# EWF Authority Nodes - Requirements and Procedures

Markus Keil, Friedrich Raschwitz

January 2019

**Abstract**

To provide a stable and secure blockchain for all affiliates and customers, it is important that all parties that run authority nodes follow the procedures and guidelines written in this document. Authority nodes are not used for interaction with the chain. If an affiliate wants to access the chain via RPC through a central node it has to be set up separately as non-authority.

# Contents

# 1 Nomenclature

**EWF Network Operations (NetOps)** Group of people on EWF side with the responsibility to keep the blockchain secure and operational at all times.

**EWF Governance Operations (GovOps)** Group of people on EWF side that decide on governance questions.

**NodeControl** A system component that carries out operational tasks on a validator node on behalf of NetOps

**Bootnode** Parity node that runs in fullnode configuration and is part of the bootnode section of the chainspec file. New clients that join the chain will contact a bootnode to discover other nodes on the network that are known to the bootnode.

**Validator Node** Parity node that seals transactions into new blocks based on the AURA consensus algorithm.

**Genesis Node** A special validator node. The genesis nodes are the first ones on the network and operated by EWF. These nodes will bootstrap the blockchain with the genesis block.

**Fullnode** A simple node on the network that don't seal new blocks. These nodes are not in the scope of these guidelines

**Supportnode** A host that runs auxiliary services like IPFS or Incubed Server

# 2 System Design

The system of the validator nodes and their supporting components are designed to provide security and stability. The basic layout is shown in Figure 1. Also along the active validator set is a set of standby validator nodes called Shadow Validators, that can swap in for any failed or otherwise compromised validator.

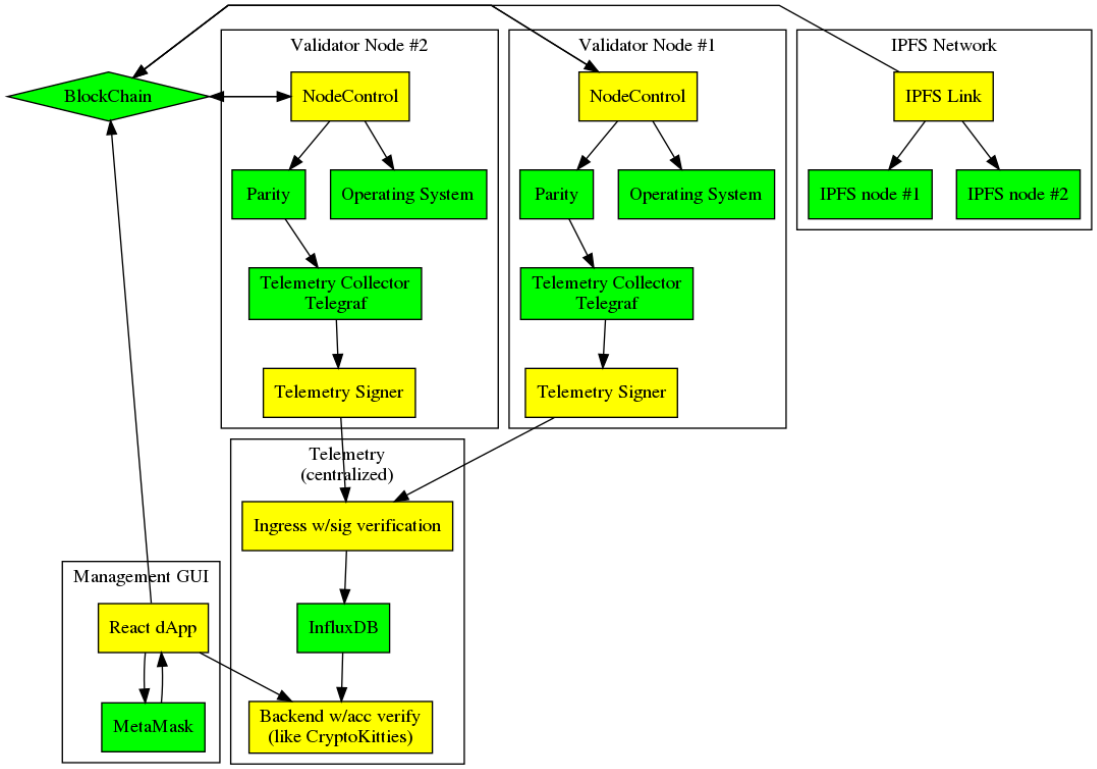


Figure 1: Validator System Design

## 2.1 Shadow Validators [Not part of MVP]

As running hundreds of validators at the same time increases network traffic and also time to finality a lot, the concept of shadow validators can be used to have a small set of 23 active validators and a number of dormant shadow validators.

During onboarding an affiliate can choose either to run a single validator node or provide resources for two validator nodes from which only one gets enlisted into the active validator set. The second node is also completely onboarded but would run with the sealer disabled. It would be just a normal peer acting as a full node (as any other validator would). If another validator node would go offline or start misbehaving, the network starts to counteract this behavior by activating a shadow validator, while in the same time remove the misbehaving validator from the active set. This will happen automatically via the validator smart contract. To ensure the network can heal it self this way, the validator address of the shadow node is recorded to the validator contracts shadow node list, from which it then can draw new nodes. The contract will emit a wake up event that will be picked up by NodeControl on the dormant validator node and restarts the blockchain client with the sealer enabled. NodeControl reports back successful activation to the validator contract which then will add the now active shadow node to the active list of validators. Along with the wakeup event a disable event is recorded on chain for the misbehaving one.

