undefined

Table of contents:

- Network Maintainers
  - Node
  - Collator
  - Nominator
  - Validator
  - Governance
- Polkadot Parameters
    - Periods of common actions and attributes
    - Governance
    - Staking, Validating, and Nominating
    - Treasury
    - Precision
- Set up a Full Node
    - Types of Nodes
    - Fast Install Instructions (Mac)
    - Fast Install Instructions (Windows)
    - Fast Install Instructions (Linux)
  - Get Substrate
  - Clone and Build
  - Run
  - Running an Archive Node
  - Using Docker
- Networks
  - Polkadot Networks
    - Polkadot Mainnet
    - Kusama Canary Network
    - Westend Test Network
    - Differences
  - Substrate Networks
    - Flaming Fir
  - Telemetry Dashboard
- Set up Secure WebSocket for Remote Connections
  - Set up a node
  - Set up a certificate
    - Domain and Certbot
    - Self-signed
  - Set up Nginx server
  - Importing the Certificate
  - Connecting to the node
- Errors and How to Resolve Them
  - PolkadotJS Apps Explorer
  - Polkascan and Subscan
  - Common Errors
  - Error Table
- How to Nominate on Polkadot
  - Setting up Stash and Controller keys
  - Using Polkadot-JS UI
    - Step 1: Bond your tokens
    - Step 2: Nominate a validator
    - Step 3: Stop nominating
  - Using Command-Line Interface (CLI)
    - Step 1: Install @polkadot/api-cli
    - Step 2. Bond your DOT
    - Step 3. Nominate a validator
- Unbonding and Rebonding
    - Step 1: Stop Nominating
    - Step 2: Unbonding an amount
  - Rebonding before the end of the unbonding period
- Run a Validator (Polkadot)
  - Preliminaries
    - How many DOT do I need?

    - Initial Set-up
        - Requirements
        - Install Rust
        - Install & Configure Network Time Protocol (NTP) Client
        - Building and Installing the polkadot Binary
        - Synchronize Chain Data
    - Bond DOT
    - Set Session Keys
        - Generating the Session Keys
        - Submitting the setKeys Transaction
    - Validate
    - Thousand Validators Programme
    - FAQ
        - Why am I unable to synchronize the chain with 0 peers?
        - How do I clear all my chain data?
    - VPS List
    - Using Docker
- Validator Payout Overview
    - Era Points
    - Payout Scheme
    - Running Multiple Validators
    - Slashing
    - Nominators and Validator Payments
- Using systemd for the Validator Node
- Secure Validator
    - High Availability
    - Key Management
        - Signing Outside the Client
    - Monitoring Tools
    - Linux Best Practices
    - Conclusions
        - Validators
        - Monitoring
    - Resources
- How to use Polkadot Secure Validator Setup
    - Dependencies
        - NodeJS
        - Yarn
        - Terraform
        - Ansible
    - Deployment
        - Step One: Clone the repository
        - Step Two: Generate the SSH keys
        - Configuration
- Set Up a Sentry Node - Public Node
    - DEPRECATED
- How to Upgrade Your Validator
    - Key Components
        - Session Keys
        - Database
    - Steps
        - Session N
        - Session N+1
- Monitor your node
    - Preparation
    - Installing and Configuring Prometheus
    - Starting Prometheus
    - Installing Grafana
    - Installing and Configuring Alertmanager (Optional)
        - Gmail Setup
        - AlertManager Configuration
        - AlertManager Integration
- How to Chill
    - Chilling as a Nominator

  - Chilling as a Validator
    - Involuntary Chills
- How to Stop Validating
  - Chill Validator
  - Purge validator session keys
  - Unbond your tokens
- Participate in Democracy
  - Important Parameters
  - Proposing an Action
  - Submitting a Preimage
  - Seconding a Proposal
  - Voting on a Proposal
  - Unlocking Locked Tokens
  - Delegate a Vote
  - Undelegate a Vote
  - Voting with a Governance Proxy
  - Interpreting On-Chain Voting Data
- Join the Council
  - Submit Candidacy
  - Voting on Candidates
  - Winning
- Voting for Councillors
  - Voting for Councillors
  - Removing your Vote

Network Maintainers

Welcome to the network maintainers section of the Polkadot wiki. Here you will find information and guides to set up a node and run the network.

Node

- Networks Guide - A list of the available Polkadot networks that you can connect to with a node.
- Set up a Full Node - Get up and started by syncing a full node for the Kusama network. The steps in the guide will broadly apply also to any Substrate-based network (like Polkadot).
- Set up WSS using Nginx - Set up a Secure WebSockets server for your node's WebSockets connection.

Collator

- Learn about Collators - High level overview of collators and related links.

Nominator

- Learn about Nominators - High level overview of nominators and related links.
- Nomination Guide (Polkadot) - Walkthrough on how to nominate on the Polkadot network.
- Nomination Guide (Kusama) - Walkthrough on how to nominate on the Kusama canary network.
- How to stop being a Nominator - Guide on how to stop nominating.

Validator

- Learn about Validators - High level overview of validator and related links.
- Validator Payouts - Overview on how validator rewards are calculated and paid.
- Validation Guide (Polkadot) - Walkthrough on how to validate on the Polkadot network.
- Validation Guide (Kusama) - Walkthrough on how to validate on the Kusama canary network.
- Using systemmd for the Validator Node - Configuring systemmd with the Validator node.
- Secure Validator - Best tips and practices for validating.
- How to use Polkadot Secure Validator - Walkthrough on how to set up a validator securely.
- How to upgrade a Validator Node - Guide on upgrading your validator node.
- How to Chill - Walkthrough on how to chill as a validator.

Governance

- How to pariticipate in Governance - Walkthrough on how to participate in governance.
- How to join the Council - Step by step guide for running for the Council.
- How to vote for a Councillor - Step by step guide for voting for your favorite councillors.

Polkadot Parameters

Many of these parameter values can be updated via on-chain governance. If you require absolute certainty of these parameter values, it is recommended you directly check the constants by looking at the chain state and/or storage.

Periods of common actions and attributes

NOTE: Kusama generally runs 4x as fast as Polkadot, except in the time slot duration itself. See Polkadot Parameters for more details on how Kusama's parameters differ from Polkadot's.

- Slot: 6 seconds *(generally one block per slot, although see note below)
- Epoch: 4 hours (2_400 slots x 6 seconds)
- Session: 4 hours (Session and Epoch lengths are the same)
- Era: 24 hours (6 sessions per Era, 2_400 slots x 6 epochs x 6 seconds)

Polkadot	Time	Slots*
Slot	6 seconds	1
Epoch	4 hours	2_400
Session	4 hours	2_400
Era	24 hours	14_400

**A maximum of one block per slot can be in a canonical chain. Occasionally, a slot will be without a block in the chain. Thus, the times given are estimates. See Consensus for more details.*

Governance

Democracy	Time	Slots	Description
Voting period	28 days	403_200	How long the public can vote on a referendum.
Launch period	28 days	403_200	How long the public can select which proposal to hold a referendum on, i.e., every week, the highest-weighted proposal will be selected to have a referendum.
Enactment period	28 days	403_200	Time it takes for a successful referendum to be implemented on the network.

Council	Time	Slots	Description
Term duration	7 days	100_800	The length of a council member's term until the next election round.
Voting period	7 days	100_800	The council's voting period for motions.

The Polkadot Council consists of up to 13 members and up to 20 runners up.

Technical committee	Time	Slots	Description
Cool-off period	7 days	100_800	The time a veto from the technical committee lasts before the proposal can be submitted again.

Technical committee	Time	Slots	Description
Emergency voting period	3 hours	1_800	The voting period after the technical committee expedites voting.

Staking, Validating, and Nominating

Kusama	Time	Slots	Description
Term duration	1 Day	14_400	The time for which a validator is in the set after being elected. Note, this duration can be shortened in the case that a validator misbehaves.
Nomination period	1 Day	14_400	How often a new validator set is elected according to Phragmén's method.
Bonding duration	28 days	403_200	How long until your funds will be transferrable after unbonding. Note that the bonding duration is defined in eras, not directly by slots.
Slash defer duration	28 days	403_200	Prevents overslashing and validators "escaping" and getting their nominators slashed with no repercussions to themselves. Note that the bonding duration is defined in eras, not directly by slots.

Treasury

Treasury	Time	Slots	Description
Periods between spends	24 days	345_600	When the treasury can spend again after spending previously.

Burn percentage is currently `1.00%`.

Precision

DOT have 10 decimals of precision. In other words, 10 ** 10 (10_000_000_000 or ten billion) Plancks make up a DOT.

The denomination of DOT was changed from 12 decimals of precision at block #1*248_328 in an event known as _Denomination Day*. See Redenomination for details.

Set up a Full Node

If you're building dapps or products on a Substrate-based chain like Polkadot, Kusama or a custom Substrate implementation, you probably want the ability to run a node-as-a-back-end. After all, it's always better to rely on your own infrastructure than on a third-party-hosted one in this brave new decentralized world.

This guide will show you how to connect to Kusama network, but the same process applies to any other Substrate-based chain. First, let's clarify the term *full node*.

Types of Nodes

A blockchain's growth comes from a *genesis block*, *extrinsics*, and *events*.

When a validator seals block 1, it takes the blockchain's state at block 0. It then applies all pending changes on top of it, and emits the events that are the result of these changes. Later, the state of the chain at block 1 is used in the same way to build the state of the chain at block 2, and so on. Once two thirds of the validators agree on a specific block being valid, it is finalized.

An **archive node** keeps all the past blocks. An archive node makes it convenient to query the past state of the chain at any point in time. Finding out what an account's balance at a certain block was, or which extrinsics resulted in a certain state change are fast operations when using an archive node. However, an archive node takes up a lot of disk space - around Kusama's 1.6 millionth block this was around 15 to 20GB.

A **full node** is *pruned*, meaning it discards all information older than 256 blocks, but keeps the extrinsics for all past blocks, and the genesis block. A node that is pruned this way requires much less space than an archive node. In order to query past state through a full node, a user would have to wait for the node to rebuild the chain up until that block. A full node *can* rebuild the entire chain with no additional input from other nodes and become an archive node. One caveat is that if finality stalled for some reason and the last finalized block is more than 256 blocks behind, a pruned full node will not be able to sync to the network.

Archive nodes are used by utilities that need past information - like block explorers, council scanners, discussion platforms like Polkassembly, and others. They need to be able to look at past on-chain data. Full nodes are used by everyone else - they allow you to read the current state of the chain and to submit transactions directly to the chain without relying on a centralized infrastructure provider.

Another type of node is a **light node**. A light node has only the runtime and the current state, but does not store past extrinsics and so cannot restore the full chain from genesis. Light nodes are useful for resource restricted devices. An interesting use-case of light nodes is a Chrome extension, which is a node in its own right, running the runtime in WASM format: https://github.com/paritytech/substrate-light-ui

Fast Install Instructions (Mac)

Not recommended if you're a validator. Please see secure validator setup

- Type terminal in the ios searchbar/searchlight to open the 'terminal' application
- Install Homebrew within the terminal by running: `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"`
- Then run: `brew install openssl cmake llvm`
- Install Rust in your terminal by running: `curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh`
- Once Rust is installed, run the following command to clone and build the kusama code:

```
git clone https://github.com/paritytech/polkadot kusama
cd kusama
./scripts/init.sh
cargo build --release
```

- Run the following command to start your node: `./target/release/polkadot --name "My node's name"`
- Find your node at https://telemetry.polkadot.io/#list/Kusama

Fast Install Instructions (Windows)

This works only on Windows Pro with virtualization enabled.

