Guide: How to build a Cardano Stake Pool

On Ubuntu/Debian, this guide will illustrate how to install and configure a Cardano stake pool from source code on a two node setup with 1 block producer node and 1 relay node.

∞ Pre-Announcements

> Thank you for your support and kind messages! It really energizes us to keep creating the best crypto guides. Use cointr.ee to find our donation addresses.

> As of Dec 28 2020, this is **guide version 3.0.0** and written for **cardano mainnet** with **release v.1.24.2**

0. Prerequisites

Mandatory skills for stake pool operators

As a stake pool operator for Cardano, you will be competent with the following abilities:

- operational knowledge of how to set up, run and maintain a Cardano node continuously
- a commitment to maintain your node 24/7/365
- system operation skills
- server administration skills (operational and maintenance).

Mandatory experience for stake pool operators

- experience of development and operations (DevOps)

- experience on how to harden and secure a server.
- passed the official Stake Pool School course.

⚠ **Before continuing this guide, you must satisfy the above requirements.**

Minimum Stake Pool Hardware Requirements

- **Two separate servers:** 1 for block producer node, 1 for relay node
- **One air-gapped offline machine (cold environment)**
- **Operating system:** 64-bit Linux (i.e. Ubuntu Server 20.04 LTS)
- **Processor:** An Intel or AMD x86 processor with two or more cores, at 1.6GHz or faster (2GHz or faster for a stake pool or relay)
- **Memory:** 4GB of RAM (8GB for a relay or stake pool)
- **Storage:** 10GB of free storage (20GB for a stake pool)
- **Internet:** Broadband internet connection with speeds at least 10 Mbps.
- **Data Plan**: at least 1GB per hour. 720GB per month.
- **Power:** Reliable electrical power
- **ADA balance:** at least 505 ADA

Recommended Future-proof Stake Pool Hardware Setup

- **Three separate servers:** 1 for block producer node, 2 for relay nodes
- **One air-gapped offline machine (cold environment)**
- **Operating system:** 64-bit Linux (i.e. Ubuntu 20.04 LTS)
- **Processor:** 4 core or higher CPU
- **Memory:** 8GB+ RAM
- **Storage:** 256GB+ SSD
- **Internet:** Broadband internet connections with speeds at least 100 Mbps
- **Data Plan**: Unlimited
- **Power:** Reliable electrical power with UPS
- **ADA balance:** more pledge is better, to be determined by **a0**, the pledge influence factor

Recommended Stake Pool Security

If you need ideas on how to harden your stake pool's nodes, refer to

→ How to Harden Ubuntu Server /coins/overview-ada/guide-how-to-build-a-haskell-stakepool-node/how-to-harden-ubuntu-server

Setup Ubuntu

If you need to install **Ubuntu Server**, refer to

Install Ubuntu Server | Ubuntu

Ubuntu is an open source software operating system that runs from the desktop, to the cloud, to all your internet connected

ubuntu.com

For instructions on installing **Ubuntu Desktop**, refer to the following:

→ 2. Install Ubuntu /coins/overview-xtz/guide-how-to-setup-a-baker/install-ubuntu

Rebuilding Nodes

If you are rebuilding or reusing an existing `cardano-node` installation, refer to section 18.2 on how to reset the installation.

1. Install Cabal and GHC

If using Ubuntu Desktop, **press** Ctrl+Alt+T. This will launch a terminal window.

First, update packages and install Ubuntu dependencies.

```
sudo apt-get update -y
```

```
sudo apt-get upgrade -y
```

```
sudo apt-get install git jq bc make rsync htop curl build-essential pkg-confi
```

Install Libsodium.

```
mkdir $HOME/git
cd $HOME/git
git clone https://github.com/input-output-hk/libsodium
cd libsodium
git checkout 66f017f1
./autogen.sh
./configure
make
sudo make install
```

Install Cabal.

```
cd
wget https://downloads.haskell.org/~cabal/cabal-install-3.2.0.0/cabal-insta
tar -xf cabal-install-3.2.0.0-x86_64-unknown-linux.tar.xz
rm cabal-install-3.2.0.0-x86_64-unknown-linux.tar.xz cabal.sig
mkdir -p $HOME/.local/bin
mv cabal $HOME/.local/bin/
```

Install GHC.

```
wget https://downloads.haskell.org/ghc/8.10.2/ghc-8.10.2-x86_64-deb9-linux.t
tar -xf ghc-8.10.2-x86_64-deb9-linux.tar.xz
rm ghc-8.10.2-x86_64-deb9-linux.tar.xz
cd ghc-8.10.2
./configure
sudo make install
```

Update PATH to include Cabal and GHC and add exports. Your node's location will be in **$NODE_HOME**. The cluster configuration is set by **$NODE_CONFIG** and **$NODE_BUILD_NUM**.

```
echo PATH="$HOME/.local/bin:$PATH" >> $HOME/.bashrc
echo export LD_LIBRARY_PATH="/usr/local/lib:$LD_LIBRARY_PATH" >> $HOME/.bash
echo export NODE_HOME=$HOME/cardano-my-node >> $HOME/.bashrc
echo export NODE_CONFIG=mainnet>> $HOME/.bashrc
echo export NODE_BUILD_NUM=$(curl https://hydra.iohk.io/job/Cardano/iohk-ni
source $HOME/.bashrc
```

Update cabal and verify the correct versions were installed successfully.

```
cabal update
cabal -V
ghc -V
```

Cabal library should be version 3.2.0.0 and GHC should be version 8.10.2

2. Build the node from source code

Download source code and switch to the latest tag.

```
cd $HOME/git
git clone https://github.com/input-output-hk/cardano-node.git
cd cardano-node
git fetch --all --recurse-submodules --tags
git checkout tags/1.24.2
```

Configure build options.

```
cabal configure -O0 -w ghc-8.10.2
```

Update the cabal config, project settings, and reset build folder.

```
1 echo -e "package cardano-crypto-praos\n flags: -external-libsodium-vrf" > ca
2 sed -i $HOME/.cabal/config -e "s/overwrite-policy:/overwrite-policy: always,
3 rm -rf $HOME/git/cardano-node/dist-newstyle/build/x86_64-linux/ghc-8.10.2
```

Build the cardano-node from source code.

```
cabal build cardano-cli cardano-node
```

> Building process may take a few minutes up to a few hours depending on your computer's processing power.

Copy **cardano-cli** and **cardano-node** files into bin directory.

```
sudo cp $(find $HOME/git/cardano-node/dist-newstyle/build -type f -name "card
```

```
sudo cp $(find $HOME/git/cardano-node/dist-newstyle/build -type f -name "card
```

Verify your **cardano-cli** and **cardano-node** are the expected versions.

```
1 cardano-node version
2 cardano-cli version
```

3. Configure the nodes

Here you'll grab the config.json, genesis.json, and topology.json files needed to configure your node.

```
mkdir $NODE_HOME
cd $NODE_HOME
wget -N https://hydra.iohk.io/build/${NODE_BUILD_NUM}/download/1/${NODE_CONI
wget -N https://hydra.iohk.io/build/${NODE_BUILD_NUM}/download/1/${NODE_CONI
wget -N https://hydra.iohk.io/build/${NODE_BUILD_NUM}/download/1/${NODE_CONI
wget -N https://hydra.iohk.io/build/${NODE_BUILD_NUM}/download/1/${NODE_CONI
```

Run the following to modify **config.json** and

- update TraceBlockFetchDecisions to "true"

```
sed -i ${NODE_CONFIG}-config.json \
    -e "s/TraceBlockFetchDecisions\": false/TraceBlockFetchDecisions\": true
```

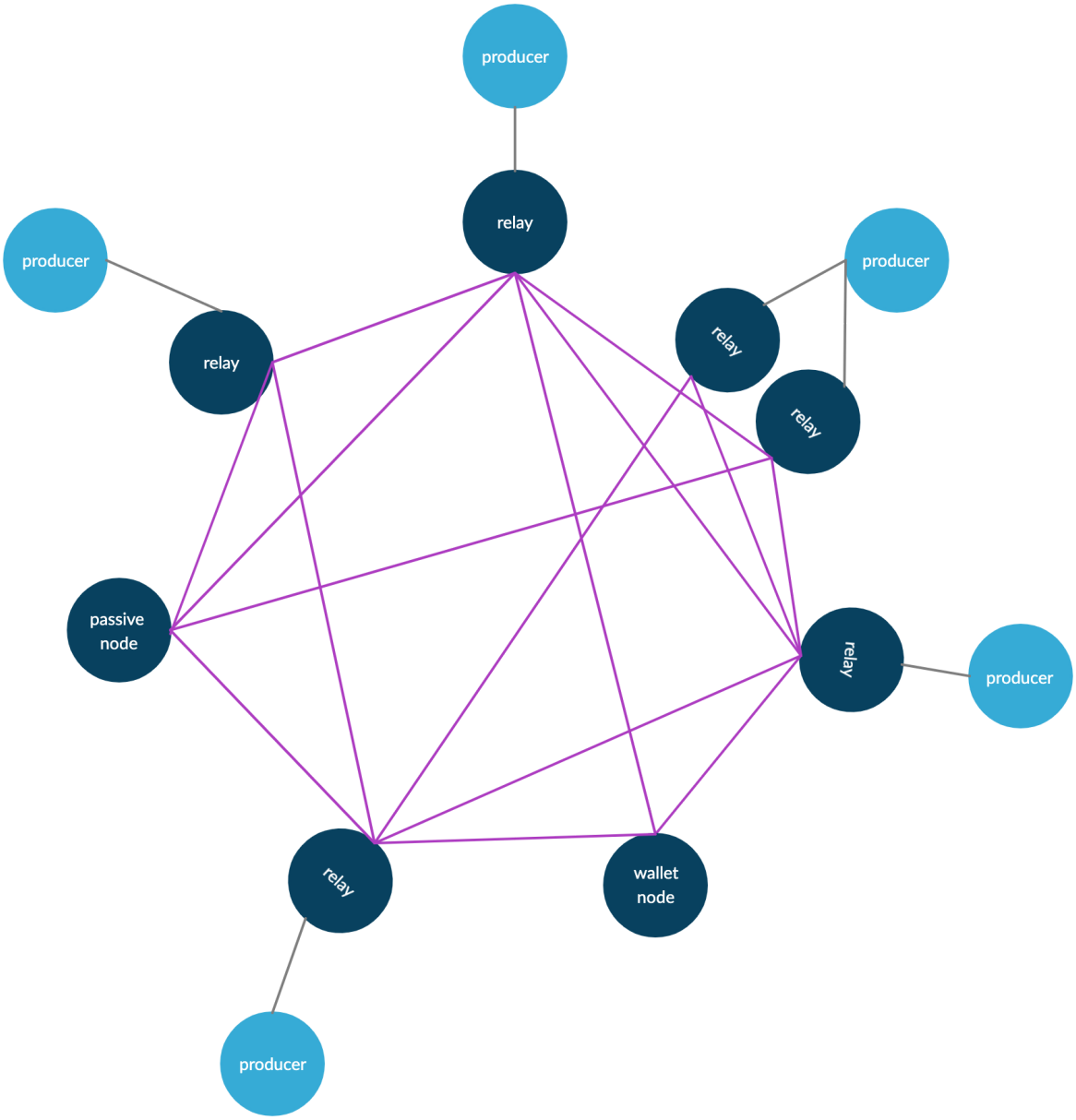
Update **.bashrc** shell variables.

```
echo export CARDANO_NODE_SOCKET_PATH="$NODE_HOME/db/socket" >> $HOME/.bashrc
source $HOME/.bashrc
```

4. Configure the block-producer node

> A block producer node will be configured with various key-pairs needed for block generation (cold keys, KES hot keys and VRF hot keys). It can only connect to its relay nodes.

A relay node will not be in possession of any keys and will therefore be unable to produce blocks. It will be connected to its block-producing node, other relays and external nodes.