The main purpose of this is to counteract single-node attacks, where one node is brought down due to some kind of DoS attack. Once the attacked node is back up again and re-validated by NetOps, it can replace the activated shadow node which then would go back to be dormant.

Additional idea is to use this mechanism to swap random nodes in and out of the shadow set on a regular basis. Governance has to make sure that over the period of a year each validator had an equal share of

"active time" to evenly distribute transaction fees and block rewards.

## 2.2 System Components

These system components are tailor made or where developed for the validator network.

### 2.2.1 NodeControl

A small deamon that will carry out updates on behalf of NetOps. See 3.

### 2.2.2 Telemetry Signer

System telemetry is collected using Telegraf. But to ship the telemetry, Telegraf won't send it directly to the telemetry backend but instead to a local deamon called "Telemetry Signer". This deamon will make sure that telemetry is only send to a verified telemetry collection endpoint (Telemetry Ingress) using TLS certificate fingerprints. Each telemetry package send to the Ingress will be signed by the telemetry signer.

### 2.2.3 Telemetry Ingress

A deamon that will wait for connections from telemetry signers. It will verify the signature of each incoming telemetry package before inserting it into the InfluxDB telemetry datastore. If the signature is not valid it will report this to a log file.

### 2.2.4 IPFS Link [Not part of MVP]

IPFS Link is a service that allows human readable urls to be resolved to IPFS hashes. It is used in this system to provide the link to the NetOps Management dApp. This way the management dApp can be hosted decentralized to increas reliability.

## 2.3 Other Components

These components are standardize open-source components.

### 2.3.1 IPFS [Not part of MVP]

IPFS is used to distribute the management ui. A central support node would provide an IPFS node that is connected to the public IPFS network. The management UI would be accessed either through a local IPFS client or trough a public IPFS gateway. This way the UI stays accessible, even when the support node fails.

### 2.3.2 Telemetry Collector

The system uses Telegraf to collect the telemetry from on the nodes. This collected data is then send via the InfluxDB wire protocol to the also locally running telemetry signer.

### 2.3.3 Grafana [Only MVP]

To visualize the telemetry the telemetry node will run a Grafana instance. In a later milestone this then could be replaced or integrated into the governance dApp.

### 2.3.4 Blockchain Client

The blockchain client is the main component of the system as it provides the connection to the blockchain and also carries out signing and validation duties. The probably used software will be Parity-Ethereum from Parity Tech running the AURA Proof-of-Authority engine.

# 3 NodeControl

NodeControl is a small management application that runs on each validator node. It is in charge of carrying out simple update tasks. It will listen to the local blockchain client via the HTTP RPC interface. If the local blockchain is not available it will fallback to the Incubed network to receive and verify update events [1]. The ethereum address of the validator account is used as a node identifier. NodeControl will only act on events directly directed to its assigned address. This allows granular deployment of updates.

NodeControl will also track the local blockchain client to determine the current number of validators [2]. This is needed to determine finality of an update command. The trigger process is shown in Fig. 2.

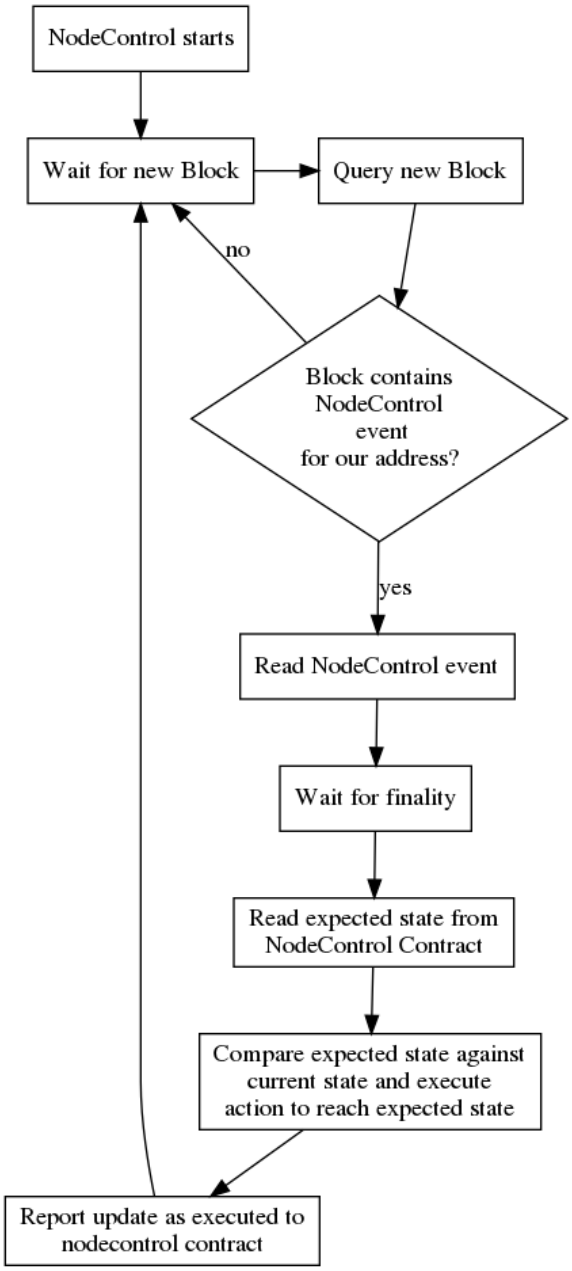


Figure 2: NodeControl Trigger Process

[1] Incubed and this feature are not part of the MVP
[2] MVP will have a fixed, configurable number of block to determine finality

## 3.1 Interface to Contract

This is an abstract definition of the Smartcontract interface between the NodeControl deamon and the accompanying contract on-chain.