Not recommended if you're a validator. Please see secure validator setup

- Install WSL: https://docs.microsoft.com/en-us/windows/wsl/install-win10
- Install Ubuntu (same webpage): https://docs.microsoft.com/en-us/windows/wsl/install-win10
- Determine the latest version of the Polkadot binary (you can see the latest releases here: https://github.com/paritytech/polkadot/releases)
- Download the correct Polkadot binary within Ubuntu by running the following command. Replace `*VERSION*` with the tag of the latest version from the last step (e.g. `v0.8.22`): `curl -sL https://github.com/paritytech/polkadot/releases/download/*VERSION*/polkadot -o polkadot`
- Run the following: `sudo chmod +x polkadot`
- Run the following: `./polkadot --name "Your Node Name Here"`
- Find your node at https://telemetry.polkadot.io/#list/Kusama

Fast Install Instructions (Linux)

Not recommended if you're a validator. Please see secure validator setup

For the most recent binary please see the release page on the polkadot repository. The URL in the code snippet below may become slightly out-of-date.

Also please note that the nature of pre-built binaries means that they may not work on your particular architecture or Linux distribution. If you see an error like `cannot execute binary file: Exec format error` it likely means the binary is not compatible with your system. You will either need to compile the source code yourself or use docker.

- Determine the latest version of the Polkadot binary (you can see the latest releases here: https://github.com/paritytech/polkadot/releases)
- Download the correct Polkadot binary within Ubuntu by running the following command. Replace `*VERSION*` with the tag of the latest version from the last step (e.g. `v0.8.22`): `curl -sL https://github.com/paritytech/polkadot/releases/download/*VERSION*/polkadot -o polkadot`
- Run the following: `sudo chmod +x polkadot`
- Run the following: `./polkadot --name "Your Node Name Here"`
- Find your node at https://telemetry.polkadot.io/#list/Kusama

Get Substrate

Follow instructions as outlined here - note that Windows users will have their work cut out for them. It's better to use a virtual machine instead.

Test if the installation was successful by running `cargo --version`.

```
λ cargo --version
cargo 1.41.0 (626f0f40e 2019-12-03)
```

Clone and Build

The paritytech/polkadot repo's master branch contains the latest Kusama code.

```
git clone https://github.com/paritytech/polkadot kusama
cd kusama
./scripts/init.sh
cargo build --release
```

Alternatively, if you wish to use a specific release, you can check out a specific tag (`v0.8.3` in the example below):

```
git clone https://github.com/paritytech/polkadot kusama
cd kusama
git checkout tags/v0.8.3
./scripts/init.sh
cargo build --release
```

Run

The built binary will be in the `target/release` folder, called `polkadot`.

```
./target/release/polkadot --name "My node's name"
```

Use the `--help` flag to find out which flags you can use when running the node. For example, if connecting to your node remotely, you'll probably want to use `--ws-external` and `--rpc-cors all`.

The syncing process will take a while depending on your bandwidth, processing power, disk speed and RAM. On a \$10 DigitalOcean droplet, the process can complete in some 36 hours.

Congratulations, you're now syncing with Kusama. Keep in mind that the process is identical when using any other Substrate chain.

Running an Archive Node

When running as a simple sync node (above), only the state of the past 256 blocks will be kept. When validating, it defaults to archive mode. To keep the full state use the `--pruning` flag:

```
./target/release/polkadot --name "My node's name" --pruning archive
```

It is possible to almost quadruple synchronization speed by using an additional flag: `--wasm-execution Compiled`. Note that this uses much more CPU and RAM, so it should be turned off after the node is in sync.

Using Docker

Finally, you can use Docker to run your node in a container. Doing this is a bit more advanced so it's best left up to those that either already have familiarity with docker, or have completed the other set-up instructions in this guide. If you would like to connect to your node's WebSockets ensure that you run you node with the `--rpc-external` and `--ws-external` commands.

```
docker run -p 9944:9944 parity/polkadot:v0.8.24 --name "calling_home_from_a_docker_container" --rpc-external --
ws-external
```

Networks

Polkadot is built on top of Substrate, a modular framework for blockchains. One feature of Substrate is to allow for connection to different networks using a single executable and configuring it with a start-up flag. Here are some of the networks associated with Polkadot or Substrate that you may want to connect to and join.

Polkadot Networks

To connect to a Polkadot network please follow the instructions for installing the Polkadot executable.

Polkadot Mainnet

Currently Polkadot is built from the tip of master and is the default option when starting a node.

To start a Polkadot node, run the Polkadot binary:

```
polkadot
```

and you will connect and start syncing to Polkadot.

Check your node is connected by viewing it on Telemetry (you can set a custom name by specifying `--name "my custom name"`)

Kusama Canary Network

Kusama is a canary network and holds real economic value.

Run the Polkadot binary and specify `kusama` as the chain:

```
polkadot --chain=kusama
```

and you will connect and start syncing to Kusama.

Check your node is connected by viewing it on Telemetry (you can set a custom name by specifying `--name "my custom name"`)

Westend Test Network

Westend is the latest test network for Polkadot. The tokens on this network are called *Westies* and they purposefully hold no economic value.

Run the Polkadot binary and specify `westend` as the chain:

```
polkadot --chain=westend
```

and you will connect and start syncing to Westend.

Check your node is connected by viewing it on Telemetry (you can set a custom name by specifying `--name "my custom name"`)

Westend Faucet

Follow the instruction here for instructions on acquiring Westies.

Differences

Runtime differences (e.g. existential and multisignature deposit sizes) between the different networks can be found by doing a `diff` between the `src/lib.rs` of the respositories. For example, to compare the Polkadot and Westend runtimes:

- `git clone https://github.com/paritytech/polkadot && cd polkadot/runtime`

- `ls` - show the available runtimes
- `diff polkadot/src/lib.rs westend/src/lib.rs`

You can also paste the runtimes (Polkadot, Westend) into a web-based diff tool like Diffchecker if you're not comfortable with the CLI.

Substrate Networks

To connect to a Substrate public network first follow the instructions for installing the Substrate executable.

Flaming Fir

Flaming Fir is the public Substrate test network. It contains some pallets that will not be included in the Polkadot runtime.

Flaming Fir is built from the tip of master and is the default option when running the Substrate executable.

Run Substrate without a flag or explicitly state `fir`:

```
substrate --chain fir
```

and you will connect and start syncing Flaming Fir.

Telemetry Dashboard

If you connect to the public networks, the default configuration for your node will connect it to the public Telemetry service.

You can verify that your node is connected by navigating to the correct network on the dashboard and finding the name of your node.

There is a built-in search function for the nodes page. Simply start typing keystrokes in the main window to make it available.

Set up Secure WebSocket for Remote Connections

You might want to host a node on one server and then connect to it from a UI hosted on another, e.g. Polkadot-JS UI. This will not be possible unless you set up a secure proxy for websocket connections. Let's see how we can set up WSS on a remote Substrate node.

*Note: this should **only** be done for sync nodes used as back-end for some dapps or projects. Never open websockets to your validator node - there's no reason to do that and it can only lead to security gaffes.*

In this guide we'll be using Ubuntu 18.04 hosted on a \$10 DigitalOcean droplet. We'll assume you're using a similar OS, and that you have nginx installed (if not, run `sudo apt-get install nginx`).

Set up a node

Whether it's a generic Substrate node, a Kusama node, or your own private blockchain, they all default to the same websocket connection: port 9944 on localhost. For this example, we'll set up a Kusama sync node (non-validator).

Create a new server on your provider of choice or locally at home (preferred). We'll assume you're using Ubuntu 18.04. Then install Substrate and build the node.

```
curl https://getsubstrate.io -sSf | bash
git clone https://github.com/paritytech/polkadot kusama
cd kusama
./scripts/init.sh
cargo build --release
./target/release/polkadot --name "DigitalOcean 10 USD droplet ftw" --rpc-cors all
```

This will start the syncing process with Kusama's mainnet.

Note: the `--rpc-cors` mode needs to be set to all so that all external connections are allowed.

Set up a certificate

To get WSS (secure websocket), you need an SSL certificate. There are two possible approaches.

Domain and Certbot

The first approach is getting a dedicated domain, redirecting its nameservers to your IP address, setting up an Nginx server for that domain, and finally following LetsEncrypt instructions for Nginx setup. This will auto-generate an SSL certificate and include it in your Nginx configuration. This will let you connect Polkadot-JS UI to a URL like mynode.mydomain.com rather than 82.196.8.192:9944, which is arguably more user friendly.

This is simple to do on cloud hosting providers or if you have a static IP, but harder to pull off when running things from your home server.

Self-signed

The second approach and one we'll follow here is generating a self-signed certificate and relying on the raw IP address of your node when connecting to it.

Generate a self-signed certificate.

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/nginx-selfsigned.key -out
/etc/ssl/certs/nginx-selfsigned.crt
sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048
```

Set up Nginx server

Now it's time to tell Nginx to use these certificates. The server block below is all you need, but keep in mind that you need to replace some placeholder values. Notably:

- `SERVER_ADDRESS` should be replaced by your domain name if you have it, or your server's IP address if not.
- `CERT_LOCATION` should be `/etc/letsencrypt/live/YOUR_DOMAIN/fullchain.pem` if you used Certbot, or `/etc/ssl/certs/nginx-selfsigned.crt` if self-signed.
- `CERT_LOCATION_KEY` should be `/etc/letsencrypt/live/YOUR_DOMAIN/privkey.pem` if you used Certbot, or `/etc/ssl/private/nginx-selfsigned.key` if self-signed.
- `CERT_DHPARAM` should be `/etc/letsencrypt/ssl-dhparams.pem` if you used Certbot, and `/etc/ssl/certs/dhparam.pem` if self-signed.

Note that if you used Certbot, it should have made the path insertions below for you if you followed the official instructions

```
server {

        server_name SERVER_ADDRESS;

        root /var/www/html;
        index index.html;

        location / {
          try_files $uri $uri/ =404;

          proxy_pass http://localhost:9944;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header Host $host;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

          proxy_http_version 1.1;
          proxy_set_header Upgrade $http_upgrade;
          proxy_set_header Connection "upgrade";
        }

        listen [::]:443 ssl ipv6only=on;
        listen 443 ssl;
        ssl_certificate CERT_LOCATION;
        ssl_certificate_key CERT_LOCATION_KEY;

        ssl_session_cache shared:cache_nginx_SSL:1m;
        ssl_session_timeout 1440m;

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers on;

        ssl_ciphers "ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-
SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-
SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-
RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-
DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-
SHA:AES256-SHA:DES-CBC3-SHA:!DSS";

        ssl_dhparam CERT_DHPARAM;