For the purposes of this guide, we will be building **two nodes** on two **separate servers**. One node will be designated the **block producer node**, and the other will be the relay node, named **relaynode1**.

Configure **topology.json** file so that

- relay node(s) connect to public relay nodes (like IOHK and buddy relay nodes) and your block-producer node
- block-producer node **only** connects to your relay node(s)

On your **block-producer node,** run the following. Update the **addr** with your relay node's public IP address.

block producer node

```
cat > $NODE_HOME/${NODE_CONFIG}-topology.json << EOF
 {
    "Producers": [
      {
        "addr": "<RELAYNODE1'S PUBLIC IP ADDRESS>",
        "port": 6000,
        "valency": 1
      }
    ]
  }
EOF
```

5. Configure the relay node(s)

On your other server that will be designed as your relay node or what we will call **relaynode1** throughout this guide, carefully **repeat steps 1 through 3** in order to build the cardano binaries.

You can have multiple relay nodes as you scale up your stake pool architecture. Simply create **relaynodeN** and adapt the guide instructions accordingly.

On your **relaynode1,** run with the following after updating with your block producer's public IP address.

relaynode1

```
cat > $NODE_HOME/${NODE_CONFIG}-topology.json << EOF
 {
    "Producers": [
      {
        "addr": "<BLOCK PRODUCER NODE'S PUBLIC IP ADDRESS>",
        "port": 6000,
        "valency": 1
      },
      {
        "addr": "relays-new.cardano-mainnet.iohk.io",
        "port": 3001,
        "valency": 2
      }
    ]
  }
EOF
```

Valency tells the node how many connections to keep open. Only DNS addresses are affected. If value is 0, the address is ignored.

Port Forwarding Tip: You'll need to forward and open ports 6000 to your nodes. Check with https://www.yougetsignal.com/tools/open-ports/ or https://canyouseeme.org/ .

6. Configure the air-gapped offline machine

An air-gapped offline machine is called your cold environment.

- Protects against key-logging attacks, malware/virus based attacks and other firewall or security exploits.
- Physically isolated from the rest of your network.
- Must not have a network connection, wired or wireless.
- Is not a VM on a machine with a network connection.
- Learn more about air-gapping at wikipedia.

air-gapped offline machine

```
echo export NODE_HOME=$HOME/cardano-my-node >> $HOME/.bashrc
source $HOME/.bashrc
mkdir -p $NODE_HOME
```

Copy from your **hot environment**, also known as your block producer node, a copy of the `cardano-cli` binaries to your **cold environment**, this air-gapped offline machine.

⚠ In order to remain a true air-gapped environment, you must move files physically between your cold and hot environments with USB keys or other removable media.

7. Create startup scripts

The startup script contains all the variables needed to run a cardano-node such as directory, port, db path, config file, and topology file.

block producer node

```
cat > $NODE_HOME/startBlockProducingNode.sh << EOF
#!/bin/bash
DIRECTORY=$NODE_HOME
PORT=6000
HOSTADDR=0.0.0.0
TOPOLOGY=\${DIRECTORY}/${NODE_CONFIG}-topology.json
DB_PATH=\${DIRECTORY}/db
SOCKET_PATH=\${DIRECTORY}/db/socket
CONFIG=\${DIRECTORY}/${NODE_CONFIG}-config.json
cardano-node run --topology \${TOPOLOGY} --database-path \${DB_PATH} -
EOF
```

relaynode1

```
cat > $NODE_HOME/startRelayNode1.sh << EOF
#!/bin/bash
DIRECTORY=$NODE_HOME
PORT=6000
HOSTADDR=0.0.0.0
TOPOLOGY=\${DIRECTORY}/${NODE_CONFIG}-topology.json
DB_PATH=\${DIRECTORY}/db
SOCKET_PATH=\${DIRECTORY}/db/socket
CONFIG=\${DIRECTORY}/${NODE_CONFIG}-config.json
cardano-node run --topology \${TOPOLOGY} --database-path \${DB_PATH} -
EOF
```

Add execute permissions to the startup script.

block producer node

```
chmod +x $NODE_HOME/startBlockProducingNode.sh
```

relaynode1

```
chmod +x $NODE_HOME/startRelayNode1.sh
```

Run the following to create a **systemd unit file** to define your `cardano-node.service` configuration.

Benefits of using systemd for your stake pool

1. Auto-start your stake pool when the computer reboots due to maintenance, power outage, etc.
2. Automatically restart crashed stake pool processes.
3. Maximize your stake pool up-time and performance.

block producer node

```
cat > $NODE_HOME/cardano-node.service << EOF
# The Cardano node service (part of systemd)
# file: /etc/systemd/system/cardano-node.service

[Unit]
Description     = Cardano node service
Wants           = network-online.target
After           = network-online.target

[Service]
User            = ${USER}
Type            = simple
WorkingDirectory= ${NODE_HOME}
ExecStart       = /bin/bash -c '${NODE_HOME}/startBlockProducingNode.s
KillSignal=SIGINT
RestartKillSignal=SIGINT
TimeoutStopSec=2
LimitNOFILE=32768
Restart=always
RestartSec=5

[Install]
WantedBy  = multi-user.target
EOF
```

relaynode1

```
cat > $NODE_HOME/cardano-node.service << EOF
# The Cardano node service (part of systemd)
# file: /etc/systemd/system/cardano-node.service

[Unit]
Description     = Cardano node service
Wants           = network-online.target
After           = network-online.target

[Service]
User            = ${USER}
Type            = simple
WorkingDirectory= ${NODE_HOME}
ExecStart       = /bin/bash -c '${NODE_HOME}/startRelayNode1.sh'
KillSignal=SIGINT
RestartKillSignal=SIGINT
TimeoutStopSec=2
LimitNOFILE=32768
Restart=always
RestartSec=5

[Install]
WantedBy  = multi-user.target
EOF
```

Move the unit file to `/etc/systemd/system` and give it permissions.

```
sudo mv $NODE_HOME/cardano-node.service /etc/systemd/system/cardano-node.serv
```

```
sudo chmod 644 /etc/systemd/system/cardano-node.service
```

Run the following to enable auto-starting of your stake pool at boot time.

```
sudo systemctl daemon-reload
sudo systemctl enable cardano-node
```

> Your stake pool is now managed by the reliability and robustness of systemd. Below are some commands for using systemd.

View the status of the node service

```
sudo systemctl status cardano-node
```

Restarting the node service

```
sudo systemctl reload-or-restart cardano-node
```

Stopping the node service

```
sudo systemctl stop cardano-node
```

Viewing and filter logs

```
journalctl --unit=cardano-node --follow
```

```
journalctl --unit=cardano-node --since=yesterday
```

```
journalctl --unit=cardano-node --since=today
```

```
journalctl --unit=cardano-node --since='2020-07-29 00:00:00' --until='2020-07
```

8. Start the nodes

Start your stake pool with systemctl and begin syncing the blockchain!

block producer node

```
sudo systemctl start cardano-node
```

relaynode1

```
sudo systemctl start cardano-node
```

Install gLiveView, a monitoring tool.

> gLiveView displays crucial node status information and works well with systemd services. Credits to the Guild Operators for creating this tool.

```
cd $NODE_HOME
sudo apt install bc tcptraceroute -y
curl -s -o gLiveView.sh https://raw.githubusercontent.com/cardano-community/
curl -s -o env https://raw.githubusercontent.com/cardano-community/guild-ope
chmod 755 gLiveView.sh
```

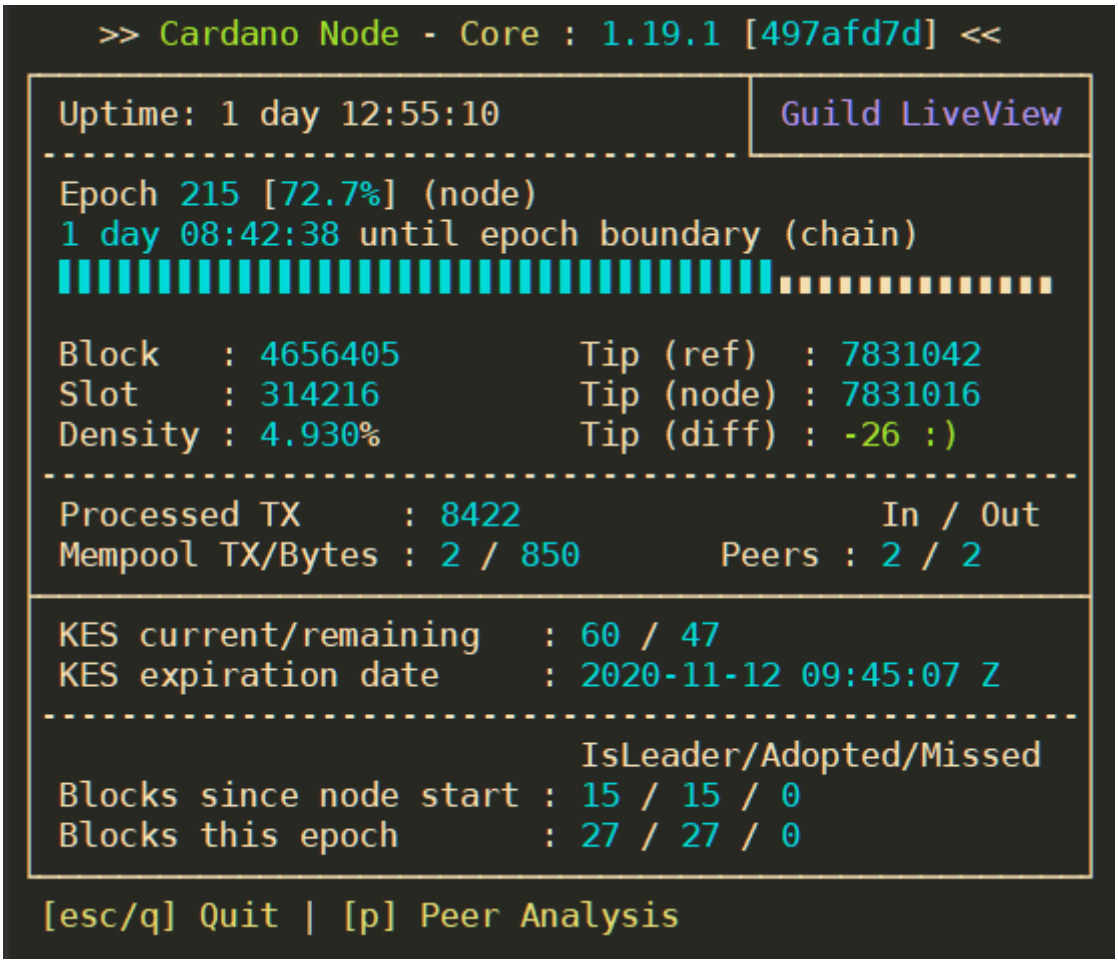
Run the following to modify **env** with the updated file locations.

```
sed -i env \
    -e "s/\#CONFIG=\"\${CNODE_HOME}\/files\/config.json\"/CONFIG=\"\${NODE_H
    -e "s/\#SOCKET=\"\${CNODE_HOME}\/sockets\/node0.socket\"/SOCKET=\"\${NOI
```

Run gLiveView to monitor the progress of the sync'ing of the blockchain.

```
./gLiveView.sh
```

Sample output of gLiveView.



Guild Live View

For more information, refer to the official Guild Live View docs.

Pro tip: If you synchronize a node's database, you can copy the database directory over to your other node directly and save time.

Congratulations! Your node is running successfully now. Let it sync up.

9. Generate block-producer keys

The block-producer node requires you to create 3 keys as defined in the Shelley ledger specs:

- stake pool cold key (node.cert)
- stake pool hot key (kes.skey)
- stake pool VRF key (vrf.skey)

First, make a KES key pair.

block producer node

```
cd $NODE_HOME
cardano-cli node key-gen-KES \
    --verification-key-file kes.vkey \
    --signing-key-file kes.skey
```

KES (key evolving signature) keys are created to secure your stake pool against hackers who might compromise your keys.