- event UpdateAvailable(address targetValidator, unit eventId)
  The contract has to emit this event once an update should be triggered on a node. If one would like to update multiple validators at once the event has to be send for each node individual.

- function RetrieveUpdate(address targetValidator, unit eventId)
  returns (uint command, byte[] payloadHash, string payload)
  NodeControl will call this function after finality has been reached to retrieve the actual command and payload.
  Command is an uint based on this map:
  0 => updateParity Will update the parity docker container.
  1 => updateChainspec Will update the chainspec file.
  2 => enableSigner Will restart Parity with signing enabled (payload and hash empty)
  3 => disableSigner Will restart Parity with signing disabled (payload and hash empty)

  The command code will also determine the meaning of the two remaining fields payloadHash and payload.

  - updateParity
    payloadHash => The SHA256 Id of the docker image returned by docker image inspect
    payload => The dockerimage from dockerhub including tag (eg. parity/parity:v2.2.5)
  - updateChainspec
    payloadHash => The SHA256 hash of the downloaded file
    payload => HTTPS url to download the new chainspec

- function ConfirmUpdate(unit eventId, boolean success)
  After NodeControl carried out the update it will report back with a transaction calling this function. It will send a boolean if the update was successful or not. The transaction will be send from the validator account over HTTP RPC.

# 4 Genesis Network

The Genesis Network is a special set of validators and full nodes that will be used during launch. Hosts of the genesis network will be geographically distributed across different AWS regions. These regions will be linked via VPC peering, so traffic inside the genesis network is not public facing.

The genesis network will be run by the EnergyWeb Foundation.

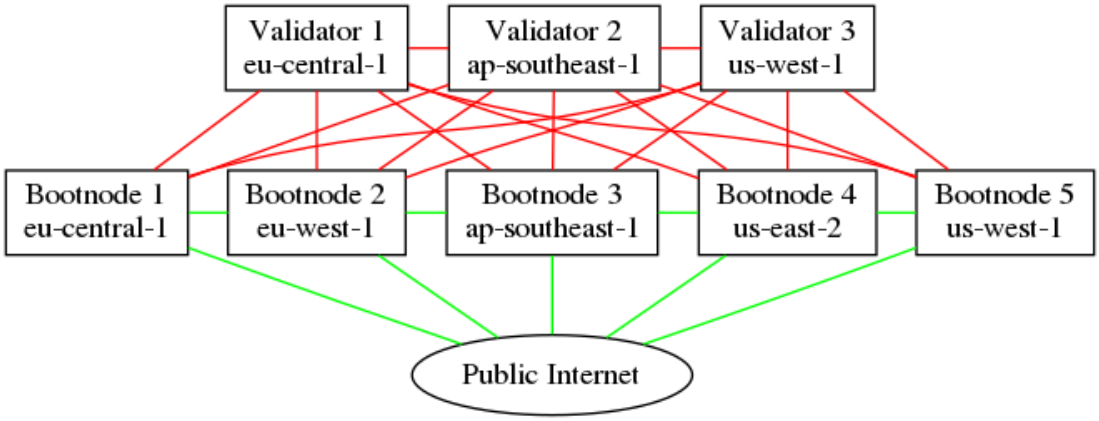


Figure 3: Genesis Network Layout

## 4.1 Validators

The genesis validators will be starting the chain and also their validator account addresses are hard coded as constructor arguments to the validator contract in the chain specification. The hosts running those validator nodes adhere to the same security standards as normal validators with the addition that they don't have any connection to the public internet blockchain wise.
The following outgoing traffic is permitted though an AWS NAT gateway:

- DNS (63/UDP)
- HTTPS (443/TCP) - mainly for sys updates

Blockchain traffic is only allowed inside the AWS VPC to the Bootnodes. The validators will have a special configuration file for parity to facilitate that along with the following security group settings

- Allow P2P traffic (30303/tcp+udp) inbound only from the bootnodes via RFC1918 inside the VPC
- Allow P2P traffic (30303/tcp+udp) outbound only to the bootnodes via RFC1918 inside the VPC
- Allow SSH traffic only inbound via a jump host inside the VPC over RFC1918

A preferred EC2 Instance type would be i3.xlarge (4 cores - Xeon E5 2686v4, 30GB RAM, 950GB NVMe SSD). These nodes also don't have dedicated public IP's

## 4.2 Bootnodes

The genesis bootnodes are regular fullnodes (state pruned, no tracing, no fatdb, no warp). They accept connections from the public internet and the validator VPC's on their blockchain P2P port.

Enode addresses of those bootnodes will be part of the general public chainspec.

# 5 Threat model

This section describes potential attack vectors to the system and how they are either mitigated by the system design, automatic intervention or human intervention.

## 5.1 Telemetry

As telemetry needs to flow from the validators to a single ingress point on the management system it is not protected by decentralization. This telemetry helps to detect abnormalities in the operation of the validators caused by either system malfunction or deliberate attacks.

**Sending tampered data** If an attacker manages to disturb node operation, he might try to disguise this by sending normal looking telemetry to the telemetry ingress on behalf of the attacked node. NetOps would detect the faulty node with increased time delay as telemetry data would contradict actual node behavior.
**Prevention:** Each telemetry package will be signed with the validators private key and verified by telemetry ingress. This way an attacker is not able to send tampered telemetry on behalf of a node.