}
```

Restart nginx after setting this up: `sudo service nginx restart`.

Importing the Certificate

If you used the self-signed certificate approach, modern browsers will not let you connect to this websocket endpoint without that certificate being imported - they will emit an `NET:ERR_CERT_AUTHORITY_INVALID` message.

ERR_CERT_AUTHORITY_INVALID

Every websocket connection bootstraps itself with `https` first, so to allow the certificate, visit the IP of your machine in the browser prefixed with `https`, like so: `https://MY_IP`. This should produce a "Not private" warning which you can skip by going to "Advanced" and the clicking on "Proceed to Site". You have now whitelisted this IP and its self-signed certificate for connecting.

Connecting to the node

Open Polkadot-JS UI and click the logo in the top left to switch the node. Activate the "Development" toggle and input your node's address - either the domain or the IP address. Remember to prefix with `wss://` and if you're using the 443 port, append `:443`, like so: `wss://example.com:443`.

Now you have a secure remote connect setup for your Substrate node.

Errors and How to Resolve Them

Errors in Substrate-based chains are usually accompanied by descriptive messages. However, to read these messages, a tool parsing the blockchain data needs to request *chain metadata* from a node. That metadata explains how to read the messages. One such tool with a built-in parser for chain metadata is the Polkadot-JS Apps UI.

If this page does not answer your question, try searching for your problem at the Polkadot Knowledge Base for more information on troubleshooting your issue.

PolkadotJS Apps Explorer

Here's how to find out the detailed error description through Polkadot-JS Apps.

A typical failed transactions looks something like this:

The image displays only the error name as defined in the code, not its error message. Despite this error being rather self-explanatory, let's find its details.

In the explorer tab, find the block in which this failure occured. Then, expand the `system.ExtrinsicFailed` frame:

Notice how the `details` field contains a human-readable description of the error. Most errors will have this, if looked up this way.

This block is a live example of the above.

If you cannot look up the error this way, or there is no message in the `details` field, consult the table below.

Polkascan and Subscan

Polkascan and Subscan show the `ExtrinsicFailed` event when a transaction does not succeed (example). This event gives us the `error` and `index` indices of the error but does not give us a nice message to understand what it means. We will look up the error in the codebase ourselves to understand what went wrong.

First, we should understand that the `index` number is the index of the pallet in the runtime from which the error originated. The `error` is likewise the index of that pallet's errors which is the exact one we're looking for. Both of these indices start counting from 0.

For example, if `index` is 5 and `error` is 3, as in the example linked above, we need to look at the runtime for the fourth error (index 3) in the sixth pallet (index 5).

By looking at the runtime code we see that the pallet at index 5 is Balances. Now we will check the Balances pallet's code which is hosted in the Substrate repository, and look for the fourth error in the Error enum. According to its source the error that we got is InsufficientBalance or in other words "Balance too low to send value".

Common Errors

The table below lists the most commonly encountered errors and ways to resolve them.

Error	Description	Solution
BadOrigin	You are not allowed to do this operation, e.g. trying to create a council motion with a non-council account.	Either switch to an account that has the necessary permissions, or check if the operation you're trying to execute is permitted at all (e.g. calling `system.setCode` to do a runtime upgrade directly, without voting).

Error	Description	Solution
BadProof	The transaction's signature seems invalid.	It's possible that the node you're connected to is following an obsolete fork - trying again after it catches up usually resolves the issue. To check for bigger problems, inspect the last finalized and current best block of the node you're connected to and compare the values to chain stats exposed by other nodes - are they in sync? If not, try connecting to a different node.
Future	Transaction nonce too high, i.e. it's "from the future".	Reduce the nonce to +1 of current nonce. Check current nonce by inspecting the address you're using to send the transaction.
Stale	Transaction nonce too low.	Increase the nonce to +1 of current nonce. Check current nonce by inspecting the address you're using to send the transaction.
ExhaustsResources	There aren't enough resources left in the current block to submit this transaction.	Try again in the next block.
Payment	Unable to pay for TX fee.	You might not have enough free balance to cover the fee this transaction would incur.
Temporarily banned	The transaction is temporarily banned.	The tx is already in pool. Either try on a different node, or wait to see if the initial transaction goes through.

Error Table

The below table is a reference to the errors that exists in Polkadot. It is generated from the runtime's metadata.

Pallet	Error	Documentation
System (0)		
	InvalidSpecName (0)	The name of specification does not match between the current runtime and the new runtime.
	SpecVersionNeedsToIncrease (1)	The specification version is not allowed to decrease between the current runtime and the new runtime.
	FailedToExtractRuntimeVersion (2)	Failed to extract the runtime version from the new runtime. Either calling `Core_version` or decoding `RuntimeVersion` failed.
	NonDefaultComposite (3)	Suicide called when the account has non-default composite data.
	NonZeroRefCount (4)	There is a non-zero reference count preventing the account from being purged.
Scheduler (1)		
	FailedToSchedule (0)	Failed to schedule a call
	NotFound (1)	Cannot find the scheduled call.
	TargetBlockNumberInPast (2)	Given target block number is in the past.
	RescheduleNoChange (3)	Reschedule failed because it does not change scheduled time.

Pallet	Error	Documentation
Balances (5)		
	VestingBalance (0)	Vesting balance too high to send value
	LiquidityRestrictions (1)	Account liquidity restrictions prevent withdrawal
	Overflow (2)	Got an overflow after adding
	InsufficientBalance (3)	Balance too low to send value
	ExistentialDeposit (4)	Value too low to create account due to existential deposit
	KeepAlive (5)	Transfer/payment would kill account
	ExistingVestingSchedule (6)	A vesting schedule already exists for this account
	DeadAccount (7)	Beneficiary account must pre-exist
Authorship (6)		
	InvalidUncleParent (0)	The uncle parent not in the chain.
	UnclesAlreadySet (1)	Uncles already set in the block.
	TooManyUncles (2)	Too many uncles.
	GenesisUncle (3)	The uncle is genesis.
	TooHighUncle (4)	The uncle is too high in chain.
	UncleAlreadyIncluded (5)	The uncle is already included.
	OldUncle (6)	The uncle isn't recent enough to be included.
Staking (7)		
	NotController (0)	Not a controller account.
	NotStash (1)	Not a stash account.
	AlreadyBonded (2)	Stash is already bonded.
	AlreadyPaired (3)	Controller is already paired.
	EmptyTargets (4)	Targets cannot be empty.
	DuplicateIndex (5)	Duplicate index.
	InvalidSlashIndex (6)	Slash record index out of bounds.
	InsufficientValue (7)	Can not bond with value less than minimum balance.
	NoMoreChunks (8)	Can not schedule more unlock chunks.
	NoUnlockChunk (9)	Can not rebond without unlocking chunks.
	FundedTarget (10)	Attempting to target a stash that still has funds.
	InvalidEraToReward (11)	Invalid era to reward.

Pallet	Error	Documentation
	InvalidNumberOfNominations (12)	Invalid number of nominations.
	NotSortedAndUnique (13)	Items are not sorted and unique.
	AlreadyClaimed (14)	Rewards for this era have already been claimed for this validator.
	OffchainElectionEarlySubmission (15)	The submitted result is received out of the open window.
	OffchainElectionWeakSubmission (16)	The submitted result is not as good as the one stored on chain.
	SnapshotUnavailable (17)	The snapshot data of the current window is missing.
	OffchainElectionBogusWinnerCount (18)	Incorrect number of winners were presented.
	OffchainElectionBogusWinner (19)	One of the submitted winners is not an active candidate on chain (index is out of range in snapshot).
	OffchainElectionBogusCompact (20)	Error while building the assignment type from the compact. This can happen if an index is invalid, or if the weights *overflow*.
	OffchainElectionBogusNominator (21)	One of the submitted nominators is not an active nominator on chain.
	OffchainElectionBogusNomination (22)	One of the submitted nominators has an edge to which they have not voted on chain.
	OffchainElectionSlashedNomination (23)	One of the submitted nominators has an edge which is submitted before the last non-zero slash of the target.
	OffchainElectionBogusSelfVote (24)	A self vote must only be originated from a validator to ONLY themselves.
	OffchainElectionBogusEdge (25)	The submitted result has unknown edges that are not among the presented winners.
	OffchainElectionBogusScore (26)	The claimed score does not match with the one computed from the data.
	OffchainElectionBogusElectionSize (27)	The election size is invalid.
	CallNotAllowed (28)	The call is not allowed at the given time due to restrictions of election period.
	IncorrectHistoryDepth (29)	Incorrect previous history depth input provided.
	IncorrectSlashingSpans (30)	Incorrect number of slashing spans provided.
Session (9)		
	InvalidProof (0)	Invalid ownership proof.
	NoAssociatedValidatorId (1)	No associated validator ID for account.
	DuplicatedKey (2)	Registered duplicate key.

Pallet	Error	Documentation
	NoKeys (3)	No keys are associated with this account.
Grandpa (11)		
	PauseFailed (0)	Attempt to signal GRANDPA pause when the authority set isn't live (either paused or already pending pause).
	ResumeFailed (1)	Attempt to signal GRANDPA resume when the authority set isn't paused (either live or already pending resume).
	ChangePending (2)	Attempt to signal GRANDPA change with one already pending.
	TooSoon (3)	Cannot signal forced change so soon after last.
	InvalidKeyOwnershipProof (4)	A key ownership proof provided as part of an equivocation report is invalid.
	InvalidEquivocationProof (5)	An equivocation proof provided as part of an equivocation report is invalid.
	DuplicateOffenceReport (6)	A given equivocation report is valid but already previously reported.
ImOnline (12)		
	InvalidKey (0)	Non existent public key.
	DuplicatedHeartbeat (1)	Duplicated heartbeat.
Democracy (14)		
	ValueLow (0)	Value too low
	ProposalMissing (1)	Proposal does not exist
	BadIndex (2)	Unknown index
	AlreadyCanceled (3)	Cannot cancel the same proposal twice
	DuplicateProposal (4)	Proposal already made
	ProposalBlacklisted (5)	Proposal still blacklisted
	NotSimpleMajority (6)	Next external proposal not simple majority
	InvalidHash (7)	Invalid hash
	NoProposal (8)	No external proposal
	AlreadyVetoed (9)	Identity may not veto a proposal twice
	NotDelegated (10)	Not delegated
	DuplicatePreimage (11)	Preimage already noted
	NotImminent (12)	Not imminent
	TooEarly (13)	Too early
	Imminent (14)	Imminent

Pallet	Error	Documentation
	PreimageMissing (15)	Preimage not found
	ReferendumInvalid (16)	Vote given for invalid referendum
	PreimageInvalid (17)	Invalid preimage
	NoneWaiting (18)	No proposals waiting
	NotLocked (19)	The target account does not have a lock.
	NotExpired (20)	The lock on the account to be unlocked has not yet expired.
	NotVoter (21)	The given account did not vote on the referendum.
	NoPermission (22)	The actor has no permission to conduct the action.
	AlreadyDelegating (23)	The account is already delegating.
	Overflow (24)	An unexpected integer overflow occurred.
	Underflow (25)	An unexpected integer underflow occurred.
	InsufficientFunds (26)	Too high a balance was provided that the account cannot afford.
	NotDelegating (27)	The account is not currently delegating.
	VotesExist (28)	The account currently has votes attached to it and the operation cannot succeed until these are removed, either through `unvote` or `reap_vote`.
	InstantNotAllowed (29)	The instant referendum origin is currently disallowed.
	Nonsense (30)	Delegation to oneself makes no sense.
	WrongUpperBound (31)	Invalid upper bound.
	MaxVotesReached (32)	Maximum number of votes reached.
	InvalidWitness (33)	The provided witness data is wrong.