On mainnet, you will need to regenerate the KES key every 90 days.

Cold keys must be generated and stored on your air-gapped offline machine.

The cold keys are the files stored in `$HOME/cold-keys`.

Make a directory to store your cold keys

Air-gapped offline machine

```
mkdir $HOME/cold-keys
pushd $HOME/cold-keys
```

Make a set of cold keys and create the cold counter file.

air-gapped offline machine

```
cardano-cli node key-gen \
    --cold-verification-key-file node.vkey \
    --cold-signing-key-file node.skey \
    --operational-certificate-issue-counter node.counter
```

Be sure to **back up your all your keys** to another secure storage device. Make multiple copies.

Determine the number of slots per KES period from the genesis file.

block producer node

```
pushd +1
slotsPerKESPeriod=$(cat $NODE_HOME/${NODE_CONFIG}-shelley-genesis.jsor
echo slotsPerKESPeriod: ${slotsPerKESPeriod}
```

> Before continuing, your node must be fully synchronized to the blockchain. Otherwise, you won't calculate the latest KES period. Your node is synchronized when the *epoch* and *slot#* is equal to that found on a block explorer such as https://pooltool.io/

block producer node

```
slotNo=$(cardano-cli query tip --mainnet | jq -r '.slotNo')
echo slotNo: ${slotNo}
```

Find the kesPeriod by dividing the slot tip number by the slotsPerKESPeriod.

block producer node

```
kesPeriod=$((${slotNo} / ${slotsPerKESPeriod}))
echo kesPeriod: ${kesPeriod}
startKesPeriod=${kesPeriod}
echo startKesPeriod: ${startKesPeriod}
```

With this calculation, you can generate a operational certificate for your pool.

Copy **kes.vkey** to your **cold environment**.

Change the <**startKesPeriod**> value accordingly.

> Stake pool operators must provide an operational certificate to verify that the pool has the authority to run. The certificate includes the operator's signature, and includes key information about the pool (addresses, keys, etc.). Operational certificates represent the link between the operator's offline key and their operational key.

air-gapped offline machine

```
cardano-cli node issue-op-cert \
    --kes-verification-key-file kes.vkey \
    --cold-signing-key-file $HOME/cold-keys/node.skey \
    --operational-certificate-issue-counter $HOME/cold-keys/node.count
    --kes-period <startKesPeriod> \
    --out-file node.cert
```

Copy **node.cert** to your **hot environment**.

Make a VRF key pair.

block producer node

```
cardano-cli node key-gen-VRF \
    --verification-key-file vrf.vkey \
    --signing-key-file vrf.skey
```

Update vrf key permissions to read-only.

```
chmod 400 vrf.skey
```

Stop your stake pool by running the following:

block producer node

```
sudo systemctl stop cardano-node
```

Update your startup script with the new **KES, VRF and Operation Certificate.**

block producer node

```
cat > $NODE_HOME/startBlockProducingNode.sh << EOF
DIRECTORY=$NODE_HOME
PORT=6000
HOSTADDR=0.0.0.0
TOPOLOGY=\${DIRECTORY}/${NODE_CONFIG}-topology.json
DB_PATH=\${DIRECTORY}/db
SOCKET_PATH=\${DIRECTORY}/db/socket
CONFIG=\${DIRECTORY}/${NODE_CONFIG}-config.json
KES=\${DIRECTORY}/kes.skey
VRF=\${DIRECTORY}/vrf.skey
CERT=\${DIRECTORY}/node.cert
cardano-node run --topology \${TOPOLOGY} --database-path \${DB_PATH} -
EOF
```

> To operate a stake pool, you need the KES, VRF key and Operational Certificate. Cold keys generate new operational certificates periodically.

Now start your block producer node.

block producer node

```
sudo systemctl start cardano-node

# Monitor with gLiveView
./gLiveView.sh
```

10. Setup payment and stake keys

First, obtain the protocol-parameters.

> Wait for the block-producing node to start syncing before continuing if you get this error message.
>
> ```
> cardano-cli: Network.Socket.connect: : does not exist (No such file or directory)
> ```

block producer node

```
cardano-cli query protocol-parameters \
    --mainnet \
    --allegra-era \
    --out-file params.json
```

> Payment keys are used to send and receive payments and stake keys are used to manage stake delegations.

There are two ways to create your `payment` and `stake` key pair. Pick the one that best suits your needs.

Critical Operational Security Advice: `payment` and `stake` keys must be generated and used to build transactions in an cold environment. In other words, your **air-gapped offline machine**. Copy `cardano-cli` binary over to your offline machine and run the CLI method or mnemonic method. The only steps performed online in a hot environment are those steps that require live data. Namely the follow type of steps:

- querying the current slot tip
- querying the balance of an address
- submitting a transaction

CLI Method

Create a new payment key pair: `payment.skey` & `payment.vkey`

```
###
### On air-gapped offline machine,
###
cd $NODE_HOME
cardano-cli address key-gen \
    --verification-key-file payment.vkey \
    --signing-key-file payment.skey
```

Create a new stake address key pair: `stake.skey` & `stake.vkey`

```
###
### On air-gapped offline machine,
###
cardano-cli stake-address key-gen \
    --verification-key-file stake.vkey \
    --signing-key-file stake.skey
```

Create your stake address from the stake address verification key and store it in `stake.addr`

```
###
### On air-gapped offline machine,
###
cardano-cli stake-address build \
    --stake-verification-key-file stake.vkey \
    --out-file stake.addr \
    --mainnet
```

Build a payment address for the payment key `payment.vkey` which will delegate to the stake address, `stake.vkey`

```
###
### On air-gapped offline machine,
```

```
###
cardano-cli address build \
    --payment-verification-key-file payment.vkey \
    --stake-verification-key-file stake.vkey \
    --out-file payment.addr \
    --mainnet
```

Mnemonic Method

Credits to ilap for creating this process.

Benefits: Track and control pool rewards from any wallet (Daedalus, Yoroi or any other wallet) that support stakings.

Create a 15-word or 24-word length shelley compatible mnemonic with Daedalus or Yoroi on a offline machine preferred.

Using your online block producer node, download `cardano-wallet`

```
###
### On block producer node,
###
cd $NODE_HOME
wget https://hydra.iohk.io/build/3662127/download/1/cardano-wallet-she
```

Verify the legitimacy of `cardano-wallet` by checking the sha256 hash found in the **Details** button.

```
echo "f75e5b2b4cc5f373d6b1c1235818bcab696d86232cb2c5905b2d91b4805bae84
```

Example valid output:

> cardano-wallet-shelley-2020.7.28-linux64.tar.gz: OK

Only proceed if the sha256 check passes with **OK**!

Transfer the **cardano-wallet** to your **air-gapped offline machine** via USB key or other removable media.

Extract the wallet files and cleanup.

```
###
### On air-gapped offline machine,
###
tar -xvf cardano-wallet-shelley-2020.7.28-linux64.tar.gz
rm cardano-wallet-shelley-2020.7.28-linux64.tar.gz
```

Create `extractPoolStakingKeys.sh` script.

```
###
### On air-gapped offline machine,
###
cat > extractPoolStakingKeys.sh << HERE
#!/bin/bash

CADDR=\${CADDR:=\$( which cardano-address )}
[[ -z "\$CADDR" ]] && ( echo "cardano-address cannot be found, exiting
CCLI=\${CCLI:=\$( which cardano-cli )}
[[ -z "\$CCLI" ]] && ( echo "cardano-cli cannot be found, exiting..."

OUT_DIR="\$1"
[[ -e "\$OUT_DIR"  ]] && {
        echo "The \"\$OUT_DIR\" is already exist delete and run again.
        exit 127
} || mkdir -p "\$OUT_DIR" && pushd "\$OUT_DIR" >/dev/null

shift
MNEMONIC="\$*"

# Generate the master key from mnemonics and derive the stake account
# as extended private and public keys (xpub, xprv)
echo "\$MNEMONIC" |\
"\$CADDR" key from-recovery-phrase Shelley > root.prv

cat root.prv |\
"\$CADDR" key child 1852H/1815H/0H/2/0 > stake.xprv

cat root.prv |\
"\$CADDR" key child 1852H/1815H/0H/0/0 > payment.xprv

TESTNET=0
MAINNET=1
```

```
NETWORK=\$MAINNET

cat payment.xprv |\
"\$CADDR" key public | tee payment.xpub |\
"\$CADDR" address payment --network-tag \$NETWORK |\
"\$CADDR" address delegation \$(cat stake.xprv | "\$CADDR" key public
tee base.addr_candidate |\
"\$CADDR" address inspect
echo "Generated from 1852H/1815H/0H/{0,2}/0"
cat base.addr_candidate
echo

# XPrv/XPub conversion to normal private and public key, keep in mind
# keypars are not a valind Ed25519 signing keypairs.
TESTNET_MAGIC="--testnet-magic 42"
MAINNET_MAGIC="--mainnet"
MAGIC="\$MAINNET_MAGIC"

SESKEY=\$( cat stake.xprv | bech32 | cut -b -128 )\$( cat stake.xpub
PESKEY=\$( cat payment.xprv | bech32 | cut -b -128 )\$( cat payment.xp

cat << EOF > stake.skey
{
    "type": "StakeExtendedSigningKeyShelley_ed25519_bip32",
    "description": "",
    "cborHex": "5880\$SESKEY"
}
EOF

cat << EOF > payment.skey
{
    "type": "PaymentExtendedSigningKeyShelley_ed25519_bip32",
    "description": "Payment Signing Key",
    "cborHex": "5880\$PESKEY"
}
EOF

"\$CCLI" shelley key verification-key --signing-key-file stake.skey --
"\$CCLI" shelley key verification-key --signing-key-file payment.skey

"\$CCLI" shelley key non-extended-key --extended-verification-key-file
"\$CCLI" shelley key non-extended-key --extended-verification-key-file


"\$CCLI" shelley stake-address build --stake-verification-key-file sta
"\$CCLI" shelley address build --payment-verification-key-file payment
"\$CCLI" shelley address build \
    --payment-verification-key-file payment.vkey \
    --stake-verification-key-file stake.vkey \
    \$MAGIC > base.addr

echo "Important the base.addr and the base.addr_candidate must be the
diff base.addr base.addr_candidate
popd >/dev/null
HERE
```

Add permissions and export PATH to use the binaries.

```
###
### On air-gapped offline machine,
###
chmod +x extractPoolStakingKeys.sh
export PATH="$(pwd)/cardano-wallet-shelley-2020.7.28:$PATH"
```

Extract your keys. Update the command with your mnemonic phrase.

```
###
### On air-gapped offline machine,
###
./extractPoolStakingKeys.sh extractedPoolKeys/ <15|24-word length mner
```

> ⚠ **Important**: The **base.addr** and the **base.addr_candidate** must be the same. Review the screen output.

Your new staking keys are in the folder `extractedPoolKeys/`

Now move `payment/stake` key pair over to your `$NODE_HOME` for use with your stake pool.

```
###
### On air-gapped offline machine,
###
cd extractedPoolKeys/
cp stake.vkey stake.skey stake.addr payment.vkey payment.skey base.adc
cd $NODE_HOME
#Rename to base.addr file to payment.addr
mv base.addr payment.addr
```

> **payment.addr**, or also known as base.addr from this extraction script, will be the cardano address which holds your pool's pledge.

Clear the bash history in order to protect your mnemonic phrase and remove the `cardano-wallet` files.

```
###
### On air-gapped offline machine,
###
history -c && history -w
rm -rf $NODE_HOME/cardano-wallet-shelley-2020.7.28
```

Finally close all your terminal windows and open new ones with zero history.

> Awesome. Now you can track your pool rewards in your wallet.

Next step is to fund your payment address.