**Denial-of-Service against the telemetry ingress** An attacker might try to bring down the telemetry ingress completely to "blind" the EWF NetOps team about the status of the validator network.
**Mitigation:** Telemetry senders on the validators will keep track of connection failures to the telemetry ingress. If the connection is unstable or down for a particular amount of time the node will send periodically (<10 minutes) basic health metrics via a second channel (eg. E-Mail). EWF NetOps has access to this channel and can at least verify basic health of the system.

**Phishing for telemetry** An attacker might try to receive telemetry from a node by re-routing telemetry traffic to an attacker system that mimics the ingress (DNS spoofing, MITM). The attacker can gain knowledge about for example the systems load patterns or how a system might respond to an attack.
**Prevention:** Connection between telemetry sender and telemetry ingress is HTTPS. The SHA256 fingerprint of the ingress certificate is verified against the known good fingerprints. The fingerprint gets deployed along with the telemetry sender component and can only be updated through an update request from the blockchain.

**DNS-Spoof/MitM against GUI** An attacker might manage to DNS Spoof/MitM the connection between the browser of a member from the NetOps team and the telemetry backend. This way the attacker could present a faked system state of the validator network to NetOps.
**Prevention:** Each JSON response from the telemetry backend needs to carry a signature for the payload that can be verified by the management dApp

**Unauthorized access to telemetry data** Telemetry data should only be available to the EWF NetOps Team. An attacker might try to directly call the telemetry backend to receive the data.
**Prevention:** The telemetry backend will present a challenge to the GUI which the EWF NetOps team member has to sign with his private key using metamask. This challenge+signature is then send along with all following requests and verified against a smart contract.

## 5.2 Node Control

The Node control component is used to carry out updates to the validator nodes. These updates need to be legitimized by the EWF NetOps and GovOps teams. The component has the potential to disable a node due to a faulty or malicious update .

**Send fake update event** An attacker might try to fool the NodeControl into carrying out an update script that is not approved.
**Prevention:** NodeControl will talk to its local blockchain client most of the time via IPC. If it has to talk to outside nodes because the local client is down, it'll use incubed to verify responses.

**Compromised Payload** An attacker might MitM traffic to any server NodeControl would download a payload from (mainly used for the chain spec file update process).
**Containment:** Each update will have the SHA256 hash of its agreed payload on chain. The payload downloaded by NodeControl is verified against this hash. If the hashes don't match NodeControl will not carry out the command and report a faulty update to the NodeControl Contract.

## 5.3 Management GUI/dApp

The Management GUI is the central hub for the NetOps team to verify healthiness and current operational status of the validator network. Therefore it has a high requirement on reliability.

**Server delivering UI to NetOps is not available** An attacker might be able to DDoS the Server/CDN or otherwise disturb connection to the assets needed to run the management dApp.
**Prevention:** The dApp itself is hosted on IPFS. Therefore taking down a single UI serving node won't disturb operation. For improved reliability NetOps members should run their own local IPFS nodes and use the IPFS companion extension, This allows the local IPFS node to be used to deliver the GUI to the browser.

**Deliver malicious dApp to NetOps team** An attacker might be able to deliver a malicious dApp to the NetOps team. With this it might be possible to deceive NetOps about the current status of the validator network or have them sign bogus update commands.
**Prevention:** Provide a local bootstrap copy of the index.html page that loads the dApp from IPFS, preferably with a local IPFS node. Have hardcoded checksums of those files in the index.html and verify payload during load as the attacker could intercept HTTPS traffic to the IPFS gateways or the local node and deliver a malicious asset to the browser.

## 5.4 Other Attack Vectors

This section consolidates attack vectors that are either not specific to a single component or targeted against miscellaneous components.

**Attack time servers** The Aura PoA consensus algorithm highly depends on accurate system clocks. An attacker could try to manipulate the system clock by intercepting requests made by the operating system via NTP (Network Time Protocol). If the attacker shifts the system time by +block time or -block time the validator will start signing during the wrong slot. Other validators then won't accept that block because it is out-of-turn and vice-versa the manipulated node would mark all incoming blocks as invalid as from its point of view all other validators appear to be syncing out-of-turn.
**Mitigation:** Some validator nodes should use high-precision hardware clocks such as GPS-based ones or DCF-77 clocks instead of NTP for their time source. This way not the whole network is affected. Normal operation could be restored by removing all non-precision-clock nodes from the network. This could mean reduced chain performance.

# 6 Incentivation

The EWF chain knows four levels on nodes:

- The double-validator with an active sealer and a shadow node
- The single validator with only the active sealer
- A support node running a non-sealing full node along with an IPFS and InCubed server. Support nodes don't need to fulfill the higher security standards that validator nodes need to provide.
- plain non-validator nodes, which are just normal blockchain client nodes that don't need to adhere to any increased security standards. They can be run by anybody and therefore are not part of these guidelines.

Best case scenario would be that every affiliate runs the double validator configuration along with a separate support node. This would take 3 separate hosts the affiliate has to maintain. Each affiliate should at least run a support node as it the easiest configuration to run and to secure.

The target for the active validator set should be at 25 validator nodes with the same amount of shadow nodes.

# 7 Validators run by Affiliates

This section will go through the requirements needed to run a standard validator by an EWF affiliate.

## 7.1 Hardware/VM selection for Validators

Affiliate can choose to run the node either On-Premise on their own hardware or at one of the supported cloud-providers.

### 7.1.1 On-Premise

A on-premise node should have these specs or higher. These resources should be reserved for the validator node and not shared with other workloads.