	TooManyProposals (34)	Maximum number of proposals reached.
Council (15)		
	NotMember (0)	Account is not a member
	DuplicateProposal (1)	Duplicate proposals not allowed
	ProposalMissing (2)	Proposal must exist
	WrongIndex (3)	Mismatched index
	DuplicateVote (4)	Duplicate vote ignored
	AlreadyInitialized (5)	Members are already initialized!
	TooEarly (6)	The close call was made too early, before the end of the voting.

Pallet	Error	Documentation
	TooManyProposals (7)	There can only be a maximum of `MaxProposals` active proposals.
	WrongProposalWeight (8)	The given weight bound for the proposal was too low.
	WrongProposalLength (9)	The given length bound for the proposal was too low.
TechnicalCommittee (16)		
	NotMember (0)	Account is not a member
	DuplicateProposal (1)	Duplicate proposals not allowed
	ProposalMissing (2)	Proposal must exist
	WrongIndex (3)	Mismatched index
	DuplicateVote (4)	Duplicate vote ignored
	AlreadyInitialized (5)	Members are already initialized!
	TooEarly (6)	The close call was made too early, before the end of the voting.
	TooManyProposals (7)	There can only be a maximum of `MaxProposals` active proposals.
	WrongProposalWeight (8)	The given weight bound for the proposal was too low.
	WrongProposalLength (9)	The given length bound for the proposal was too low.
ElectionsPhragmen (17)		
	UnableToVote (0)	Cannot vote when no candidates or members exist.
	NoVotes (1)	Must vote for at least one candidate.
	TooManyVotes (2)	Cannot vote more than candidates.
	MaximumVotesExceeded (3)	Cannot vote more than maximum allowed.
	LowBalance (4)	Cannot vote with stake less than minimum balance.
	UnableToPayBond (5)	Voter can not pay voting bond.
	MustBeVoter (6)	Must be a voter.
	ReportSelf (7)	Cannot report self.
	DuplicatedCandidate (8)	Duplicated candidate submission.
	MemberSubmit (9)	Member cannot re-submit candidacy.
	RunnerSubmit (10)	Runner cannot re-submit candidacy.
	InsufficientCandidateFunds (11)	Candidate does not have enough funds.
	NotMember (12)	Not a member.

Pallet	Error	Documentation
	InvalidCandidateCount (13)	The provided count of number of candidates is incorrect.
	InvalidVoteCount (14)	The provided count of number of votes is incorrect.
	InvalidRenouncing (15)	The renouncing origin presented a wrong `Renouncing` parameter.
	InvalidReplacement (16)	Prediction regarding replacement after member removal is wrong.
Treasury (19)		
	InsufficientProposersBalance (0)	Proposer's balance is too low.
	InvalidIndex (1)	No proposal or bounty at that index.
	ReasonTooBig (2)	The reason given is just too big.
	AlreadyKnown (3)	The tip was already found/started.
	UnknownTip (4)	The tip hash is unknown.
	NotFinder (5)	The account attempting to retract the tip is not the finder of the tip.
	StillOpen (6)	The tip cannot be claimed/closed because there are not enough tippers yet.
	Premature (7)	The tip cannot be claimed/closed because it's still in the countdown period.
	UnexpectedStatus (8)	The bounty status is unexpected.
	RequireCurator (9)	Require bounty curator.
	InvalidValue (10)	Invalid bounty value.
	InvalidFee (11)	Invalid bounty fee.
	PendingPayout (12)	A bounty payout is pending. To cancel the bounty, you must unassign and slash the curator.
Claims (24)		
	InvalidEthereumSignature (0)	Invalid Ethereum signature.
	SignerHasNoClaim (1)	Ethereum address has no claim.
	SenderHasNoClaim (2)	Account ID sending tx has no claim.
	PotUnderflow (3)	There's not enough in the pot to pay out some unvested amount. Generally implies a logic error.
	InvalidStatement (4)	A needed statement was not included.
	VestedBalanceExists (5)	The account already has a vested balance.
Vesting (25)		
	NotVesting (0)	The account given is not vesting.

Pallet	Error	Documentation
	ExistingVestingSchedule (1)	An existing vesting schedule already exists for this account that cannot be clobbered.
	AmountLow (2)	Amount being transferred is too low to create a vesting schedule.
Identity (28)		
	TooManySubAccounts (0)	Too many subs-accounts.
	NotFound (1)	Account isn't found.
	NotNamed (2)	Account isn't named.
	EmptyIndex (3)	Empty index.
	FeeChanged (4)	Fee is changed.
	NoIdentity (5)	No identity found.
	StickyJudgement (6)	Sticky judgement.
	JudgementGiven (7)	Judgement given.
	InvalidJudgement (8)	Invalid judgement.
	InvalidIndex (9)	The index is invalid.
	InvalidTarget (10)	The target is invalid.
	TooManyFields (11)	Too many additional fields.
	TooManyRegistrars (12)	Maximum amount of registrars reached. Cannot add any more.
	AlreadyClaimed (13)	Account ID is already named.
	NotSub (14)	Sender is not a sub-account.
	NotOwned (15)	Sub-account isn't owned by sender.
Proxy (29)		
	TooMany (0)	There are too many proxies registered or too many announcements pending.
	NotFound (1)	Proxy registration not found.
	NotProxy (2)	Sender is not a proxy of the account to be proxied.
	Unproxyable (3)	A call which is incompatible with the proxy type's filter was attempted.
	Duplicate (4)	Account is already a proxy.
	NoPermission (5)	Call may not be made by proxy because it may escalate its privileges.
	Unannounced (6)	Announcement, if made at all, was made too recently.
Multisig (30)		

Pallet	Error	Documentation
	MinimumThreshold (0)	Threshold must be 2 or greater.
	AlreadyApproved (1)	Call is already approved by this signatory.
	NoApprovalsNeeded (2)	Call doesn't need any (more) approvals.
	TooFewSignatories (3)	There are too few signatories in the list.
	TooManySignatories (4)	There are too many signatories in the list.
	SignatoriesOutOfOrder (5)	The signatories were provided out of order; they should be ordered.
	SenderInSignatories (6)	The sender was contained in the other signatories; it shouldn't be.
	NotFound (7)	Multisig operation not found when attempting to cancel.
	NotOwner (8)	Only the account that originally created the multisig is able to cancel it.
	NoTimepoint (9)	No timepoint was given, yet the multisig operation is already underway.
	WrongTimepoint (10)	A different timepoint was given to the multisig operation that is underway.
	UnexpectedTimepoint (11)	A timepoint was given, yet no multisig operation is underway.
	WeightTooLow (12)	The maximum weight information provided was too low.
	AlreadyStored (13)	The data to be stored is already stored.

How to Nominate on Polkadot

> The following information applies to the Polkadot network. If you want to nominate on Kusama, check out the Kusama guide instead.

Nominators are one type of participant in the staking subsystem of Polkadot. They are responsible for appointing their stake to the validators who are the second type of participant. By appointing their stake, they are able to elect the active set of validators and share in the rewards that are paid out.

While the validators are active participants in the network that engage in the block production and finality mechanisms, nominators take a slightly more passive role. Being a nominator does not require running a node of your own or worrying about online uptime. However, a good nominator performs due diligence on the validators that they elect. When looking for validators to nominate, a nominator should pay attention to their own reward percentage for nominating a specific validator - as well as the risk that they bare of being slashed if the validator gets slashed.

Setting up Stash and Controller keys

> If you prefer a video format, the following videos related to staking are available:
>
> - Staking with a Ledger and PolkadotJS Apps
> - Staking with a Ledger and Ledger Live

Nominators are recommended to set up separate stash and controller accounts. Explanation and reasoning for generating distinct accounts for this purpose is elaborated in the keys section of the Wiki.

You can generate your stash and controller account via any of the recommended methods that are detailed on the account generation page.

Starting with runtime version v23 natively included in client version 0.8.23, payouts can go to any custom address. If you'd like to redirect payments to an account that is neither the controller nor the stash account, set one up. Note that it is extremely unsafe to set an exchange address as the recipient of the staking rewards.

Using Polkadot-JS UI

Step 1: Bond your tokens

On the Polkadot-JS UI navigate to the "Staking" tab (within the "Network" menu).

The "Staking Overview" subsection will show you all the active validators and their information - their identities, the amount of DOT that are staking for them, amount that is their own provided stake, how much they charge in commission, the era points they've earned in the current era, and the last block number that they produced. If you click on the chart button it will take you to the "Validator Stats" page for that validator that shows you more detailed and historical information about the validator's stake, rewards and slashes.

The "Account actions" subsection (link) allows you to stake and nominate.

The "Payouts" subsection (link) allows you to claim rewards from staking.

The "Targets" subsection (link) will help you estimate your earnings and this is where it's good to start picking favorites.

The "Waiting" subsection (link) lists all pending validators that are awaiting more nominations to enter the active validator set. Validators will stay in the waiting queue until they have enough DOT backing them (as allocated through the Phragmén election mechanism). It is possible validator can remain in the queue for a very long time if they never get enough backing.

The "Validator Stats" subsection (link) allows you to query a validator's stash address and see historical charts on era points, elected stake, rewards, and slashes.

Pick "Account actions", then click the "+ Nominator" button.

You will see a modal window that looks like the below:

Select a "value bonded" that is **less** than the total amount of DOT you have, so you have some left over to pay transaction fees. Transaction fees are currently around 0.01 DOT, but they are dynamic based on a variety of factors including the load of recent blocks.

Also be mindful of the reaping threshold - the amount that must remain in an account lest it be burned. That amount is 1 DOT on Polkadot, so it's recommended to keep at least 1.5 DOT in your account to be on the safe side.

Choose whatever payment destination that makes sense to you. If you're unsure, you can choose "Stash account (increase amount at stake)" to simply accrue the rewards into the amount you're staking and earn compound interest.

> These concepts have been further explained in Polkadot's UI Walkthrough Video

Step 2: Nominate a validator

You are now bonded. Being bonded means your tokens are locked and could be slashed if the validators you nominate misbehave. All bonded funds can now be distributed to up to 16 validators. Be careful about the validators you choose since you will be slashed if your validator commits an offence.

Click on "Nominate" on an account you've bonded and you will be presented with another popup asking you to select up to 16 validators. Although you may choose up to 16 validators, due to the Phragmén election algorithm your stake may be dispersed in different proportions to any subset or all of the validators your choose.

Select them, confirm the transaction, and you're done - you are now nominating. Your nominations will become active in the next era. Eras last twenty-four hours on Polkadot - depending on when you do this, your nominations may become active almost immediately, or you may have to wait almost the entire twenty-four hours before your nominations are active. You can chek how far along Polkadot is in the current era on the Staking page.

Assuming at least one of your nominations ends up in the active validator set, you will start to get rewards allocated to you. In order to claim them (i.e., add them to your account), you must manually claim them. See the Claiming Rewards section of the Staking wiki page for more details.

Step 3: Stop nominating

At some point, you might decide to stop nominating one or more validators. You can always change who you're nominating, but you cannot withdraw your tokens unless you unbond them. Detailed instructions are available here.

Using Command-Line Interface (CLI)

Apart from using Polkadot-JS Apps to participate in staking, you can do all these things in CLI instead. The CLI approach allows you to interact with the Polkadot network without going to the Polkadot-JS Apps dashboard.

Step 1: Install @polkadot/api-cli

We assume you have installed NodeJS with npm. Run the following command to install the `@polkadot/api-cli` globally:

```
npm install -g @polkadot/api-cli
```

Step 2. Bond your DOT

Executing the following command:

```
polkadot-js-api --seed "MNEMONIC_PHRASE" tx.staking.bond CONTROLLER_ADDRESS NUMBER_OF_TOKENS REWARD_DESTINATION -
-ws WEBSOCKET_ENDPOINT
```

`CONTROLLER_ADDRESS`: An address you would like to bond to the stash account. Stash and Controller can be the same address but it is not recommended since it defeats the security of the two-account staking model.

`NUMBER_OF_TOKENS`: The number of DOT you would like to stake to the network.

> **Note**: DOT has ten decimal places and is always represented as an integer with zeroes at the end. So 1 DOT = 10_000_000_000 Plancks.

`REWARD_DESTINATION`:

- `Staked` - Pay into the stash account, increasing the amount at stake accordingly.
- `Stash` - Pay into the stash account, not increasing the amount at stake.
- `Account` - Pay into a custom account, like so: `Account DMTHrNcmA8QbqRS4rBq8LXn8ipyczFoNMb1X4cY2WD9tdBX`.
- `Controller` - Pay into the controller account.

Example:

```
polkadot-js-api --seed "xxxx xxxxx xxxx xxxxx" tx.staking.bond DMTHrNcmA8QbqRS4rBq8LXn8ipyczFoNMb1X4cY2WD9tdBX
1000000000000 Staked --ws wss://rpc.polkadot.io
```

Result:

```
...
...
    "status": {
      "InBlock": "0x0ed1ec0ba69564e8f98958d69f826adef895b5617366a32a3aa384290e98514e"
    }
```

You can check the transaction status by using the value of the `InBlock` in Polkascan. Also, you can verify the bonding state under the Staking page on the Polkadot-JS Apps Dashboard.

Step 3. Nominate a validator

To nominate a validator, you can execute the following command:

```
polkadot-js-api --seed "MNEMONIC_PHRASE" tx.staking.nominate '["VALIDATOR_ADDRESS"]' --ws WS_ENDPOINT
```

```
polkadot-js-api --seed "xxxx xxxxx xxxx xxxxx" tx.staking.nominate
'["CmD9vaMYoiKe7HiFnfkftwvhKbxN9bhyjcDrfFRGbifJEG8","E457XaKbj2yTB2URy8N4UuzmyuFRkcdxYs67UvSgVr7HyFb"]' --ws
wss://rpc.polkadot.io
```

After a few seconds, you should see the hash of the transaction and if you would like to verify the nomination status, you can check that on the Polkadot-JS UI as well.

Unbonding and Rebonding

The following describes how to stop nominating or validating and retrieve your tokens. Please note that all networks on which you can nominate have a delayed exit period, called the *unbonding period*, which serves as a cooldown. You will not be able to transfer your tokens before this period has elapsed, and you will not receive any staking rewards during this period (as you are not nominating any validators).

Step 1: Stop Nominating

On the Polkadot-JS Apps navigate to the "Staking" tab.

On this tab click on the "Account Actions" tab at the top of the screen.

Here, click "Stop Nominating" or "Stop Validating" (depending on your role) on an account you're staking with and would like to free the funds for. This will "chill" the tokens.

After you confirm this transaction, your tokens will remain *bonded*. This means they stay ready to be distributed among nominees or used as validator self-stake again. To actually withdraw them, you need to unbond.

Step 2: Unbonding an amount

To unbond the amount, click the little gear icon next to the account you want to unbond tokens for, and select "Unbond funds".

Select the amount you wish to unbond and click Unbond, then confirm the transaction.

If successful, your balance will show as "unbonding" with an indicator of how many more blocks remain until the amount is fully unlocked.

This duration will vary depending on the network you're on and will typically be four times as fast on Kusama as it is on Polkadot.

Once this process is complete, you will have to issue another, final transaction: Withdraw Unbonded.

You can also check how long you have to wait in order to withdraw your stake in the Accounts page by expanding your account balance. There is a tiny icon beside the word "unbonding" that will eventually become an unlock icon once the remaning blocks get passed.

Then, you can click that icon directly to submit the withdraw transaction. Finally, your transferrable balance will increase by the amount of tokens you've just fully unbonded.

Rebonding before the end of the unbonding period

If you want to rebond your tokens before the unbonding period is over you can do this by issuing a `rebond` extrinsic. This allows you to bond your tokens that are still locked without waiting until the end of the unbonding period.

In order to do this you will need to issue an extrinsic manually from Polkadot-JS Apps.

Go to the "Extrinsics" option that's located in the "Developer" dropdown in the top menu.

Select the "staking" pallet and the "rebond" extrinsic. Enter the amount of tokens that are currently locked in unbonding that you want to rebond. Then click "Submit Transaction".

Confirm the transaction in the next pop-up. Once the transaction is included in the next block your tokens will be rebonded again and you can start staking with them.

Run a Validator (Polkadot)

The following information applies to the Polkadot network. If you want to set up a validator on Kusama, check out the Kusama guide instead.

This guide will instruct you how to set up a validator node on the Polkadot network.

Preliminaries

Running a validator on a live network is a lot of responsibility! You will be accountable for not only your own stake, but also the stake of your current nominators. If you make a mistake and get slashed, your money and your reputation will be at risk. However, running a validator can also be very rewarding, knowing that you contribute to the security of a decentralized network while growing your stash.

Warning: It is highly recommended that you have significant system administration experience before attempting to run your own validator.

Since security is so important to running a successful validator, you should take a look at the secure validator information to make sure you understand the factors to consider when constructing your infrastructure. The Web3 Foundation also maintains a reference implementation for a secure validator set-up that you can use by deploying yourself (video walkthrough is available here). As you progress in your journey as a validator, you will likely want to use this repository as a *starting point* for your own modifications and customizations.

If you need help, please reach out on the Polkadot Validator Lounge on Riot. The team and other validators are there to help answer questions and provide tips from experience.

How many DOT do I need?

You can have a rough estimate on that by using the methods listed here. Validators are elected based on Phragmén's algorithm. To be elected into the set, you need a minimum stake behind your validator. This stake can come from yourself or from nominators. This means that as a minimum, you will need enough DOT to set up Stash and Controller accounts with the existential deposit, plus a little extra for transaction fees. The rest can come from nominators.

Warning: Any DOT that you stake for your validator is liable to be slashed, meaning that an insecure or improper setup may result in loss of DOT tokens! If you are not confident in your ability to run a validator node, it is recommended to nominate your DOT to a trusted validator node instead.

Initial Set-up

Requirements

The most common way for a beginner to run a validator is on a cloud server running Linux. You may choose whatever VPS provider that your prefer, and whatever operating system you are comfortable with. For this guide we will be using **Ubuntu 18.04**, but the instructions should be similar for other platforms.

The transactions weights in Polkadot were benchmarked on standard hardware. It is recommended that validators run at least the standard hardware in order to ensure they are able to process all blocks in time. The following are not *minimum requirements* but if you decide to run with less than this beware that you might have performance issue.

Standard Hardware

For the full details of the standard hardware please see here.

- **CPU** - Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz
- **Storage** - A NVMe solid state drive. Should be reasonably sized to deal with blockchain growth. Starting around 80GB - 160GB will be okay for the first six months of Polkadot, but will need to be re-evaluated every six months.
- **Memory** - 64GB.

The specs posted above are by no means the minimum specs that you could use when running a validator, however you should be aware that if you are using less you may need to toggle some extra optimizations in order to be equal to other validators that are running the standard.

Install Rust

Once you choose your cloud service provider and set-up your new server, the first thing you will do is install Rust.

If you have never installed Rust, you should do this first. This command will fetch the latest version of Rust and install it.

```
curl https://sh.rustup.rs -sSf | sh
```

Otherwise, if you have already installed Rust, run the following command to make sure you are using the latest version.

```
rustup update
```

Finally, run this command to install the necessary dependencies for compiling and running the Polkadot node software.

```
sudo apt install make clang pkg-config libssl-dev build-essential
```

Note - if you are using OSX and you have Homebrew installed, you can issue the following equivalent command INSTEAD of the previous one:

```
brew install cmake pkg-config openssl git llvm
```

Install & Configure Network Time Protocol (NTP) Client

NTP is a networking protocol designed to synchronize the clocks of computers over a network. NTP allows you to synchronize the clocks of all the systems within the network. Currently it is required that validators' local clocks stay reasonably in sync, so you should be running NTP or a similar service. You can check whether you have the NTP client by running:

If you are using Ubuntu 18.04 / 19.04, NTP Client should be installed by default.

```
timedatectl
```

If NTP is installed and running, you should see `System clock synchronized: yes` (or a similar message). If you do not see it, you can install it by executing:

```
sudo apt-get install ntp
```

ntpd will be started automatically after install. You can query ntpd for status information to verify that everything is working:

```
sudo ntpq -p
```

> *WARNING*: Skipping this can result in the validator node missing block authorship opportunities. If the clock is out of sync (even by a small amount), the blocks the validator produces may not get accepted by the network. This will result in `ImOnline` heartbeats making it on chain, but zero allocated blocks making it on chain.

Building and Installing the `polkadot` Binary

You will need to build the `polkadot` binary from the paritytech/polkadot repository on GitHub using the source code available in the **v0.8** branch.

You should generally use the latest **0.8.x** tag. You should either review the output from the "git tag" command or visit the Releases to see a list of all the potential 0.8 releases. You should replace `VERSION` below with the latest build (i.e., the highest number).

Note: If you prefer to use SSH rather than HTTPS, you can replace the first line of the below with `git clone git@github.com:paritytech/polkadot.git`.

```
git clone https://github.com/paritytech/polkadot.git
cd polkadot
git tag -l | sort -V | grep -v -- '-rc'
echo Get the latest version and replace VERSION (below) with it.
git checkout VERSION
./scripts/init.sh
cargo build --release
```

This step will take a while (generally 10 - 40 minutes, depending on your hardware).

Note if you run into compile errors, you may have to switch to a less recent nightly. This can be done by running:

```
rustup install nightly-2020-10-06
rustup target add wasm32-unknown-unknown --toolchain nightly-2020-10-06
cargo +nightly-2020-10-06 build --release
```

If you are interested in generating keys locally, you can also install `subkey` from the same directory. You may then take the generated `subkey` executable and transfer it to an air-gapped machine for extra security.