Copy **payment.addr** to your **hot environment**.

Payment address can be funded from your Daedalus / Yoroi wallet.

Run the following to find your payment address.

```
cat payment.addr
```

After funding your account, check your payment address balance.

> ⚠ Before continuing, your nodes must be fully synchronized to the blockchain. Otherwise, you won't see your funds.

block producer node

```
cardano-cli query utxo \
    --address $(cat payment.addr) \
    --allegra-era \
    --mainnet
```

You should see output similar to this. This is your unspent transaction output (UXTO).

```
                           TxHash                                 TxIx
--------------------------------------------------------------------------------
100322a39d02c2ead....                                              0
```

11. Register your stake address

Create a certificate, `stake.cert` , using the `stake.vkey`

air-gapped offline machine

```
cardano-cli stake-address registration-certificate \
    --stake-verification-key-file stake.vkey \
    --out-file stake.cert
```

Copy **stake.cert** to your **hot environment.**

You need to find the **tip** of the blockchain to set the **invalid-hereafter** parameter properly.

block producer node

```
currentSlot=$(cardano-cli query tip --mainnet | jq -r '.slotNo')
echo Current Slot: $currentSlot
```

Find your balance and **UTXOs**.

block producer node

```
cardano-cli query utxo \
    --address $(cat payment.addr) \
    --allegra-era \
    --mainnet > fullUtxo.out

tail -n +3 fullUtxo.out | sort -k3 -nr > balance.out

cat balance.out

tx_in=""
total_balance=0
while read -r utxo; do
    in_addr=$(awk '{ print $1 }' <<< "${utxo}")
    idx=$(awk '{ print $2 }' <<< "${utxo}")
    utxo_balance=$(awk '{ print $3 }' <<< "${utxo}")
    total_balance=$((${total_balance}+${utxo_balance}))
    echo TxHash: ${in_addr}#${idx}
    echo ADA: ${utxo_balance}
    tx_in="${tx_in} --tx-in ${in_addr}#${idx}"
done < balance.out
txcnt=$(cat balance.out | wc -l)
echo Total ADA balance: ${total_balance}
echo Number of UTXOs: ${txcnt}
```

Find the keyDeposit value.

block producer node

```
keyDeposit=$(cat $NODE_HOME/params.json | jq -r '.keyDeposit')
echo keyDeposit: $keyDeposit
```

Registration of a stake address certificate (keyDeposit) costs 2000000 lovelace.

Run the build-raw transaction command

The **invalid-hereafter** value must be greater than the current tip. In this example, we use current slot + 10000.

block producer node

```
cardano-cli transaction build-raw \
    ${tx_in} \
    --tx-out $(cat payment.addr)+0 \
    --invalid-hereafter $(( ${currentSlot} + 10000)) \
    --fee 0 \
    --out-file tx.tmp \
    --allegra-era \
    --certificate stake.cert
```

Calculate the current minimum fee:

block producer node

```
fee=$(cardano-cli transaction calculate-min-fee \
    --tx-body-file tx.tmp \
    --tx-in-count ${txcnt} \
    --tx-out-count 1 \
    --mainnet \
    --witness-count 2 \
    --byron-witness-count 0 \
    --protocol-params-file params.json | awk '{ print $1 }')
echo fee: $fee
```

Ensure your balance is greater than cost of fee + keyDeposit or this will not work.

Calculate your change output.

block producer node

```
txOut=$((${total_balance}-${keyDeposit}-${fee}))
echo Change Output: ${txOut}
```

Build your transaction which will register your stake address.

block producer node

```
cardano-cli transaction build-raw \
    ${tx_in} \
    --tx-out $(cat payment.addr)+${txOut} \
    --invalid-hereafter $(( ${currentSlot} + 10000)) \
    --fee ${fee} \
    --certificate-file stake.cert \
    --allegra-era \
    --out-file tx.raw
```

Copy **tx.raw** to your **cold environment**.

Sign the transaction with both the payment and stake secret keys.

air-gapped offline machine

```
cardano-cli transaction sign \
    --tx-body-file tx.raw \
    --signing-key-file payment.skey \
    --signing-key-file stake.skey \
    --mainnet \
    --out-file tx.signed
```

Copy **tx.signed** to your **hot environment.**

Send the signed transaction.

block producer node

```
cardano-cli transaction submit \
    --tx-file tx.signed \
    --mainnet
```

12. Register your stake pool

Create your pool's metadata with a JSON file. Update with your pool information.

> **ticker** must be between 3-5 characters in length. Characters must be A-Z and 0-9 only.

> **description** cannot exceed 255 characters in length.

block producer node

```
cat > poolMetaData.json << EOF
{
"name": "MyPoolName",
"description": "My pool description",
"ticker": "MPN",
"homepage": "https://myadapoolnamerocks.com"
}
EOF
```

Calculate the hash of your metadata file.

block producer node

```
cardano-cli stake-pool metadata-hash --pool-metadata-file poolMetaData.
```

Now upload your **poolMetaData.json** to your website or a public website such as https://pages.github.com/

Refer to the following quick guide if you need help hosting your metadata on github.com

→ How to upload poolMetaData.json to Github /coins/overview-ada/guide-how-to-build-a-haskell-stakepool-node/how-to-upload-poolmetadata.json-to-github

Find the minimum pool cost.

block producer node

```
minPoolCost=$(cat $NODE_HOME/params.json | jq -r .minPoolCost)
echo minPoolCost: ${minPoolCost}
```

ⓘ minPoolCost is 340000000 lovelace or 340 ADA. Therefore, your `--pool-cost` must be at a minimum this amount.

Create a registration certificate for your stake pool. Update with your **metadata URL** and your **relay node information**. Choose one of the three options available to configure relay nodes -- DNS based, Round Robin DNS based, or IP based.

ⓘ DNS based relays are recommended for simplicity of node management. In other words, you don't need to re-submit this **registration certificate** transaction every time your IP changes. Also you can easily update the DNS to point towards a new IP should you re-locate or re-build a relay node, for example.

ⓘ **How to configure multiple relay nodes.**

Update the next operation

```
cardano-cli stake-pool registration-certificate
```

to be run on your air-gapped offline machine appropriately.

DNS based relays, 1 entry per DNS record

```
    --single-host-pool-relay relaynode1.myadapoolnamerocks.com\
    --pool-relay-port 6000 \
    --single-host-pool-relay relaynode2.myadapoolnamerocks.com\
    --pool-relay-port 6000 \
```

Round Robin DNS based relays, 1 entry per SRV DNS record

```
    --multi-host-pool-relay relayNodes.myadapoolnamerocks.com\
    --pool-relay-port 6000 \
```

IP based relays, 1 entry per IP address

```
    --pool-relay-port 6000 \
    --pool-relay-ipv4 <your first relay node public IP address> \
    --pool-relay-port 6000 \
    --pool-relay-ipv4 <your second relay node public IP address> \
```

metadata-url must be no longer than 64 characters.

air-gapped offline machine

```
cardano-cli stake-pool registration-certificate \
    --cold-verification-key-file $HOME/cold-keys/node.vkey \
    --vrf-verification-key-file vrf.vkey \
    --pool-pledge 100000000 \
    --pool-cost 345000000 \
    --pool-margin 0.15 \
    --pool-reward-account-verification-key-file stake.vkey \
    --pool-owner-stake-verification-key-file stake.vkey \
    --mainnet \
    --single-host-pool-relay <dns based relay, example ~ relaynode1.my
    --pool-relay-port 6000 \
    --metadata-url <url where you uploaded poolMetaData.json> \
    --metadata-hash $(cat poolMetaDataHash.txt) \
    --out-file pool.cert
```

Here we are pledging 100 ADA with a fixed pool cost of 345 ADA and a pool margin of 15%.

Copy **pool.cert** to your **hot environment.**

Pledge stake to your stake pool.

air-gapped offline machine

```
cardano-cli stake-address delegation-certificate \
    --stake-verification-key-file stake.vkey \
    --cold-verification-key-file $HOME/cold-keys/node.vkey \
    --out-file deleg.cert
```

Copy **deleg.cert** to your **hot environment**.

This operation creates a delegation certificate which delegates funds from all stake addresses associated with key `stake.vkey` to the pool belonging to cold key `node.vkey`

A stake pool owner's promise to fund their own pool is called **Pledge**.

- Your balance needs to be greater than the pledge amount.
- You pledge funds are not moved anywhere. In this guide's example, the pledge remains in the stake pool's owner keys, specifically `payment.addr`
- Failing to fulfill pledge will result in missed block minting opportunities and your delegators would miss rewards.
- Your pledge is not locked up. You are free to transfer your funds.

You need to find the **tip** of the blockchain to set the **invalid-hereafter** parameter properly.

block producer node

```
currentSlot=$(cardano-cli query tip --mainnet | jq -r '.slotNo')
echo Current Slot: $currentSlot
```

Find your balance and **UTXOs**.

block producer node

```
cardano-cli query utxo \
    --address $(cat payment.addr) \
    --allegra-era \
    --mainnet > fullUtxo.out

tail -n +3 fullUtxo.out | sort -k3 -nr > balance.out

cat balance.out

tx_in=""
total_balance=0
while read -r utxo; do
    in_addr=$(awk '{ print $1 }' <<< "${utxo}")
    idx=$(awk '{ print $2 }' <<< "${utxo}")
    utxo_balance=$(awk '{ print $3 }' <<< "${utxo}")
    total_balance=$((${total_balance}+${utxo_balance}))
    echo TxHash: ${in_addr}#${idx}
    echo ADA: ${utxo_balance}
    tx_in="${tx_in} --tx-in ${in_addr}#${idx}"
done < balance.out
txcnt=$(cat balance.out | wc -l)
echo Total ADA balance: ${total_balance}
echo Number of UTXOs: ${txcnt}
```

Find the deposit fee for a pool.

block producer node

```
poolDeposit=$(cat $NODE_HOME/params.json | jq -r '.poolDeposit')
echo poolDeposit: $poolDeposit
```

Run the build-raw transaction command.

> The **invalid-hereafter** value must be greater than the current tip. In this example, we use current slot + 10000.

block producer node

```
cardano-cli transaction build-raw \
    ${tx_in} \
    --tx-out $(cat payment.addr)+$(( ${total_balance} - ${poolDeposit}
    --invalid-hereafter $(( ${currentSlot} + 10000)) \
    --fee 0 \
    --certificate-file pool.cert \
    --certificate-file deleg.cert \
    --allegra-era \
    --out-file tx.tmp
```

Calculate the minimum fee:

block producer node

```
fee=$(cardano-cli transaction calculate-min-fee \
    --tx-body-file tx.tmp \
    --tx-in-count ${txcnt} \
    --tx-out-count 1 \
    --mainnet \
    --witness-count 3 \
    --byron-witness-count 0 \
    --protocol-params-file params.json | awk '{ print $1 }')
echo fee: $fee
```

Ensure your balance is greater than cost of fee + minPoolCost or this will not work.

Calculate your change output.

block producer node

```
txOut=$((${total_balance}-${poolDeposit}-${fee}))
echo txOut: ${txOut}
```

Build the transaction.

block producer node

```
cardano-cli transaction build-raw \
    ${tx_in} \
    --tx-out $(cat payment.addr)+${txOut} \
    --invalid-hereafter $(( ${currentSlot} + 10000)) \
    --fee ${fee} \
    --certificate-file pool.cert \
    --certificate-file deleg.cert \
    --allegra-era \
    --out-file tx.raw
```

Copy **tx.raw** to your **cold environment.**

Sign the transaction.

air-gapped offline machine

```
cardano-cli transaction sign \
    --tx-body-file tx.raw \
    --signing-key-file payment.skey \
    --signing-key-file $HOME/cold-keys/node.skey \
    --signing-key-file stake.skey \
    --mainnet \
    --out-file tx.signed
```

Copy **tx.signed** to your **hot environment.**

Send the transaction.

block producer node

```
cardano-cli transaction submit \
    --tx-file tx.signed \
    --mainnet
```

13. Locate your Stake pool ID and verify everything is working

Your stake pool ID can be computed with:

air-gapped offline machine

```
cardano-cli stake-pool id --cold-verification-key-file $HOME/cold-keys
cat stakepoolid.txt
```

Copy **stakepoolid.txt** to your **hot environment.**

Now that you have your stake pool ID, verify it's included in the blockchain.

block producer node

```
cardano-cli query ledger-state --mainnet --allegra-era | grep publicKey
```

> A non-empty string return means you're registered!