1. Modern Multi-core x64 CPU (at least 4 threads, preferably Xeon-class)
2. 8GB RAM (preferably ECC)
3. Local SSD storage, 250GB free capacity for blockchain, redundant in RAID-1
4. 1 GBit NIC

### 7.1.2 Amazon AWS

The following EC2 instance sizes are appropriate to run validators:

- m5.xlarge
- m5.2xlarge
- m5a.xlarge
- m5a.2xlarge
- c5.xlarge
- c5.2xlarge

The default EBS storage assigned (normally 8GB) is not large enough to run the node. Make sure to run the node with following EBS storage settings:

- General Purpose SSD (gp2)
- at least 300GB size

### 7.1.3 Microsoft Azure

The following Azure VirtualMachine sizes are suitable to run a validator:

- D4s_v3
- DS3_v2
- B4ms

Use **Premium SSD** as attached storage with a size of at least **300GB**.

## 7.2 Reliability and Diversity

EWF NetOps will guarantee a certain reliability and diversity by enforcing heterogeneity of geographical location, Operative System and cloud service. The rationale is that a bug of a specific OS, a failure of a cloud service provider or the backbone internet service provider should not harm the consensus of the Validators.
For this reason, NetOps establishes a set of rules:

- There are at least two Validators running different OS in the system.
- The second most common OS is run by at least 20% of Validators.
- There are at least 15% Validators operating from America, 15% from Europe and 15% from Asia.
- At least two cloud services are used. The second most popular is used by at least 20% of the Validators.

## 7.3 Supported Operating Systems

The following Linux-based Operating Systems are supported for running a authority node:

- Ubuntu Server 18.04 LTS
- Debian 9.8
- CentOS 7

Affiliates can apply for a certain operating system, but in order to keep a heterogeneous infrastructure EWF NetOps can instruct the Affiliate to use a specific operating system from the above list. The operating system must be installed according to the settings described in this document to qualify the host for becoming part of the authority network.

## 7.4 Security Requirements

Running an authority nodes requires raised awareness of host and node security as authorities are a main attack surface to disturb operation of the block chain. The following security rules apply:

- No services are permitted to run on the same host that are not part of the authority node package
- All incoming connections on all ports except SSH (22/tcp) and the P2P (30303/tcp+udp) port have to be firewalled on the host with DROP rules. To guarantee proper network etiquette, incoming ICMP has to be accepted.
- SSH access is only allowed for non-root users
- SSH access is only allowed through RSA keys
- Parity RPC endpoints (HTTP, WebSocket) have to be disabled
- System updates have to applied regularly and in a timely manner
- Regular run of rootkit detectors

Most of these rules will be provided in the following set up guides.

## 7.5 Connectivity Requirements

The following requirements should be met to ensure proper operation:

- Wired connection with 100 MBit/s symmetric link to the internet
- Low latency connection to next internet hop (<5ms)
- No data volume limitations

Even this document requires a 100MBit/s connection, that connection will not be saturated by the node. You can expect 10-30MBit/s [3] when the chain is under load. Traffic will mainly flow on port 30303 (udp/tcp). If chosen to run on a cloud provider, EWF NetOps needs to be consulted to select an appropriate hosting region to ensure node distribution across multiple regions. If chosen to run on-premise, EWF NetOps has to be informed about the hosting location to ensure proper global distribution decisions can be made. The hosting location should be chosen to be as close as possible to one of the major internet exchanges:

- Germany/Continental Europe/USA - DE-CIX / AMS-IX
- United Kingdom - LINX

## 7.6 AWS Security

When using EC2 instances one can provide at least one additional layer of security using a virtual firewall. The access of administrative tasks outside of the virtual OS also needs to be taken care of.

### 7.6.1 Access Management Recommendations

1. Do not use the root account for managing EC2 instances, instead assign a new user administrative rights for doing that.
2. Use groups to manage permissions with multiple users and always keep the set of permissions to a minimum
3. Enable multi-factor authentication for administrative accounts
4. Regularly check logs regarding account activity, e.g. using Cloudfront AWS Docs: Access Logs

### 7.6.2 AWS Firewall

AWS provides an alternative firewall solution for their virtual OS. Instead of configuring the firewall in the OS itself (or additionally), one can configure it one layer above. This firewall is defined by a set of "security groups". Each security group contains rules what packets should be permitted to or from the EC2 instance. That means that instead of having a set of 'Allow' or 'Deny' rules security groups can only contain 'Allow' rules while everything else is denied by default.

That also means that by default SSH access needs to be allowed in one security group. More information on that can be found here:
AWS Docs: authorizing-access-to-an-instance

### 7.6.3 EC2 Key Pairs

EC2 key pairs are basically just RSA key pairs generated and distributed using AWS-specific commands. In general one can follow the guide in the SSH section. However, these commands may be more convenient to some users, more information here:
AWS Docs: ec2-key-pairs

[3] TODO: need to verify

## 7.7 Setup Procedure

To setup a new authority node read this whole document and then use this procedure to carry out the set up. The full process is shown in Figure 4

1. Choose hosting provider (on-premise or qualified cloud provider) and favored operating system
2. Consult with EWF NetOps on location and operating system
3. EWF NetOps will provide the location and OS to use, along with the official installation script for the chosen operating system
4. Install operating system according to this document
5. Deploy blockchain node in none-validator mode using the script given by NetOps
6. Contact EWF NetOps to confirm installation and incoming telemetry
7. Create Validator Keys and prepare your node configuration to switch to authority mode using the script.
8. Send the validator account information to the EWF Governance team
9. Re-start your node in authority-mode

# 8 Operating system installation

The following section provide a comprehensive guide for installation of one the supported operating systems. All further deployment procedures are based on the installation results.