```
cargo install --force --git https://github.com/paritytech/substrate subkey
```

Synchronize Chain Data

Note: By default, Validator nodes are in archive mode. If you've already synced the chain not in archive mode, you must first remove the database with `polkadot purge-chain` and then ensure that you run Polkadot with the `--pruning=archive` option.

You may run a validator node in non-archive mode by adding the following flags: `--unsafe-pruning --pruning <NUM OF BLOCKS>`, a reasonable value being 1000. Note that an archive node and non-archive node's databases are not compatible with each other, and to switch you will need to purge the chain data.

You can begin syncing your node by running the following command:

```
./target/release/polkadot --pruning=archive
```

if you do not want to start in validator mode right away.

The `--pruning=archive` flag is implied by the `--validator` and `--sentry` flags, so it is only required explicitly if you start your node without one of these two options. If you do not set your pruning to archive node, even when not running in validator and sentry mode, you will need to re-sync your database when you switch.

Note: Validators should sync using the RocksDb backend. This is implicit by default, but can be explicit by passing the `--database RocksDb` flag.

In the future, it is recommended to switch to the faster and more efficient ParityDB option. Note that **ParityDB is still experimental and should not be used in production.** If you want to test out ParityDB, you can add the flag `--database paritydb`. Switching between database backends will require a resync.

Depending on the size of the chain when you do this, this step may take anywhere from a few minutes to a few hours.

If you are interested in determining how much longer you have to go, your server logs (printed to STDOUT from the `polkadot` process) will tell you the latest block your node has processed and verified. You can then compare that to the current highest block via Telemetry or the PolkadotJS Block Explorer.

Bond DOT

It is highly recommended that you make your controller and stash accounts be two separate accounts. For this, you will create two accounts and make sure each of them have at least enough funds to pay the fees for making transactions. Keep most of your funds in the stash account since it is meant to be the custodian of your staking funds.

Make sure not to bond all your DOT balance since you will be unable to pay transaction fees from your bonded balance.

It is now time to set up our validator. We will do the following:

- Bond the DOT of the Stash account. These DOT will be put at stake for the security of the network and can be slashed.
- Select the Controller. This is the account that will decide when to start or stop validating.

First, go to the Staking section. Click on "Account Actions", and then the "+ Stash" button.

- **Stash account** - Select your Stash account. In this example, we will bond 100 milliDOT - make sure that your Stash account contains *at least* this much. You can, of course, stake more than this.
- **Controller account** - Select the Controller account created earlier. This account will also need a small amount of DOT in order to start and stop validating.
- **Value bonded** - How much DOT from the Stash account you want to bond/stake. Note that you do not need to bond all of the DOT in that account. Also note that you can always bond *more* DOT later. However, *withdrawing* any bonded amount requires the duration of the unbonding period. On Kusama, the unbonding period is 7 days. On Polkadot, the planned unbonding period is 28 days.
- **Payment destination** - The account where the rewards from validating are sent. More info here. Starting with runtime version v23 natively included in client version 0.8.23, payouts can go to any custom address. If you'd like to redirect payments to an account that is neither the controller nor the stash account, set one up. Note that it is extremely unsafe to set an exchange address as the recipient of the staking rewards.

Once everything is filled in properly, click `Bond` and sign the transaction with your Stash account.

After a few seconds, you should see an "ExtrinsicSuccess" message. You should now see a new card with all your accounts (note: you may need to refresh the screen). The bonded amount on the right corresponds to the funds bonded by the Stash account.

Set Session Keys

> **Note:** The session keys are consensus critical, so if you are not sure if your node has the current session keys that you made the `setKeys` transaction then you can use one of the two available RPC methods to query your node: hasKey to check for a specific key or hasSessionKeys to check the full session key public key string.

Once your node is fully synced, stop the process by pressing Ctrl-C. At your terminal prompt, you will now start running the node.

```
./target/release/polkadot --validator --name "name on telemetry"
```

You can give your validator any name that you like, but note that others will be able to see it, and it will be included in the list of all servers using the same telemetry server. Since numerous people are using telemetry, it is recommended that you choose something likely to be unique.

Generating the Session Keys

You need to tell the chain your Session keys by signing and submitting an extrinsic. This is what associates your validator node with your Controller account on Polkadot.

Option 1: PolkadotJS-APPS

You can generate your Session keys in the client via the apps RPC. If you are doing this, make sure that you have the PolkadotJS-Apps explorer attached to your validator node. You can configure the apps dashboard to connect to the endpoint of your validator in the Settings tab. If you are connected to a default endpoint hosted by Parity of Web3 Foundation, you will not be able to use this method since making RPC requests to this node would effect the local keystore hosted on a *public node* and you want to make sure you are interacting with the keystore for *your node*.

Once ensuring that you have connected to your node, the easiest way to set session keys for your node is by calling the `author_rotateKeys` RPC request to create new keys in your validator's keystore. Navigate to Toolbox tab and select RPC Calls then select the author > rotateKeys() option and remember to save the output that you get back for a later step.

Option 2: CLI

If you are on a remote server, it is easier to run this command on the same machine (while the node is running with the default HTTP RPC port configured):

```
curl -H "Content-Type: application/json" -d '{"id":1, "jsonrpc":"2.0", "method": "author_rotateKeys", "params":
[]}' http://localhost:9933
```

The output will have a hex-encoded "result" field. The result is the concatenation of the four public keys. Save this result for a later step.

You can restart your node at this point.

Submitting the `setKeys` Transaction

You need to tell the chain your Session keys by signing and submitting an extrinsic. This is what associates your validator with your Controller account.

Go to Staking > Account Actions, and click "Set Session Key" on the bonding account you generated earlier. Enter the output from `author_rotateKeys` in the field and click "Set Session Key".

Submit this extrinsic and you are now ready to start validating.

Validate

To verify that your node is live and synchronized, head to Telemetry and find your node. Note that this will show all nodes on the Polkadot network, which is why it is important to select a unique name!

If everything looks good, go ahead and click on "Validate" in Polkadot UI.

- **Payment preferences** - You can specify the percentage of the rewards that will get paid to you. The remaining will be split among your nominators.

Click "Validate".

If you go to the "Staking" tab, you will see a list of active validators currently running on the network. At the top of the page, it shows the number of validator slots that are available as well as the number of nodes that have signaled their intention to be a validator. You can go to the "Waiting" tab to double check to see whether your node is listed there.

The validator set is refreshed every era. In the next era, if there is a slot available and your node is selected to join the validator set, your node will become an active validator. Until then, it will remain in the *waiting* queue. If your validator is not selected to become part of the validator set, it will remain in the *waiting* queue until it is. There is no need to re-start if you are not selected for the validator set in a particular era. However, it may be necessary to increase the number of DOT staked or seek out nominators for your validator in order to join the validator set.

Congratulations! If you have followed all of these steps, and been selected to be a part of the validator set, you are now running a Polkadot validator! If you need help, reach out on the Polkadot Validator chat.

Thousand Validators Programme

The Thousand Validators Programme is a joint initiative by Web3 Foundation and Parity Technologies to provide support for community validators. If you are interested in applying for the programme, you can find more information on the wiki page.

FAQ

Why am I unable to synchronize the chain with 0 peers?

Make sure to enable `30333` libp2p port. Eventually, it will take a little bit of time to discover other peers over the network.

How do I clear all my chain data?

```
./target/release/polkadot purge-chain
```

VPS List

- OVH
- Digital Ocean
- Vultr
- Linode
- Contabo
- Scaleway

Using Docker

If you have Docker installed, you can use it to start your validator node without needing to build the binary. You can do this with a simple one line command:

```
$ docker run parity/polkadot:latest --validator --name "name on telemetry"
```

Validator Payout Overview

Era Points

For every era (a period of time approximately 6 hours in length in Kusama, and 24 hours in Polkadot), validators are paid proportionally to the amount of *era points* they have collected. Era points are reward points earned for payable actions like:

- issuing validity statements for parachain blocks.
- producing a non-uncle block in the Relay Chain.
- producing a reference to a previously unreferenced uncle block.
- producing a referenced uncle block.

Note: An uncle block is a Relay Chain block that is valid in every regard, but which failed to become canonical. This can happen when two or more validators are block producers in a single slot, and the block produced by one validator reaches the next block producer before the others. We call the lagging blocks uncle blocks.

Payments occur at the end of every era.

Payout Scheme

No matter how much total stake is behind a validator, all validators split the block authoring payout essentially equally. The payout of a specific validator, however, may differ based on era points, as described above. Although there is a probabilistic component to receiving era points, and they may be impacted slightly depending on factors such as network connectivity, well-behaving validators should generally average out to having similar era point totals over a large number of eras.

Validators may also receive "tips" from senders as an incentive to include transactions in their produced blocks. Validators will receive 100% of these tips directly.

Validators will receive staking rewards in the form of the native token of that chain (KSM for Kusama and DOT for Polkadot).

For simplicity, the examples below will assume all validators have the same amount of era points, and received no tips.

```
Validator Set Size (v): 4
Validator 1 Stake (v1): 18 tokens
Validator 2 Stake (v2):  9 tokens
Validator 3 Stake (v3):  8 tokens
Validator 4 Stake (v4):  7 tokens
Payout (p): 8 DOT

Payout for each validator (v1 - v4):
p / v = 8 / 4 = 2 tokens
```

Note that this is different than most other Proof-of-Stake systems such as Cosmos. As long as a validator is in the validator set, it will receive the same block reward as every other validator. Validator `v1`, who had 18 tokens staked, received the same reward (2 tokens) in this era as `v4` who had only 7 tokens staked.

Running Multiple Validators

It is possible for a single entity to run multiple validators. Running multiple validators may provide a better risk/reward ratio. Assuming you have enough DOT, or enough stake nominates your validator, to ensure that your validators remain in the validator set, running multiple validators will result in a higher return than running a single validator.

For the following example, assume you have 18 DOT to stake. For simplicity's sake, we will ignore nominators. Running a single validator, as in the example above, would net you 2 DOT in this era.

Note that while DOT is used as an example, this same formula would apply to KSM when running a validator on Kusama.

```
Validator Set Size (v): 4
Validator 1 Stake (v1): 18 DOT <- Your validator
Validator 2 Stake (v2):  9 DOT
Validator 3 Stake (v3):  8 DOT
Validator 4 Stake (v4):  7 DOT
Payout (p): 8 DOT

Your payout = (p / v) * 1 = (8 / 4) * 1 = 2
```

Running two validators, and splitting the stake equally, would result in the original validator `v4` to be kicked out of the validator set, as only the top `v` validators (as measured by stake) are selected to be in the validator set. More important, it would also double the reward that you get from each era.