With your stake pool ID, now you can find your data on block explorers such as https://pooltool.io/

14. Configure your topology files

> Shelley has been launched without peer-to-peer (p2p) node discovery so that means we will need to manually add trusted nodes in order to configure our topology. This is a **critical step** as skipping this step will result in your minted blocks being orphaned by the rest of the network.

There are two ways to configure your topology files.

- **topologyUpdate.sh method** is automated and works after 4 hours.
- **Pooltool.io method** gives you control over who your nodes connect to.

topologyUpdater.sh Method

Publishing your Relay Node with topologyUpdater.sh

> Credits to GROWPOOL for this addition and credits to CNTOOLS Guild OPS on creating this process.

Create the `topologyUpdater.sh` script which publishes your node information to a topology fetch list.

```
###
### On relaynode1
###
cat > $NODE_HOME/topologyUpdater.sh << EOF
#!/bin/bash
# shellcheck disable=SC2086,SC2034

USERNAME=$(whoami)
CNODE_PORT=6000 # must match your relay node port as set in the startu
CNODE_HOSTNAME="CHANGE ME"  # optional. must resolve to the IP you are
CNODE_BIN="/usr/local/bin"
CNODE_HOME=$NODE_HOME
CNODE_LOG_DIR="\${CNODE_HOME}/logs"
GENESIS_JSON="\${CNODE_HOME}/${NODE_CONFIG}-shelley-genesis.json"
NETWORKID=\$(jq -r .networkId \$GENESIS_JSON)
CNODE_VALENCY=1   # optional for multi-IP hostnames
NWMAGIC=\$(jq -r .networkMagic < \$GENESIS_JSON)
[[ "\${NETWORKID}" = "Mainnet" ]] && HASH_IDENTIFIER="--mainnet" || HA
[[ "\${NWMAGIC}" = "764824073" ]] && NETWORK_IDENTIFIER="--mainnet" |

export PATH="\${CNODE_BIN}:\${PATH}"
export CARDANO_NODE_SOCKET_PATH="\${CNODE_HOME}/db/socket"

blockNo=\$(/usr/local/bin/cardano-cli query tip \${NETWORK_IDENTIFIER}

# Note:
# if you run your node in IPv4/IPv6 dual stack network configuration a
# IPv4 address only please add the -4 parameter to the curl command be
if [ "\${CNODE_HOSTNAME}" != "CHANGE ME" ]; then
  T_HOSTNAME="&hostname=\${CNODE_HOSTNAME}"
else
  T_HOSTNAME=''
fi

if [ ! -d \${CNODE_LOG_DIR} ]; then
```

```
  mkdir -p \${CNODE_LOG_DIR};
fi

curl -s "https://api.clio.one/htopology/v1/?port=\${CNODE_PORT}&blockN
EOF
```

Add permissions and run the updater script.

```
###
### On relaynode1
###
cd $NODE_HOME
chmod +x topologyUpdater.sh
./topologyUpdater.sh
```

When the `topologyUpdater.sh` runs successfully, you will see

```
{ "resultcode": "201", "datetime":"2020-07-28 01:23:45", "clientIp":
"1.2.3.4", "iptype": 4, "msg": "nice to meet you" }
```

> Every time the script runs and updates your IP, a log is created in **`$NODE_HOME/logs`**

Add a crontab job to automatically run `topologyUpdater.sh` every hour on the 22nd minute. You can change the 22 value to your own preference.

```
###
### On relaynode1
###
cat > $NODE_HOME/crontab-fragment.txt << EOF
22 * * * * ${NODE_HOME}/topologyUpdater.sh
EOF
crontab -l | cat - crontab-fragment.txt >crontab.txt && crontab cronta
rm crontab-fragment.txt
```

> ✅ After four hours and four updates, your node IP will be registered in the topology fetch list.

Update your relay node topology files

> ⚠️ Complete this section after **four hours** when your relay node IP is properly registered.

Create `relay-topology_pull.sh` script which fetches your relay node buddies and updates your topology file. **Update with your block producer's public IP address.**

```
###
### On relaynode1
###
cat > $NODE_HOME/relay-topology_pull.sh << EOF
#!/bin/bash
BLOCKPRODUCING_IP=<BLOCK PRODUCERS PUBLIC IP ADDRESS>
BLOCKPRODUCING_PORT=6000
curl -s -o $NODE_HOME/${NODE_CONFIG}-topology.json "https://api.clio.
EOF
```

Add permissions and pull new topology files.

```
###
### On relaynode1
###
chmod +x relay-topology_pull.sh
./relay-topology_pull.sh
```

The new topology takes after after restarting your stake pool.

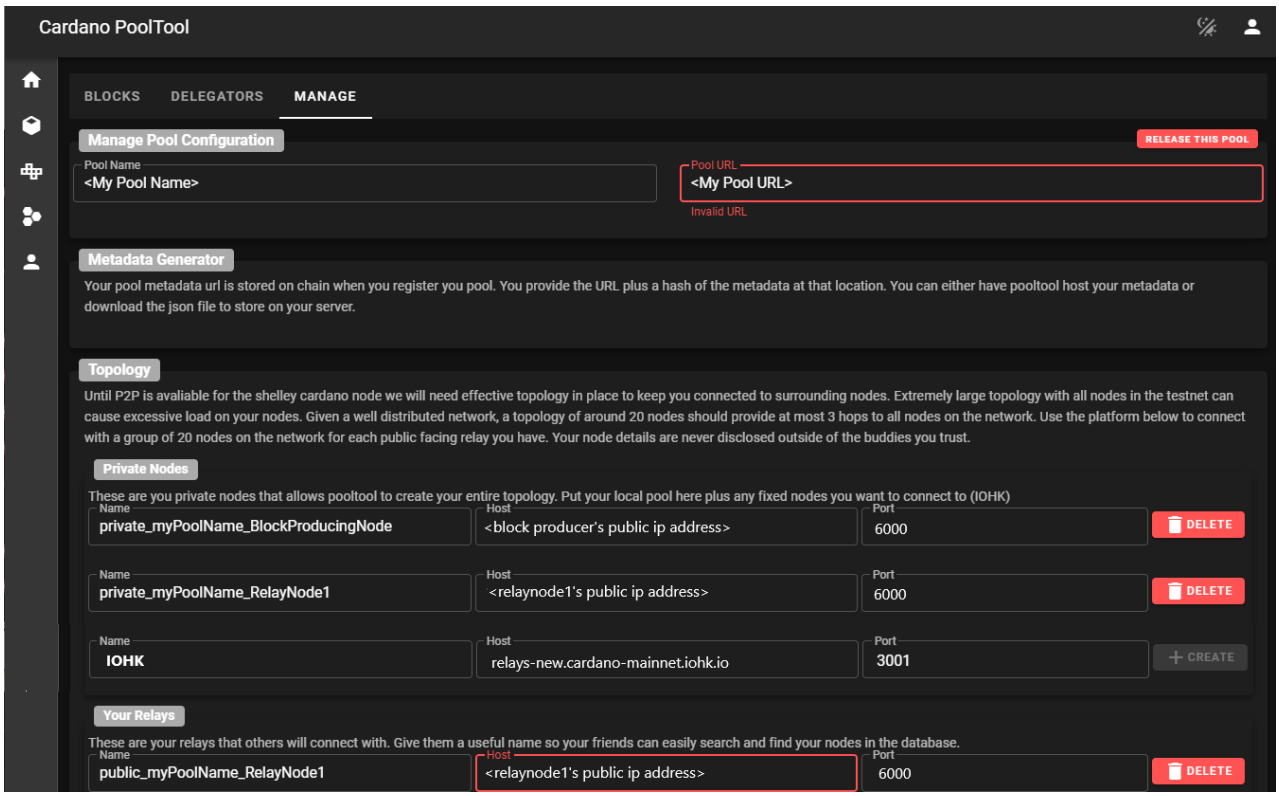
```
###
### On relaynode1
###
```

```
4  sudo systemctl restart cardano-node
```

Don't forget to restart your relay nodes after every time you fetch the topology!

Pooltool.io Method

1. Visit https://pooltool.io/
2. Create an account and login
3. Search for your stakepool id
4. Click **Pool Details** > **Manage** > **CLAIM THIS POOL**
5. Fill in your pool name and pool URL if you have one.
6. Fill in your **Private Nodes** and **Your Relays** as follows.



> You can find your public IP with https://www.whatismyip.com/ or
>
> ```
> curl http://ifconfig.me/ip
> ```

Add requests for nodes or "buddies" to each of your relay nodes. Make sure you include the IOHK node and your private nodes.

IOHK's node address is:

```
relays-new.cardano-mainnet.iohk.io
```

IOHK's node port is:

```
3001
```

For example, on relaynode1's buddies you should add **requests** for

- your private BlockProducingNode
- IOHK's node
- and any other buddy/friendly nodes your can find or know

> A relay node connection is not established until there is a request and an approval.

For **relaynode1**, create a get_buddies.sh script to update your topology.json file.

```
###
### On relaynode1
###
cat > $NODE_HOME/get_buddies.sh << EOF
#!/usr/bin/env bash

# YOU CAN PASS THESE STRINGS AS ENVIRONMENTAL VARIABLES, OR EDIT THEM
if [ -z "\$PT_MY_POOL_ID" ]; then
## CHANGE THESE TO SUIT YOUR POOL TO YOUR POOL ID AS ON THE EXPLORER
PT_MY_POOL_ID="XXXXXXXX"
fi

if [ -z "\$PT_MY_API_KEY" ]; then
## GET THIS FROM YOUR ACCOUNT PROFILE PAGE ON POOLTOOL WEBSITE
PT_MY_API_KEY="XXXXXXXX"
fi

if [ -z "\$PT_MY_NODE_ID" ]; then
## GET THIS FROM YOUR POOL MANAGE TAB ON POOLTOOL WEBSITE
PT_MY_NODE_ID="XXXXXXXX"
fi

```

```
if [ -z "\$PT_TOPOLOGY_FILE" ]; then
## SET THIS TO THE LOCATION OF YOUR TOPOLOGY FILE THAT YOUR NODE USES
PT_TOPOLOGY_FILE="$NODE_HOME/${NODE_CONFIG}-topology.json"
fi

JSON="\$(jq -n --compact-output --arg MY_API_KEY "\$PT_MY_API_KEY" --a
echo "Packet Sent: \$JSON"
RESPONSE="\$(curl -s -H "Accept: application/json" -H "Content-Type:ap
SUCCESS="\$(echo \$RESPONSE | jq '.success')"
if [ \$SUCCESS ]; then
  echo "Success"
  echo \$RESPONSE | jq '. | {Producers: .message}' > \$PT_TOPOLOGY_FIl
  echo "Topology saved to \$PT_TOPOLOGY_FILE.  Note topology will only
else
  echo "Failure "
  echo \$RESPONSE | jq '.message'
fi
EOF
```

For each of your relay nodes, update the following variables from pooltool.io into your get_buddies.sh file

- PT_MY_POOL_ID
- PT_MY_API_KEY
- PT_MY_NODE_ID

Update your get_buddies.sh scripts with this information.

> Use **nano** to edit your files.
>
> ```
> nano $NODE_HOME/relaynode1/get_buddies.sh
> ```

Add execute permissions to these scripts. Run the scripts to update your topology files.