## 8.1 Ubuntu Server 18.04 LTS

### 8.1.1 On-Premise

Procedure based on version 18.04.2.

- Download the ISO from https://www.ubuntu.com/download/server.
- Boot the ISO
- Select **English** as language
- Choose a convenient keyboard layout
- Choose **Install Ubuntu**
- Let the network auto-configure -or- configure manually if needed. The system needs an internet connection.
- Select no proxy and keep the mirror address.
- Select **Use an entire disk** and confirm
- Choose user name and host name in next screen. Choose a strong password.
- Select **Install OpenSSH Server** but don't import keys
- Don't select any snaps and continue
- Finish installation and let it boot to the prompt
- Login as the created user and run a full system update using `sudo apt update && sudo apt dist-upgrade -y`

### 8.1.2 AWS

The Ubuntu AMI Id's for all regions in AWS can be found in Table 1

Also Ubuntu AMI's are listed at https://cloud-images.ubuntu.com/locator/ec2/. Search for **ebs 18.04 amd64** to get the correct version.

### 8.1.3 Azure

The URN for the image is `Canonical:UbuntuServer:18.04-LTS:latest`

## 8.2 Debian 9.8

### 8.2.1 On-Premise

- Download the NetInst ISO from https://www.debian.org/distrib/netinst
- Boot the ISO
- Select **Install** from the boot screen
- Select **English** as language
- Select Location based on actual location of the host

| Region | AMI ID |
|---|---|
| ap-northeast-1 | ami-0eb48a19a8d81e20b |
| ap-south-1 | ami-007d5db58754fa284 |
| ap-southeast-1 | ami-0dad20bd1b9c8c004 |
| ca-central-1 | ami-01b60a3259250381b |
| eu-central-1 | ami-090f10efc254eaf55 |
| eu-north-1 | ami-5e9c1520 |
| eu-west-1 | ami-08d658f84a6d84a80 |
| sa-east-1 | ami-09f4cd7c0b533b081 |
| us-east-1 | ami-0a313d6098716f372 |
| us-west-1 | ami-06397100adf427136 |
| cn-northwest-1 | ami-09b1225e9a1d84e4c |
| cn-north-1 | ami-09dd6088c3e46151c |
| us-gov-west-1 | ami-66bdd307 |
| us-gov-east-1 | ami-7bd2340a |
| ap-northeast-2 | ami-078e96948945fc2c9 |
| ap-southeast-2 | ami-0b76c3b150c6b1423 |
| eu-west-2 | ami-07dc734dc14746eab |
| us-east-2 | ami-0c55b159cbfafe1f0 |
| us-west-2 | ami-005bdb005fb00e791 |
| ap-northeast-3 | ami-0babd61cf592f1c03 |
| eu-west-3 | ami-03bca18cb3dc173c9 |

Table 1: Ubuntu 18.04 LTS AWS AMI Id's

- Chose a convenient keyboard layout
- Let the network auto-configure -or- configure manually if needed. The system needs an internet connection.
- Name your host. Change it from *debian* to something else
- Choose a strong root password
- create the user account and choose a strong password
- select the proper timezone
- For the partitions use **Guided - use entire disk**
- Select **All files in one partition**
- Finish partitioning and write changes to disk
- Select **No** when ask to scan more disks
- Choose a mirror close to the host
- Opt-out of the package survey
- on the **Software Selection** select only **SSH Server** and **standard system utilities**
- Install the grub bootloader to MBR and use the primary disk for that
- Finish installation and let it boot to the prompt
- Login as root and run a full system update using apt update && apt dist-upgrade -y
- Reboot

| Region | AMI ID |
|---|---|
| eu-north-1 | ami-043a919b6dc7c51cc |
| ap-south-1 | ami-0b6490868957ce747 |
| eu-west-3 | ami-0cb185e7696ffe300 |
| eu-west-2 | ami-0ef10a4062f24d89d |
| eu-west-1 | ami-035c67e6a9ef8f024 |
| ap-northeast-2 | ami-0fa1392d5d545f9e8 |
| ap-northeast-1 | ami-0c4290d7ce45d7bbe |
| sa-east-1 | ami-0bc0ce4ab8b82305c |
| ca-central-1 | ami-0857efbad274a1a89 |
| ap-southeast-1 | ami-04c9740a9ed018dba |
| ap-southeast-2 | ami-0b91189c4f9f5cd9e |
| eu-central-1 | ami-05449f21272b4ee56 |
| us-east-1 | ami-0f9e7e8867f55fd8e |
| us-east-2 | ami-00c5940f2b52c5d98 |
| us-west-1 | ami-0afda78f1d0272d99 |
| us-west-2 | ami-01d07e14f082b3ba1 |

Table 2: Debian 9.8 AWS AMI's

#### 8.2.2 AWS

The AMI Id's for all regions in AWS can be found in Table 2

You can retrieve this list also from https://wiki.debian.org/Cloud/AmazonEC2Image/Stretch