```
Validator Set Size (v): 4
Validator 1 Stake (v1): 9 DOT <- Your first validator
Validator 2 Stake (v2): 9 DOT <- Your second validator
Validator 3 Stake (v3): 9 DOT
Validator 4 Stake (v4): 8 DOT
Payout (p): 8 DOT

Your payout = (p / v) * 1 = (8 / 4) * 2 = 4
```

With enough stake, you could run more than two validators. However, each validator must have enough stake behind it to be in the validator set.

The incentives of the system favor equally-staked validators. This works out to be a dynamic, rather than static, equilibrium. Potential validators will run different numbers of validators and apply different amounts of stake to them as time goes on, and in response to the actions of other validators on the network.

Slashing

Although rewards are paid equally, slashes are relative to a validator's stake. Therefore, if you do have enough DOT to run multiple validators, it is in your best interest to do so. A slash of 30% will, of course, be more DOT for a validator with 18 DOT staked than one with 9 DOT staked.

Running multiple validators does not absolve you of the consequences of misbehavior. Polkadot punishes attacks that appear coordinated more severely than individual attacks. You should not, for example, run multiple validators hosted on the same infrastructure. A proper multi-validator configuration would ensure that they do not fail simultaneously.

Nominators have the incentive to nominate the lowest-staked validator, as this will result in the lowest risk and highest reward. This is due to the fact that while their vulnerability to slashing remains the same (since it is percentage-based), their rewards are higher since they will be a higher proportion of the total stake allocated to that validator.

To clarify this, let us imagine two validators, `v1` and `v2`. Assume both are in the active set, have commission set to 0%, and are well-behaved. The only difference is that `v1` has 90 DOT nominating it and `v2` only has 10. If you nominate `v1`, it now has `90 + 10 = 100` DOT, and you will get 10% of the staking rewards for the next era. If you nominate `v2`, it now has `10 + 10 = 20` DOT nominating it, and you will get 50% of the staking rewards for the next era. In actuality, it would be quite rare to see such a large difference between the stake of validators, but the same principle holds even for smaller differences. If there is a 10% slash of either validator, then you will lose 1 DOT in each case.

Nominators and Validator Payments

Nominated stake allows you to "vote" for validators and share in the rewards (and slashing) without running a validator node yourself. Validators can choose to keep a percentage of the rewards due to their validator to "reimburse" themselves for the cost of running a validator node. Other than that, all rewards are shared based on the stake behind each validator. This includes the stake of the validator itself, plus any stake bonded by nominators.

NOTE: Validators set their preference as a percentage of the block reward, *not* an absolute number of DOT. Polkadot's block reward is based on the *total* amount at stake, with the reward peaking when the amount staked is at 50% of the total supply. The commission is set as the amount taken by the validator; that is, 0% commission means that the validator does not receive any proportion of the rewards besides that owed to it from self-stake, and 100% commission means that the validator operator gets all rewards and gives none to its nominators.

In the following examples, we can see the results of several different validator payment schemes and split between nominator and validator stake. We will assume a single nominator for each validator. However, there can be numerous nominators for each validator. Rewards are still distributed proportionally - for example, if the total rewards to be given to nominators is 2 DOT, and there are four nominators with equal stake bonded, each will receive 0.5 DOT. Note also that a single nominator may stake different validators.

Each validator in the example has selected a different validator payment (that is, a percentage of the reward set aside directly for the validator before sharing with all bonded stake). The validator's payment percentage (in DOT, although the same calculations work for KSM) is listed in brackets (`[]`) next to each validator. Note that since the validator payment is public knowledge, having a low or non-existent validator payment may attract more stake from nominators, since they know they will receive a larger reward.

```
Validator Set Size (v): 4
Validator 1 Stake (v1) [20% commission]: 18 DOT (9 validator, 9 nominator)
Validator 2 Stake (v2) [40% commission]:  9 DOT (3 validator, 6 nominator)
Validator 3 Stake (v3) [10% commission]:  8 DOT (4 validator, 4 nominator)
Validator 4 Stake (v4) [ 0% commission]:  6 DOT (1 validator, 5 nominator)
Payout (p): 8 DOT

Payout for each validator (v1 - v4):
p / v = 8 / 4 = 2 DOT

v1:
(0.2 * 2) = 0.4 DOT -> validator payment
(2 - 0.4) = 1.6 -> shared between all stake
(9 / 18) * 1.6 = 0.8 -> validator stake share
(9 / 18) * 1.6 = 0.8 -> nominator stake share
v1 validator total reward: 0.4 + 0.8 = 1.2 DOT
v1 nominator reward: 0.8 DOT

v2:
(0.4 * 2) = 0.8 DOT -> validator payment
(2 - 0.8) = 1.2 -> shared between all stake
(3 / 9) * 1.2 = 0.4 -> validator stake share
(6 / 9) * 1.2 = 0.8 -> nominator stake share
v2 validator total reward: 0.8 + 0.4 = 1.2 DOT
v2 nominator reward: 0.8 DOT

v3:
(0.1 * 2) = 0.2 DOT -> validator payment
(2 - 0.2) = 1.8 -> shared between all stake
(4 / 8) * 1.8 = 0.9 -> validator stake share
(4 / 8) * 1.8 = 0.9 -> nominator stake share
v3 validator total reward: 0.2 + 0.9 DOT = 1.1 DOT
v3 nominator reward: 0.9 DOT

v4:
(0 * 2) = 0 DOT -> validator payment
(2 - 0) = 2.0 -> shared between all stake
(1 / 6) * 2 = 0.33 -> validator stake share
(5 / 6) * 2 = 1.67 -> nominator stake share
v4 validator total reward: 0 + 0.33 DOT = 0.33 DOT
v4 nominator reward: 1.67 DOT
```

Using systemd for the Validator Node

You can run your validator as a systemd process so that it will automatically restart on server reboots or crashes (and helps to avoid getting slashed!).

Before following this guide you should have already set up your validator by following the How to validate article.

First create a new unit file called `polkadot-validator.service` in `/etc/systemd/system/`.

```
touch /etc/systemd/system/polkadot-validator.service
```

In this unit file you will write the commands that you want to run on server boot / restart.

```
[Unit]
Description=Polkadot Validator

[Service]
ExecStart=PATH_TO_POLKADOT_BIN --validator --name SHOW_ON_TELEMETRY
Restart=always
RestartSec=120

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

WARNING: It's recommended to delay the restart of a node with `RestartSec` in the case of node crashes. It's possible that when a node crashes, consensus votes in GRANDPA aren't persisted to disk. In this case, there is potential to equivocate when immediately restarting. What can happen is the node will not recognize votes that didn't make it to disk, and will then cast conflicting votes. Delaying the restart will allow the network to progress past potentially conflicting votes, at which point other nodes will not accept them.

To enable this to autostart on bootup run:

```
systemctl enable polkadot-validator.service
```

Start it manually with:

```
systemctl start polkadot-validator.service
```

You can check that it's working with:

```
systemctl status polkadot-validator.service
```

You can tail the logs with `journalctl` like so:

```
journalctl -f -u polkadot-validator
```

Secure Validator

Validators in a Proof of Stake network are responsible for keeping the network in consensus and verifying state transitions. As the number of validators is limited, validators in the set have the responsibility to be online and faithfully execute their tasks.

This primarily means that validators:

- Must have infrastructure that protects the validator's signing keys so that an attacker cannot take control and commit slashable behavior.
- Must be high availability.

High Availability

High availability set-ups that involve redundant validator nodes may seem attractive at first. However, they can be **very dangerous** if they are not set up perfectly. The reason for this is that the session keys used by a validator should always be isolated to just a single node. Replicating session keys across multiple nodes could lead to equivocation slashes, or soon to parachain validity slashes which can make you lose **100% of your staked funds**.

The good news is that 100% uptime of your validator is not really needed, as it has some buffer within eras in order to go offline for a little while and upgrade. For this reason, we advise that you only attempt a high availability set-up if you're confident you know exactly what you're doing. Many expert validators have made mistakes in the past due to the handling of session keys.

Remember, even if your validator goes offline for some time, the offline slash is much more forgiving than the equivocation or parachain validity slashing.

Key Management

See the Polkadot Keys guide for more information on keys. The keys that are of primary concern for validator infrastructure are the Session keys. These keys sign messages related to consensus and parachains. Although Session keys are *not* account keys and therefore cannot transfer funds, an attacker could use them to commit slashable behavior.

Session keys are generated inside the node via RPC call. See the Kusama guide for instructions on setting Session keys. These should be generated and kept within your client. When you generate new Session keys, you must submit an extrinsic (a Session certificate) from your Controller key telling the chain your new Session keys.

> **NOTE:** Session keys can also be generated outside the client and inserted into the client's keystore via RPC. For most users, we recommend using the key generation functionality within the client.

Signing Outside the Client

In the future, Polkadot will support signing payloads outside the client so that keys can be stored on another device, e.g. a hardware security module (HSM) or secure enclave. For the time being, however, Session key signatures are performed within the client.

> **NOTE:** HSMs are not a panacea. They do not incorporate any logic and will just sign and return whatever payload they receive. Therefore, an attacker who gains access to your validator node could still commit slashable behavior.

An example of highly available, secure setup would be a layer of sentry nodes in front of multiple validators connected to a single signing machine. This machine could implement signing logic to avoid equivocation, even if an attacker gained access to a validator node.

Monitoring Tools

- Telemetry This tracks your node details including the version you are running, block height, CPU & memory usage, block propagation time, etc.

- Prometheus-based monitoring stack, including Grafana for dashboards and log aggregation. It includes alerting, querying, visualization, and monitoring features and works for both cloud and on-premise systems. The data from `substrate-telemetry` can be made available to Prometheus through exporters like this.

Linux Best Practices

- Never use the root user.
- Always update the security patches for your OS.
- Enable and set up a firewall.
- Never allow password-based SSH, only use key-based access.
- Disable non-essential SSH subsystems (banner, motd, scp, X11 forwarding) and harden your SSH configuration (reasonable guide to begin with).
- Back up your storage regularly.

Conclusions

- At the moment, Polkadot/Substrate can't interact with HSM/SGX, so we need to provide the signing key seeds to the validator machine. This key is kept in memory for signing operations and persisted to disk (encrypted with a password).

- Given that HA setups would always be at risk of double-signing and there's currently no built-in mechanism to prevent it, we propose having a single instance of the validator to avoid slashing. Slashing penalties for being offline are much less than those for equivocation.

Validators

- Validators should only run the Polkadot binary, and they should not listen on any port other than the configured p2p port.

- Validators should run on bare-metal machines, as opposed to VMs. This will prevent some of the availability issues with cloud providers, along with potential attacks from other VMs on the same hardware. The provisioning of the validator machine should be automated and defined in code. This code should be kept in private version control, reviewed, audited, and tested.

- Session keys should be generated and provided in a secure way.

- Polkadot should be started at boot and restarted if stopped for any reason (supervisor process).

- Polkadot should run as non-root user.

Monitoring

- There should be an on-call rotation for managing the alerts.

- There should be a clear protocol with actions to perform for each level of each alert and an escalation policy.

Resources

- Figment Network's Full Disclosure of Cosmos Validator Infrastructure
- Certus One's Knowledge Base
- EOS Block Producer Security List
- Sentry Node Architecture Overview
- HSM Policies and the Important of Validator Security

How to use Polkadot Secure Validator Setup

The following guide will walk you through using polkadot secure validator to deploy your validator in a secure way. It will work for Kusama (and later Polkadot) out of the box, and if you're using another Substrate-based chain, should work with some tweaking. We assume you will be deploying on Kusama.

It uses Terraform for defining and managing your infrastructure. Ansible, an automation tool, is used for setting up the VPN, Firewall, and the validator node. It supports a few different cloud providers such as AWS, Microsoft Azure, GCP, and Packet. The code is publicly hosted on GitHub, so please file an issue if you would like to make a feature request or report a bug.

Dependencies

The next step is to install the software dependencies for running the secure validator scripts. We will need to acquire NodeJS, Yarn, Terraform, and Ansible. Usually these are readily available using your operating system's package manager. Instructions may vary depending on which system you are on, the instructions below demonstrate the commands for a user of a Debian or Ubuntu based system.