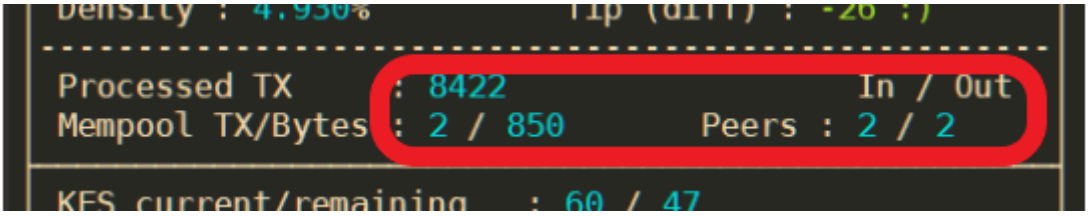
```
###
### On relaynode1
###
cd $NODE_HOME
chmod +x get_buddies.sh
./get_buddies.sh
```

Stop and then restart your stakepool in order for the new topology settings to take effect.

```
###
### On relaynode1
###
sudo systemctl restart cardano-node
```

As your REQUESTS are approved, you must re-run the get_buddies.sh script to pull the latest topology data. Restart your relay nodes afterwards.

Critical step: In order to be a functional stake pool ready to mint blocks, you must see the **Processed TX** number increasing in gLiveView. If not, review your topology file and ensure your **peers** (or relay buddies) are well connected and ideally, minted some blocks.



Processed TX must be positive in gLiveView. Must also have in / out connections.

Critical Key Security Reminder: The only stake pool **keys** and **certs** that are required to run a stake pool are those required by the block producer. Namely, the following three files.

```
###
### On block producer node
###
KES=\${DIRECTORY}/kes.skey
VRF=\${DIRECTORY}/vrf.skey
```

```
CERT=\${DIRECTORY}/node.cert
```

All other keys must remain offline in your air-gapped offline cold environment.

⚠ **Relay Node Security Reminder:** Relay nodes must not contain any `operational certifications` , `vrf` , `skey` **or** `cold` **keys**.

Congratulations! Your stake pool is registered and ready to produce blocks.

15. Checking Stake pool Rewards

After the epoch is over and assuming you successfully minted blocks, check with this:

block producer node

```
cardano-cli query stake-address-info \
 --address $(cat stake.addr) \
 --allegra-era \
 --mainnet
```

16. Setup Prometheus and Grafana Dashboard

Prometheus is a monitoring platform that collects metrics from monitored targets by scraping metrics HTTP endpoints on these targets. Official documentation is available here. Grafana is a dashboard used to visualize the collected data.

16.1 Installation

Install prometheus and prometheus node exporter.

relaynode1

```
sudo apt-get install -y prometheus prometheus-node-exporter
```

block producer node

```
sudo apt-get install -y prometheus-node-exporter
```

Install grafana.

relaynode1

```
wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -
```

relaynode1

```
1 echo "deb https://packages.grafana.com/oss/deb stable main" > grafana.
2 sudo mv grafana.list /etc/apt/sources.list.d/grafana.list
```

relaynode1

```
sudo apt-get update && sudo apt-get install -y grafana
```

Enable services so they start automatically.

relaynode1

```
1  sudo systemctl enable grafana-server.service
2  sudo systemctl enable prometheus.service
3  sudo systemctl enable prometheus-node-exporter.service
```

block producer node

```
sudo systemctl enable prometheus-node-exporter.service
```

Update **prometheus.yml** located in `/etc/prometheus/prometheus.yml`

Change the **<block producer public ip address>** in the following command.

relaynode1

```
cat > prometheus.yml << EOF
global:
  scrape_interval:     15s # By default, scrape targets every 15 secor

  # Attach these labels to any time series or alerts when communicatir
  # external systems (federation, remote storage, Alertmanager).
  external_labels:
    monitor: 'codelab-monitor'

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label job=<job_name> to any timeseries
  - job_name: 'prometheus'

    static_configs:
      - targets: ['localhost:9100']
      - targets: ['<block producer public ip address>:9100']
      - targets: ['<block producer public ip address>:12798']
        labels:
          alias: 'block-producer-node'
          type:  'cardano-node'
      - targets: ['localhost:12798']
        labels:
          alias: 'relaynode1'
          type:  'cardano-node'
EOF
sudo mv prometheus.yml /etc/prometheus/prometheus.yml
```

Finally, restart the services.

relaynode1

```
sudo systemctl restart grafana-server.service
sudo systemctl restart prometheus.service
sudo systemctl restart prometheus-node-exporter.service
```

Verify that the services are running properly:

relaynode1

```
sudo systemctl status grafana-server.service prometheus.service prometh
```

Update `${NODE_CONFIG}-config.json` config files with new `hasEKG` and `hasPrometheus` ports.

block producer node

```
cd $NODE_HOME

sed -i ${NODE_CONFIG}-config.json -e "s/127.0.0.1/0.0.0.0/g"
```

relaynodeN

```
cd $NODE_HOME

sed -i ${NODE_CONFIG}-config.json -e "s/127.0.0.1/0.0.0.0/g"
```

> Port forwarding and firewall config:
>
> On block producer node (or relaynodeN), you need to open ports 12798 and 9100
>
> On relaynode1, you will need to open ports 3000 for grafana.

Stop and restart your stake pool.

block producer node

```
sudo systemctl restart cardano-node
```

relaynode1

```
sudo systemctl restart cardano-node
```

16.2 Setting up Grafana Dashboards

1. On relaynode1, open http://localhost:3000 or http://<your relaynode1 ip address>:3000 in your local browser. You may need to open up port 3000 in your router and/or firewall.
2. Login with **admin** / **admin**
3. Change password
4. Click the **configuration gear** icon, then **Add data Source**
5. Select **Prometheus**
6. Set **Name** to "**Prometheus**"
7. Set **URL** to **http://localhost:9090**
8. Click **Save & Test**
9. **Download and save** this **json file.**
10. Click **Create +** icon > **Import**
11. Add dashboard by **Upload JSON file**
12. Click the **Import** button.

Credits to KAZE stake pool for this dashboard

Congratulations. You're basically done. More great operational and maintenance tips below.

17. Thank yous, Telegram and reference material

17.1 Donation Tip Jar

Did you find our guide useful? Let us know with a tip and we'll keep updating it. Bonus points if you use section 18.9's instructions.

It really energizes us to keep creating the best crypto guides.

Use cointr.ee to find our donation addresses.

Thank you for supporting Cardano and us! Please use the below cointr.ee link.

A Crypto Donation Platform

Cointr.ee is a platform for people who earn their money through community support.

cointr.ee

17.2 Thank yous

Thanks to all 17000+ of you, the Cardano hodlers, buidlers, stakers, and pool operators for making the better future a reality.

17.3 Telegram and Discord Chat Channel

Hang out and chat with our telegram stake pool community at https://t.me/coincashew

Discord community located @ https://discord.gg/w8Bx8W2HPW

17.4 Contributors, Donators and Friendly Stake Pools of CoinCashew

Contributors to the Guide

- Antonie of CNT for being awesomely helpful with Youtube content and in telegram.
- Special thanks to [KAZE] for the pull requests, sharing a new and improved grafana dashboard, and automatic script contributions.
- The Legend of ~~A~~da [TLOA] for translating this guide to Spanish.
- X-StakePool [BTBF] for translating this guide to Japanese.
- Chris of OMEGA | CODEX for security improvements.
- Raymond of GROW for topologyUpdater improvements and being awesome.

Tip Jar Donators

- BEBOP | BCOOL

- DEW
- GROW
- Leonardo
- PANJ
- SQUID
- TREE
- SAvvY
- YOU?! Hit us up.

CoinCashew's Preferred Stake Pools

- CNT
- OMEGA | CODEX
- BTBF
- TLOA
- KAZE
- BEBOP | BCOOL
- DEW
- GROW
- PANJ
- SQUID
- TREE
- SAvvY

17.5 Reference Material

For more information and official documentation, please refer to the following links:

Getting Started with Stake Pool Operations — Cardano Documentation 1.0.0 documentation
docs.cardano.org

Cardano Developers

Documentation for Cardano Developers.

testnets.cardano.org

input-output-hk/cardano-tutorials

ARCHIVED-This content in this repository is now located at https://docs.cardano.org/projects/cardano-node/ - input-output-

github.com

cardano-community/guild-operators

Artifacts and scripts created by Guild operators. Contribute to cardano-community/guild-operators development by creating an

github.com

gitmachtl/scripts

Useful scripts. Contribute to gitmachtl/scripts development by creating an account on GitHub.

github.com

CNTools by Guild Operators

Many pool operators have asked about how to deploy a stake pool with CNTools. The official guide can be found here.

18. Operational and Maintenance Tips

18.1 Rotate pool's KES keys - Updating the operational cert with a new KES Period

> You are required to regenerate the hot keys and issue a new operational certificate, a process called rotating the KES keys, when the hot keys expire.
>
> **Mainnet**: KES keys will be valid for 120 rotations or 90 days

When it's time to issue a new operational certificate, run the following to find the starting KES period.

block producer node

```
cd $NODE_HOME
slotNo=$(cardano-cli query tip --mainnet | jq -r '.slotNo')
slotsPerKESPeriod=$(cat $NODE_HOME/${NODE_CONFIG}-shelley-genesis.jsor
kesPeriod=$((${slotNo} / ${slotsPerKESPeriod}))
startKesPeriod=${kesPeriod}
echo startKesPeriod: ${startKesPeriod}
```

Make a new KES key pair.

block producer node

```
cd $NODE_HOME
cardano-cli node key-gen-KES \
    --verification-key-file kes.vkey \
    --signing-key-file kes.skey
```

Copy **kes.vkey** to your **cold environment.**

Create the new `node.cert` file with the following command. Update `<startKesPeriod>` with the value from above.

air-gapped offline machine

```
cd $NODE_HOME
chmod u+rwx $HOME/cold-keys
cardano-cli node issue-op-cert \
    --kes-verification-key-file kes.vkey \
    --cold-signing-key-file $HOME/cold-keys/node.skey \
    --operational-certificate-issue-counter $HOME/cold-keys/node.count
    --kes-period <startKesPeriod> \
    --out-file node.cert
chmod a-rwx $HOME/cold-keys
```

⚠ Copy **node.cert** back to your block producer node.

Stop and restart your block producer node to complete this procedure.

block producer node

```
sudo systemctl restart cardano-node
```

manual

```
cd $NODE_HOME
killall -s 2 cardano-node
./startBlockProducingNode.sh
```

Tip: With your hot keys created, you can remove access to the cold keys for improved security. This protects against accidental deletion, editing, or access.

To lock,

```
chmod a-rwx $HOME/cold-keys
```

To unlock,

```
chmod u+rwx $HOME/cold-keys
```

18.2 Resetting the installation

Want a clean start? Re-using existing server? Forked blockchain?

Delete git repo, and then rename your previous `$NODE_HOME` and `cold-keys` directory (or optionally, remove). Now you can start this guide from the beginning again.

```
rm -rf $HOME/git/cardano-node/ $HOME/git/libsodium/
mv $NODE_HOME $(basename $NODE_HOME)_backup_$(date -I)
mv $HOME/cold-keys $HOME/cold-keys_backup_$(date -I)
```

18.3 Resetting the databases

Corrupted or stuck blockchain? Delete all db folders.

```
cd $NODE_HOME
rm -rf db
```

18.4 Changing the pledge, fee, margin, etc.

> ⚠ **Important Reminder** Any changes made in this section take effect in two epochs. A common mistake is lowering the pledge amount and removing funds too soon. This results in zero rewards as the current live pledge amount is no longer met.

> ⓘ Need to change your pledge, fee, margin, pool IP/port, or metadata? Simply resubmit your stake pool registration certificate.