#### 8.2.3 Azure

The URN for the image is credativ:Debian:9:latest

### 8.3 CentOS 7

#### 8.3.1 On-Premise

- Download the minimal ISO from https://www.centos.org/download/
- Boot the ISO
- Confirm the automatic boot option **Test this media & install CentOS 7**
- Choose **English** as language
- On the installation summary choose "Installation destination" and confirm "automatic partinioning"
- Back on the installation summary screen click on "Network & Hostname"
- change the hostname
- enable the network interface and make sure it is configured properly
- Click **Done** to get back to the summary and click **Begin Installation**
- During installation set a root password
- Finish installation and let it boot to the prompt
- Login as root and run a system update with yum update

| Region | AMI ID |
|---|---|
| ap-northeast-1 | ami-25bd2743 |
| ap-northeast-2 | ami-7248e81c |
| ap-south-1 | ami-5d99ce32 |
| ap-southeast-1 | ami-d2fa88ae |
| ap-southeast-2 | ami-b6bb47d4 |
| ca-central-1 | ami-dcad28b8 |
| eu-central-1 | ami-337be65c |
| eu-west-1 | ami-6e28b517 |
| eu-west-2 | ami-ee6a718a |
| eu-west-3 | ami-bfff49c2 |
| sa-east-1 | ami-f9adef95 |
| us-east-1 | ami-4bf3d731 |
| us-east-2 | ami-e1496384 |
| us-west-1 | ami-65e0e305 |
| us-west-2 | ami-a042f4d8 |

Table 3: CentOS 7 AWS AMI's

### 8.3.2 AWS

The AMI Id's for all regions in AWS can be found in Table 3

You can retrieve the list also from
https://wiki.centos.org/Cloud/AWS#head-78d1e3a4e6ba5c5a3847750d88266916ffe69648

### 8.3.3 Azure

The URN for the image is OpenLogic:CentOS:7.5:latest

# 9 Telemetry

Authority nodes have to send automatic telemetry data to NetOps. This helps detecting attacks or other network disturbances early. The following telemetry is collected and send through a secure channel (see Telemetry Signer and Ingress in 2.2) to EWF NetOps:

- CPU usage
- Memory usage
- Disk usage
- Number of connected blockchain peers
- List of visible P2P peers
- Current block
- Network latency to 3 different and major locations (eg. cloudflare, google, amazon)
- Network throughput
- Network error rate
- Number of SSH keys [4]
- Service status for SSH, Docker and the Parity container
- SHA256 hashes of key system components/binaries
- Current chain specification (file or hash)

[4] Used to detect addition of keys

# 10 Normal Operations [Procedure proposal]

After the node joined the authority network, it has to be maintained. It is the Affiliates responsibility to ensure that the node always adheres to the requirements and guidelines outlined in this document. EWF Network Operations can only play a supportive role.

Node telemetry is automatically processed to detect failed or unmaintained nodes. The following Process applies if a node becomes unstable or unavailable:

1. The telemetry system will inform EWF NetOps and the designated contact of the affiliate about an instability
2. The affiliate has to restore service to the node in 24 hours
3. The affiliate should send an explanation about the malfunction to EWF NetOps [5]
4. If after 24 hours the node is still not back to normal operation the affiliate has to contact EWF NetOps with the reason of the malfunction and an estimated schedule when the node is back to normal operation
5. If the affiliate does not contact EWF NetOps in the aforementioned time frame, NetOps will escalate the issue to GovOps. GovOps then has to decide within 48h to remove the authority node from the list of validators.
6. NetOps will provide an incident report if the issue was severe/noteworthy

[5] Necessary to detect potential system-wide issues

# 11 Updates and Hardforks [Procedure proposal]

During normal operation of the blockchain certain updates or changes need to be made on the validator system. These updates can contain operating system updates, updates for any of the system components or updates to the chain specification. All these updates will be coordinated by EWF NetOps. Short abstract of the full flow as shown in Figure 5.

1. NetOps prepares the update and proposes it to GovOps
2. GovOps decides on it
3. NetOps splits active validators into waves
4. NetOps will trigger the update process
5. Nodes will report back success or failure of update
6. When all waves are completed the update is finished

Should a node not be able to finish the update the complete process for the entire system is reverted. The node can then be fixed manually by the affiliate in a given period of time or has to be removed from the validator set as it no longer updateable.

Most updates will be executed automatically through NodeControl.

If the affiliate does not carry out the upgrade procedures during the time frame, NetOps will escalate the matter to GovOps which then will decide upon the removal of the authority node from the validator set.

