible for FlowMint immediately for a grace period till the gateway is not ready and installed.

## Discussion

### What exactly is a gateway?

Gateways are basically a stripped down version of the blockchain node (called a shadow node) which implement only the p2p networking parts. Most notably, there's no discovery, sync, storage or consensus, so the gateway is extremely lean.

On one side, since the gateway implements the p2p networking stack, you can add the gateway as a peer to your blockchain node and it will automatically communicate with it (no invasive code changes). On the other side, it automatically discovers and maintains connections with Marlin relay nodes and is able to send and receive data from it.

### Why is a bridge sometimes needed?

The two sides of the gateway are occasionally incompatible with each other. This might be due to a variety of reasons including different event loops, incompatible languages/libraries, etc. Hence, the two sides are split into two parts which communicate between each other through the network instead.

### Does it affect the safety/stability of my node?

Not more than connecting to any other peer. The shadow node approach is designed to affect the real blockchain node as little as possible:

- there are no codebase changes
- the blockchain node still verifies all messages and runs its consensus engine as always
- communication is strictly through its own p2p networking stack

- the blockchain node doesn't interact with the Marlin network directly (in fact, it doesn't even know that Marlin exists)
- the gateway can be run in a different VM separate from the node to further isolate it

Hence, the "attack surface" is pretty much the same as another peer that your node connects to.

## My node doesn't connect to other (random) peers though.

The blockchain node still needs to get the data from some other node in your control. E.g. some chains secure their validator nodes by running them behind sentry nodes which insulate the validator against rogue peers. You can simply choose to run the gateway beside the sentry node instead of the validator.