Find the minimum pool cost.

block producer node

```
minPoolCost=$(cat $NODE_HOME/params.json | jq -r .minPoolCost)
echo minPoolCost: ${minPoolCost}
```

> ⓘ minPoolCost is 340000000 lovelace or 340 ADA. Therefore, your `--pool-cost` must be at a minimum this amount.

If you're changing your poolMetaData.json, remember to calculate the hash of your metadata file and re-upload the updated poolMetaData.json file. Refer to section 9 for information. If you're verifying your stake pool ID, the hash is already provided to you by pooltool.

block producer node

```
cardano-cli stake-pool metadata-hash --pool-metadata-file poolMetaData.
```

Update the below registration-certificate transaction with your desired settings.

If you have **multiple relay nodes,** **refer to section 12** and change your parameters appropriately.

> **metadata-url** must be no longer than 64 characters.

air-gapped offline machine

```
cardano-cli stake-pool registration-certificate \
    --cold-verification-key-file $HOME/cold-keys/node.vkey \
    --vrf-verification-key-file vrf.vkey \
    --pool-pledge 1000000000 \
    --pool-cost 345000000 \
    --pool-margin 0.20 \
    --pool-reward-account-verification-key-file stake.vkey \
    --pool-owner-stake-verification-key-file stake.vkey \
    --mainnet \
    --single-host-pool-relay <dns based relay, example ~ relaynode1.my
    --pool-relay-port 6000 \
    --metadata-url <url where you uploaded poolMetaData.json> \
    --metadata-hash $(cat poolMetaDataHash.txt) \
    --out-file pool.cert
```

> Here we are pledging 1000 ADA with a fixed pool cost of 345 ADA and a pool margin of 20%.

Copy **pool.cert** to your **hot environment.**

Pledge stake to your stake pool.

air-gapped offline machine

```
cardano-cli stake-address delegation-certificate \
    --stake-verification-key-file stake.vkey \
    --cold-verification-key-file $HOME/cold-keys/node.vkey \
    --out-file deleg.cert
```

Copy **deleg.cert** to your **hot environment.**

You need to find the **tip** of the blockchain to set the **invalid-hereafter** parameter properly.

block producer node

```
currentSlot=$(cardano-cli query tip --mainnet | jq -r '.slotNo')
echo Current Slot: $currentSlot
```

Find your balance and **UTXOs**.

block producer node

```
cardano-cli query utxo \
    --address $(cat payment.addr) \
    --allegra-era \
    --mainnet > fullUtxo.out

tail -n +3 fullUtxo.out | sort -k3 -nr > balance.out

cat balance.out

tx_in=""
total_balance=0
while read -r utxo; do
    in_addr=$(awk '{ print $1 }' <<< "${utxo}")
    idx=$(awk '{ print $2 }' <<< "${utxo}")
    utxo_balance=$(awk '{ print $3 }' <<< "${utxo}")
    total_balance=$((${total_balance}+${utxo_balance}))
    echo TxHash: ${in_addr}#${idx}
    echo ADA: ${utxo_balance}
    tx_in="${tx_in} --tx-in ${in_addr}#${idx}"
done < balance.out
txcnt=$(cat balance.out | wc -l)
echo Total ADA balance: ${total_balance}
echo Number of UTXOs: ${txcnt}
```

Run the build-raw transaction command.

> The **invalid-hereafter** value must be greater than the current tip. In this example, we use current slot + 10000.

block producer node

```
cardano-cli transaction build-raw \
    ${tx_in} \
    --tx-out $(cat payment.addr)+${total_balance} \
    --invalid-hereafter $(( ${currentSlot} + 10000)) \
    --fee 0 \
    --certificate-file pool.cert \
    --certificate-file deleg.cert \
    --allegra-era \
    --out-file tx.tmp
```

Calculate the minimum fee:

block producer node

```
fee=$(cardano-cli transaction calculate-min-fee \
    --tx-body-file tx.tmp \
    --tx-in-count ${txcnt} \
    --tx-out-count 1 \
    --mainnet \
    --witness-count 3 \
    --byron-witness-count 0 \
    --protocol-params-file params.json | awk '{ print $1 }')
echo fee: $fee
```

Calculate your change output.

block producer node

```
txOut=$((${total_balance}-${fee}))
echo txOut: ${txOut}
```

Build the transaction.

block producer node

```
cardano-cli transaction build-raw \
    ${tx_in} \
    --tx-out $(cat payment.addr)+${txOut} \
    --invalid-hereafter $(( ${currentSlot} + 10000)) \
    --fee ${fee} \
    --certificate-file pool.cert \
    --certificate-file deleg.cert \
    --allegra-era \
    --out-file tx.raw
```

Copy **tx.raw** to your **cold environment.**

Sign the transaction.

air-gapped offline machine

```
cardano-cli transaction sign \
    --tx-body-file tx.raw \
    --signing-key-file payment.skey \
    --signing-key-file $HOME/cold-keys/node.skey \
    --signing-key-file stake.skey \
    --mainnet \
    --out-file tx.signed
```

Copy **tx.signed** to your **hot environment.**

Send the transaction.

block producer node

```
cardano-cli transaction submit \
    --tx-file tx.signed \
    --mainnet
```

Changes take effect next epoch. After the next epoch transition, verify that your pool settings are correct.

block producer node

```
cardano-cli query ledger-state --mainnet --allegra-era --out-file ledg
jq -r '.esLState._delegationState._pstate._pParams."'"$(cat stakepooli
```

18.5 Transferring files over SSH

Common use cases can include

- Downloading backups of stake/payment keys
- Uploading a new operational certificate to the block producer from an offline node

To download files from a node to your local PC

```
ssh <USERNAME>@<IP ADDRESS> -p <SSH-PORT>
rsync -avzhe “ssh -p <SSH-PORT>” <USERNAME>@<IP ADDRESS>:<PATH TO NODE DEST
```

> Example:
>
> ```
> ssh myusername@6.1.2.3 -p 12345
> ```

```
rsync -avzhe "ssh -p 12345" myusername@6.1.2.3:/home/myusername/cardano-my-node/stake.vkey ./stake.vkey
```

To upload files from your local PC to a node

```
ssh <USERNAME>@<IP ADDRESS> -p <SSH-PORT>
rsync -avzhe “ssh -p <SSH-PORT>” <PATH TO LOCAL PC DESTINATION> <USERNAME>@
```

Example:

```
ssh myusername@6.1.2.3 -p 12345

rsync -avzhe "ssh -p 12345" ./node.cert myusername@6.1.2.3:/home/myusername/cardano-my-node/node.cert
```

18.7 Verify your stake pool ticker with ITN key

In order to defend against spoofing and hijacking of reputable stake pools, a owner can verify their ticker by proving ownership of an ITN stake pool.

> ⓘ Incentivized Testnet phase of Cardano’s Shelley era ran from late November 2019 to late June 2020. If you participated, you can verify your ticker.

Make sure the ITN's `jcli` binaries are present in `$NODE_HOME` . Use `jcli` to sign your stake pool id with your `itn_owner.skey`

air-gapped offline machine

```
./jcli key sign --secret-key itn_owner.skey stakepoolid.txt --output st
```

Visit pooltool.io and enter your owner public key and pool id witness data in the metadata section.

Find your pool id witness with the following command.

air-gapped offline machine

```
cat stakepoolid.sig
```

Find your owner public key in the file you generated on ITN. This data might be stored in a file ending in `.pub`

Finally, follow instructions to update your pool registration data with the pooltool generated **`metadata-url`** and **`metadata-hash`** . Notice the metadata has an "extended" field which proves your ticker ownership since ITN.

18.8 Updating your node's configuration files

Keep your config files fresh by downloading the latest .json files.

```
NODE_BUILD_NUM=$(curl https://hydra.iohk.io/job/Cardano/iohk-nix/cardano-de
cd $NODE_HOME
wget -N https://hydra.iohk.io/build/${NODE_BUILD_NUM}/download/1/${NODE_CON
wget -N https://hydra.iohk.io/build/${NODE_BUILD_NUM}/download/1/${NODE_CON
wget -N https://hydra.iohk.io/build/${NODE_BUILD_NUM}/download/1/${NODE_CON
sed -i ${NODE_CONFIG}-config.json \
    -e "s/TraceBlockFetchDecisions\": false/TraceBlockFetchDecisions\": tru
```

18.9 Send a simple transaction example

Let's walk through an example to send **10 ADA** to **CoinCashew's tip address**

First, find the **tip** of the blockchain to set the **invalid-hereafter** parameter properly.

block producer node

```
currentSlot=$(cardano-cli query tip --mainnet | jq -r '.slotNo')
echo Current Slot: $currentSlot
```

Set the amount to send in lovelaces. Remember **1 ADA** = **1,000,000 lovelaces.**

block producer node

```
amountToSend=10000000
echo amountToSend: $amountToSend
```

Set the destination address which is where you're sending funds to.

block producer node

```
destinationAddress=addr1qxhazv2dp8yvqwyxxlt7n7ufwhw582uqtcn9llqak736pt
echo destinationAddress: $destinationAddress
```

Find your balance and **UTXOs**.

block producer node

```
cardano-cli query utxo \
    --address $(cat payment.addr) \
    --allegra-era \
    --mainnet > fullUtxo.out

tail -n +3 fullUtxo.out | sort -k3 -nr > balance.out

cat balance.out

tx_in=""
total_balance=0
while read -r utxo; do
    in_addr=$(awk '{ print $1 }' <<< "${utxo}")
    idx=$(awk '{ print $2 }' <<< "${utxo}")
    utxo_balance=$(awk '{ print $3 }' <<< "${utxo}")
    total_balance=$((${total_balance}+${utxo_balance}))
    echo TxHash: ${in_addr}#${idx}
    echo ADA: ${utxo_balance}
    tx_in="${tx_in} --tx-in ${in_addr}#${idx}"
done < balance.out
txcnt=$(cat balance.out | wc -l)
echo Total ADA balance: ${total_balance}
echo Number of UTXOs: ${txcnt}
```

Run the build-raw transaction command.

block producer node

```
cardano-cli transaction build-raw \
    ${tx_in} \
    --tx-out $(cat payment.addr)+0 \
    --tx-out ${destinationAddress}+0 \
    --invalid-hereafter $(( ${currentSlot} + 10000)) \
    --fee 0 \
    --allegra-era \
    --out-file tx.tmp
```

Calculate the current minimum fee:

block producer node

```
fee=$(cardano-cli transaction calculate-min-fee \
    --tx-body-file tx.tmp \
    --tx-in-count ${txcnt} \
    --tx-out-count 2 \
    --mainnet \
    --witness-count 1 \
    --byron-witness-count 0 \
    --protocol-params-file params.json | awk '{ print $1 }')
echo fee: $fee
```

Calculate your change output.

block producer node

```
txOut=$((${total_balance}-${fee}-${amountToSend}))
echo Change Output: ${txOut}
```

Build your transaction.

block producer node

```
cardano-cli transaction build-raw \
    ${tx_in} \
    --tx-out $(cat payment.addr)+${txOut} \
    --tx-out ${destinationAddress}+${amountToSend} \
    --invalid-hereafter $(( ${currentSlot} + 10000)) \
    --fee ${fee} \
    --allegra-era \
    --out-file tx.raw
```

Copy **tx.raw** to your **cold environment.**

Sign the transaction with both the payment and stake secret keys.

air-gapped offline machine

```
cardano-cli transaction sign \
    --tx-body-file tx.raw \
    --signing-key-file payment.skey \
    --mainnet \
    --out-file tx.signed
```

Copy **tx.signed** to your **hot environment.**

Send the signed transaction.

block producer node

```
cardano-cli transaction submit \
    --tx-file tx.signed \
    --mainnet
```

Check if the funds arrived.

block producer node

```
cardano-cli query utxo \
    --address ${destinationAddress} \
    --allegra-era \
    --mainnet
```

You should see output similar to this showing the funds you sent.

```
                           TxHash                                 TxIx
--------------------------------------------------------------------------------
100322a39d02c2ead....                                              0
```

18.10 Harden your node's security

Do not skimp on this critical step to protect your pool and reputation.