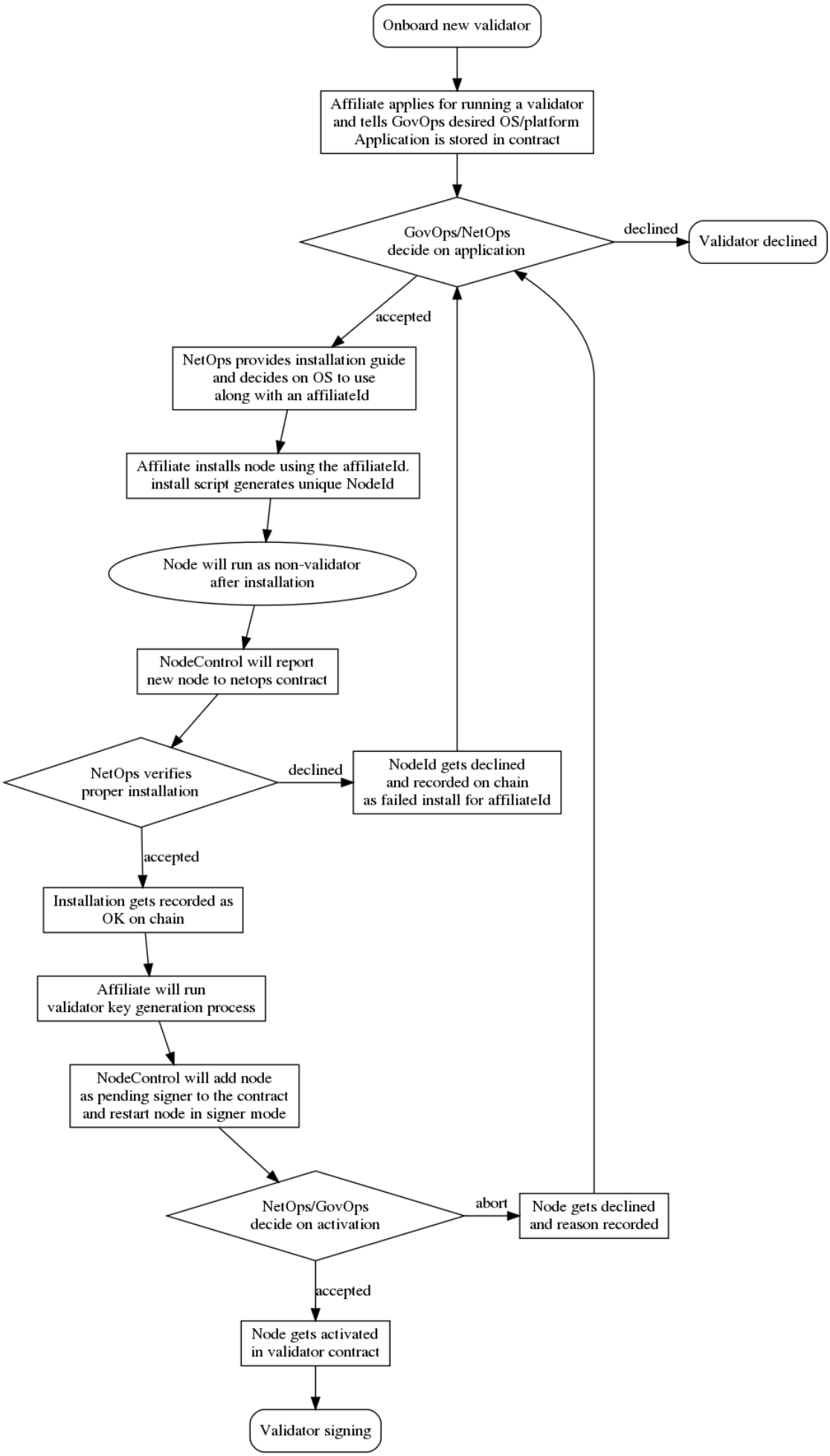


Figure 4: Validator On-Boarding Flow

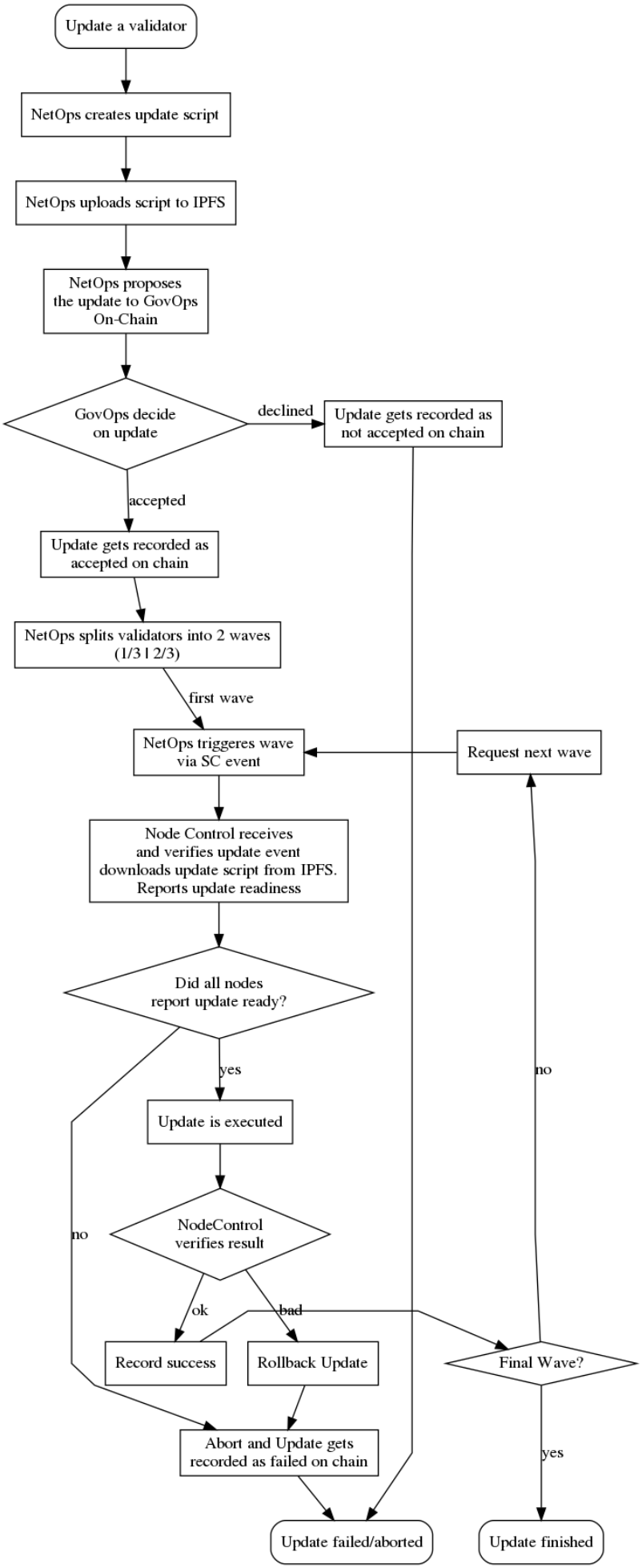


Figure 5: Update Flow