→ How to Harden Ubuntu Server /coins/overview-ada/guide-how-to-build-a-haskell-stakepool-node/how-to-harden-ubuntu-server

18.11 Claim your rewards

Let's walk through an example to claim your stake pools rewards.

> ⓘ Rewards are accumulated in the `stake.addr` address.

First, find the **tip** of the blockchain to set the **invalid-hereafter** parameter properly.

block producer node

```
currentSlot=$(cardano-cli query tip --mainnet | jq -r '.slotNo')
echo Current Slot: $currentSlot
```

Set the amount to send in lovelaces. Remember **1 ADA** = **1,000,000 lovelaces.**

block producer node

```
rewardBalance=$(cardano-cli query stake-address-info \
    --mainnet \
    --allegra-era \
    --address $(cat stake.addr) | jq -r ".[0].rewardAccountBalance")
echo rewardBalance: $rewardBalance
```

Set the destination address which is where you're moving your reward to. This address must have a positive balance to pay for transaction fees.

block producer node

```
destinationAddress=$(cat payment.addr)
echo destinationAddress: $destinationAddress
```

Find your payment.addr balance, utxos and build the withdrawal string.

block producer node

```
cardano-cli query utxo \
    --address $(cat payment.addr) \
    --allegra-era \
    --mainnet > fullUtxo.out

tail -n +3 fullUtxo.out | sort -k3 -nr > balance.out

cat balance.out

tx_in=""
total_balance=0
while read -r utxo; do
    in_addr=$(awk '{ print $1 }' <<< "${utxo}")
    idx=$(awk '{ print $2 }' <<< "${utxo}")
    utxo_balance=$(awk '{ print $3 }' <<< "${utxo}")
    total_balance=$((${total_balance}+${utxo_balance}))
    echo TxHash: ${in_addr}#${idx}
    echo ADA: ${utxo_balance}
    tx_in="${tx_in} --tx-in ${in_addr}#${idx}"
done < balance.out
txcnt=$(cat balance.out | wc -l)
echo Total ADA balance: ${total_balance}
echo Number of UTXOs: ${txcnt}

withdrawalString="$(cat stake.addr)+${rewardBalance}"
```

Run the build-raw transaction command.

block producer node

```
cardano-cli transaction build-raw \
    ${tx_in} \
    --tx-out $(cat payment.addr)+0 \
    --invalid-hereafter $(( ${currentSlot} + 10000)) \
    --fee 0 \
    --withdrawal ${withdrawalString} \
    --allegra-era \
    --out-file tx.tmp
```

Calculate the current minimum fee:

block producer node

```
fee=$(cardano-cli transaction calculate-min-fee \
    --tx-body-file tx.tmp \
    --tx-in-count ${txcnt} \
    --tx-out-count 1 \
    --mainnet \
    --witness-count 2 \
    --byron-witness-count 0 \
    --protocol-params-file params.json | awk '{ print $1 }')
echo fee: $fee
```

Calculate your change output.

block producer node

```
txOut=$((${total_balance}-${fee}+${rewardBalance}))
echo Change Output: ${txOut}
```

Build your transaction.

block producer node

```
cardano-cli transaction build-raw \
    ${tx_in} \
    --tx-out $(cat payment.addr)+${txOut} \
    --invalid-hereafter $(( ${currentSlot} + 10000)) \
    --fee ${fee} \
    --withdrawal ${withdrawalString} \
    --allegra-era \
    --out-file tx.raw
```

Copy **tx.raw** to your **cold environment.**

Sign the transaction with both the payment and stake secret keys.

air-gapped offline machine

```
cardano-cli transaction sign \
    --tx-body-file tx.raw \
    --signing-key-file payment.skey \
    --signing-key-file stake.skey \
    --mainnet \
    --out-file tx.signed
```

Copy **tx.signed** to your **hot environment.**

Send the signed transaction.

block producer node

```
cardano-cli transaction submit \
    --tx-file tx.signed \
    --mainnet
```

Check if the funds arrived.

block producer node

```
cardano-cli query utxo \
    --address ${destinationAddress} \
    --allegra-era \
    --mainnet
```

You should see output similar to this showing your updated Lovelace balance with rewards.

```
                           TxHash                                 TxIx
--------------------------------------------------------------------------------
100322a39d02c2ead....
```

18.12 Slot Leader Schedule - Find out when your pool will mint blocks

> **Hot tip**: You can calculate your slot leader schedule, which tells you when it's your stake pools turn to mint a block. This can help you know what time is best to schedule maintenance on your stake pool. It can also help verify your pool is minting blocks correctly when it is your pool's turn. Credits for inventing this process goes to the hard work by Andrew Westberg @amw7 (developer of JorManager and operator of BCSH family of stake pools).

Check if you have python installed.

block producer node

```
python3 --version
```

Otherwise, install python3.

block producer node

```
sudo apt-get update
sudo apt-get install -y software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt-get update
sudo apt-get install -y python3.9
```

Check if you have pip installed.

block producer node

```
pip3 --version
```

Install pip3 if needed.

block producer node

```
sudo apt-get install -y python3-pip
```

Install pytz which handles timezones.

```
pip3 install pytz
```

Verify python and pip are setup correctly before continuing.

block producer node

```
python3 --version
pip3 --version
```

Clone the leaderLog scripts from papacarp/pooltool.io git repo.

> Official documentation for this LeaderLogs tool can be read here.

block producer node

```
cd $HOME/git
git clone https://github.com/papacarp/pooltool.io
cd pooltool.io/leaderLogs
```

Query the ledger state.

block producer node

```
cardano-cli query ledger-state --mainnet --allegra-era --out-file ledge
```

Calculate your pool's sigma. Sigma represents your pool's share of the active stake.

block producer node

```
sigmaValue=$(python3 getSigma.py --pool-id $(cat ${NODE_HOME}/stakepoc
echo Sigma: ${sigmaValue}
```

A sigma value should look like `0.000029302885338621295`

Calculate your slot leader schedule.

block producer node

```
python3 leaderLogs.py --pool-id $(cat ${NODE_HOME}/stakepoolid.txt) --s
```

Set the timezone name to format the schedule's times properly. Use the --tz option. [Default: America/Los_Angeles]') Refer to the official documentation for more info.

If your pool is scheduled to mint blocks, you should hopefully see output similar to this. Listed by date and time, this is your slot leader schedule or in other words, when your pool is eligible to mint a block.

Your slot leader log should remain confidential. If you share this information publicly, an attacker could use this information to attack your stake pool.

```
Checking leadership log for Epoch 222 [ d Param: 0.6 ]
2020-10-01 00:11:10 ==> Leader for slot 121212, Cumulative epoch blocks: 1
2020-10-01 00:12:22 ==> Leader for slot 131313, Cumulative epoch blocks: 2
2020-10-01 00:19:55 ==> Leader for slot 161212, Cumulative epoch blocks: 3
```

19. Retire your stake pool

Calculate the current epoch.

block producer node

```
startTimeGenesis=$(cat $NODE_HOME/${NODE_CONFIG}-shelley-genesis.json
startTimeSec=$(date --date=${startTimeGenesis} +%s)
currentTimeSec=$(date -u +%s)
epochLength=$(cat $NODE_HOME/${NODE_CONFIG}-shelley-genesis.json | jq
epoch=$(( (${currentTimeSec}-${startTimeSec}) / ${epochLength} ))
echo current epoch: ${epoch}
```

Find the earliest and latest retirement epoch that your pool can retire.

block producer node

```
eMax=$(cat $NODE_HOME/params.json | jq -r '.eMax')
echo eMax: ${eMax}

minRetirementEpoch=$(( ${epoch} + 1 ))
maxRetirementEpoch=$(( ${epoch} + ${eMax} ))

echo earliest epoch for retirement is: ${minRetirementEpoch}
echo latest epoch for retirement is: ${maxRetirementEpoch}
```

Example: if we are in epoch 39 and eMax is 18,

- the earliest epoch for retirement is 40 (current epoch + 1).
- the latest epoch for retirement is 57 (eMax + current epoch).

Let's pretend we wish to retire as soon as possible in epoch 40.

Create the deregistration certificate and save it as `pool.dereg`. Update the epoch to your desired retirement epoch, usually the earliest epoch or asap.

air-gapped offline machine

```
cardano-cli stake-pool deregistration-certificate \
--cold-verification-key-file $HOME/cold-keys/node.vkey \
--epoch <retirementEpoch> \
--out-file pool.dereg
```

Copy **pool.dereg** to your **hot environment.**

Find your balance and **UTXOs**.

block producer node

```
cardano-cli query utxo \
    --address $(cat payment.addr) \
    --allegra-era \
    --mainnet > fullUtxo.out

tail -n +3 fullUtxo.out | sort -k3 -nr > balance.out

cat balance.out

tx_in=""
total_balance=0
while read -r utxo; do
    in_addr=$(awk '{ print $1 }' <<< "${utxo}")
    idx=$(awk '{ print $2 }' <<< "${utxo}")
    utxo_balance=$(awk '{ print $3 }' <<< "${utxo}")
    total_balance=$((${total_balance}+${utxo_balance}))
    echo TxHash: ${in_addr}#${idx}
    echo ADA: ${utxo_balance}
    tx_in="${tx_in} --tx-in ${in_addr}#${idx}"
done < balance.out
txcnt=$(cat balance.out | wc -l)
echo Total ADA balance: ${total_balance}
echo Number of UTXOs: ${txcnt}
```

Run the build-raw transaction command.

block producer node

```
cardano-cli transaction build-raw \
    ${tx_in} \
    --tx-out $(cat payment.addr)+${total_balance} \
    --invalid-hereafter $(( ${slotNo} + 10000)) \
    --fee 0 \
    --certificate-file pool.dereg \
    --allegra-era \
    --out-file tx.tmp
```

Calculate the minimum fee:

block producer node

```
fee=$(cardano-cli transaction calculate-min-fee \
    --tx-body-file tx.tmp \
    --tx-in-count ${txcnt} \
    --tx-out-count 1 \
    --mainnet \
    --witness-count 2 \
    --byron-witness-count 0 \
    --protocol-params-file params.json | awk '{ print $1 }')
echo fee: $fee
```

Calculate your change output.

block producer node

```
txOut=$((${total_balance}-${fee}))
echo txOut: ${txOut}
```

Build the transaction.

block producer node

```
cardano-cli transaction build-raw \
    ${tx_in} \
    --tx-out $(cat payment.addr)+${txOut} \
    --invalid-hereafter $(( ${slotNo} + 10000)) \
    --fee ${fee} \
    --certificate-file pool.dereg \
    --allegra-era \
    --out-file tx.raw
```

Copy **tx.raw** to your **cold environment.**

Sign the transaction.

air-gapped offline machine

```
cardano-cli transaction sign \
    --tx-body-file tx.raw \
    --signing-key-file payment.skey \
    --signing-key-file $HOME/cold-keys/node.skey \
    --mainnet \
    --out-file tx.signed
```

Copy **tx.signed** to your **hot environment.**

Send the transaction.

block producer node

```
cardano-cli transaction submit \
    --tx-file tx.signed \
    --mainnet
```

Pool will retire at the end of your specified epoch. In this example, retirement occurs at the end of epoch 40.

If you have a change of heart, you can create and submit a new registration certificate before the end of epoch 40, which will then overrule the deregistration certificate.

After the retirement epoch, you can verify that the pool was successfully retired with the following query which should return an empty result.

block producer node

```
cardano-cli query ledger-state --mainnet --allegra-era --out-file ledg
jq -r '.esLState._delegationState._pstate._pParams."'"$(cat stakepooli
```

20. Onwards and upwards...

Did you find our guide useful? Let us know with a tip and we'll keep updating it.

It really energizes us to keep creating the best crypto guides. Use cointr.ee to find our donation addresses and share your message.