NodeJS

We recommend using nvm as a tool to manage different NodeJS versions across projects.

```
sudo apt-get install curl
curl -sL https://deb.nodesource.com/setup_13.x | sudo -E bash -
sudo apt-get install nodejs
node -v  (Check your node version)
```

Yarn

```
curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee /etc/apt/sources.list.d/yarn.list
sudo apt update
sudo apt install yarn
```

Terraform

```
sudo apt-get install unzip
wget https://releases.hashicorp.com/terraform/0.12.16/terraform_0.12.16_linux_amd64.zip
unzip terraform_0.12.16_linux_amd64.zip
sudo mv terraform /usr/local/bin/
terraform --version  (Check whether it is configured properly)
```

Ansible

```
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible -y
sudo apt-get install python -y
```

Deployment

Step One: Clone the repository

The first step is to clone the `polkadot-secure-validator` guide locally.

```
$ git clone git@github.com:w3f/polkadot-secure-validator.git
```

Now you can `cd` into the `polkadot-secure-validator` directory and start to change the configurations to match your custom deployment. However, before we start tweaking those, let's start by creating two new SSH keys that we (or rather, the ansible playbooks) will use to access the machines.

Step Two: Generate the SSH keys

We will use SSH, a remote shell tool, to access our validator and public sentry nodes. You will first use the `ssh-keygen` command to generate two keys, one for your validator and one for the sentry nodes.

```
$ ssh-keygen -m pem -f id_rsa_validator
$ ssh-keygen -m pem -f id_rsa_public
```

Be sure to add these keys to your SSH agent. First make sure your SSH agent is evaluated, then add the keys to them.

```
$ eval $(ssh-agent)
$ ssh-add id_rsa_validator
$ ssh-add id_rsa_public
```

For this tutorial we will not set a passphrase for the SSH key, although usually that would be recommended.

Configuration

After you have installed all the required software and made your ssh keys, you can start to configure your infrastructure deployment by following the instructions. Start by cloning the `polkadot-secure-validator` repository locally, and installing the package dependencies. Then customize the configuration how you want it.

First run yarn to install the NodeJS dependencies:

```
$ yarn
```

Now you can copy the configuration sample and start to cutomize it.

```
$ cp config/main.sample.json config/main.json
# now you should customize config/main.json
```

Under `validators` and `publicNodes`, specify which cloud provider you want to use, the type of machine specification, the number of validators you are going to deploy, the machine location, and the user to use for SSH.

Getting the authorization keys

The secure validator set up supports Google Cloud, AWS, Microsoft Azure, and Packet. For this tutorial we will be using Google Cloud.

Log in to the Google Cloud console

You will need to log in to the google cloud console in order to access your authorization keys.

In the IAM&Admin panel you will navigate to service accounts. Download JSON for service account key.

Make sure to also auth into your account like so:

```
$ gcloud auth login
```

And don't forget to enable the compute engine!

Configuration Options

The other options can be mostly self explanatory. Here's some tips on what they are and how you can use them:

In the `additionalFlags` option, configure any of the additional flags you want to run for your validator. If you want to run with a specific name, this is where you would enter it.

Under the `polkadotBinary.url` field you can provide the release that is hosted in the W3F repository or use an alternate one that you build and publish yourself.

By enabling the `nodeExporter`, Ansible will install and configure the node_exporter, which will expose hardware-level metrics of your node in a format compatible with Prometheus.

The field `machineType:` will configure the machine's hardware specifications, check here for the configuration options for GCP. The other hosting providers should have similar pages in their documentation.

Under `provider` the option are `gcp` (Google Cloud Provider), `aws` (AWS), `azure` (Microsoft Azure) and `packet` for Packet.

The field `count` is the number of instances you would like to create.

The `location` and `zone` fields are for the location of the machine, for GCP check here, other cloud providers will have similar documentation.

The `telemetryUrl` field will send your node's information to a specific telemetry server. You could send all your nodes' data (e.g. IP address) to the public endpoint, but it is highly recommended that that you set up your own telemetry server to protect your validator's data from being exposed to the public. If you want to do that, see substrate telemetry source.

> NOTE: If you decided to send your node's information to public telemetry, the name for your validator and public node that is displayed on the telemetry would look something like `PROJECT_NAME-sv-public-0` / `PROJECT_NAME-sv-validator-0`.

Configure `projectId` to be the name of the project you want to use in GCP.

Configure `sshUser` to be the user that manages your machine.

For different cloud providers, you need to set the corresponding credentials as environment variables, for example, on GCP you only need to set `GOOGLE_APPLICATION_CREDENTIALS`. This variable is the path to the JSON file containing the credentials of the service account you wish to use; this service account needs to have write access to compute and network resources if you use GCP. For others, you can check that by referring to the README.

Besides that, you need two additional environment variables that will allow Ansible to connect to the created machines. These values of these variables will be the keys that you generated at the beginning of the guide.

> `SSH_ID_RSA_PUBLIC` - Path to private SSH key you want to use for the public nodes.

> `SSH_ID_RSA_VALIDATOR` - Path to private SSH key you want to use for the validator.

> NOTE: You will need to configure the Compute Engine API and enable billing on your GCP accounts to properly run these scripts.

After everything is configured properly, you can start to run the deployment with:

```
$ scripts/deploy.sh
```

> NOTE: Certain steps of the process may hang, however the scripts are idempotent so you simply need to re-run them and

When the deployment and configuration is completed, you should see some output that looks like what's below. You are able to find the validator's session keys by searching for "show rotateKeys output".

```
TASK [polkadot-validator-session-info : retrieve session info] *****************

ok: [34.80.70.172]
```

```
PLAY RECAP *********************************************************************

34.80.224.231              : ok=41   changed=1    unreachable=0    failed=0    skipped=11   rescued=0
ignored=0

34.80.70.172               : ok=49   changed=1    unreachable=0    failed=0    skipped=14   rescued=0
ignored=0

35.189.183.66              : ok=41   changed=1    unreachable=0    failed=0    skipped=11   rescued=0
ignored=0

Done
Done in 131.85s.
```

Also you can use `sshUser` to access one of the created instances that shows above.

```
TASK [polkadot-validator : show rotateKeys output] *****************************

ok: [34.80.70.172] => {
    "rotate_keys": {
        "changed": false,
        "connection": "close",
        "content_length": "295",
        "content_type": "application/json; charset=utf-8",
        "cookies": {},
        "cookies_string": "",
        "date": "Sun, 24 Nov 2019 12:13:42 GMT",
        "elapsed": 0,
        "failed": false,
        "json": {
            "id": 1,
            "jsonrpc": "2.0",
            "result":
"0xf126b68841f51988b37780fa5b224b2aa86888a8d3962a63595dbc4d85baac2dee7c9900c8ddfad1991a8884e58273f06d5c1dbfc3dc60
00c037185ccead9d692a3b3396cdd7e2def520682d65ad7e8ca234fb17630b428752e6150462998b4362a2b7e201657c8084ae8215bd14245
8ccd69506d08b18925dc897fb95f54249"
        },
        "msg": "OK (295 bytes)",
        "redirected": false,
        "status": 200,
        "url": "http://localhost:9933"
    }
}
```

The result "0xf126b68841f5.....95f54249" is your session key. Set this to your controller account in polkadot-js Apps.

After accessing one of the machines through SSH, you can keep track of the node's status by running `journalctl --follow -u polkadot`, which will show the latest synced block information.

Every time you change something in `main.json`, you can simply run `./scripts/deploy.sh` to update it.

Congratulations! You have successfully deployed a secure validator. Free feel to open an issue if you have any suggestions.

Set Up a Sentry Node - Public Node

DEPRECATED

With a recent release of Substrate and Polkadot/Kusama, support for sentry nodes are deprecated. Details can be found here. Please stay tuned for further updates.

How to Upgrade Your Validator

Validators perform critical functions for the network, and as such, have strict uptime requirements. Validators may have to go offline for periods of time to upgrade the client software or the host machine. This guide will walk you through upgrading your machine and keeping your validator online.

The process will take several hours, so make sure you understand the instructions first and plan accordingly.

Key Components

Session Keys

Session keys are stored in the client and used to sign validator operations. They are what link your validator node to your Controller account. You cannot change them mid-Session.

More info about keys in Polkadot.

Database

Validators keep a database with all of their votes. If two machines have the same Session keys but different databases, they risk equivocating. For this reason, we will generate new Session keys each time we change machines.

More info about equivocation.

Steps

You will need to start a second validator to operate while you upgrade your primary. Throughout these steps, we will refer to the validator that you are upgrading as "Validator A" and the second one as "Validator B."

Session `N`

1. Start a second node and connect it to your sentry nodes. Once it is synced, use the `--validator` flag. This is "Validator B."
2. Generate Session keys in Validator B.
3. Submit a `set_key` extrinsic from your Controller account with your new Session keys.
4. Take note of the Session that this extrinsic was executed in.

It is imperative that your Validator A keep running in this Session. `set_key` only takes effect in the next Session.

Session `N+1`

Validator B is now acting as your validator. You can safely take Validator A offline. See note at bottom.

1. Stop Validator A.
2. Perform your system or client upgrade.
3. Start Validator A, sync the database, and connect it to your sentry nodes.
4. Generate new Session keys in Validator A.
5. Submit a `set_key` extrinsic from your Controller account with your new Session keys for Validator A.
6. Take note of the Session that this extrinsic was executed in.

Again, it is imperative that Validator B keep running until the next Session.

Once the Session changes, Validator A will take over. You can safely stop Validator B.

NOTE: To verify that the Session has changed, make sure that a block in the new Session is finalized. You should see log messages like this to indicate the change:

```
2019-10-28 21:44:13 Applying authority set change scheduled at block #450092
```

```
2019-10-28 21:44:13 Applying GRANDPA set change to new set with 20 authorities
```

Monitor your node

This guide will walk you through how to set up Prometheus with Grafana to monitor your node using Ubuntu 18.04 or 20.04.

A Substrate-based chain exposes data such as the height of the chain, the number of connected peers to your node, CPU, memory usage of your machine, and more. To monitor this data, Prometheus is used to collect metrics and Grafana allows for displaying them on the dashboard.

Preparation

First, create a user for Prometheus by adding the `--no-create-home` flag to disallow `prometheus` from logging in.

```
sudo useradd --no-create-home --shell /usr/sbin/nologin prometheus
```

Create the directories required to store the configuration and executable files.

```
sudo mkdir /etc/prometheus
sudo mkdir /var/lib/prometheus
```

Change the ownership of these directories to `prometheus` so that only prometheus can access them.

```
sudo chown -R prometheus:prometheus /etc/prometheus
sudo chown -R prometheus:prometheus /var/lib/prometheus
```

Installing and Configuring Prometheus

After setting up the environment, update your OS, and install the latest Prometheus. You can check the latest release by going to their GitHub repository under the releases page.

```
sudo apt-get update && apt-get upgrade
wget https://github.com/prometheus/prometheus/releases/download/v2.26.0/prometheus-2.26.0.linux-amd64.tar.gz
tar xfz prometheus-*.tar.gz
cd prometheus-2.26.0.linux-amd64
```

The following two binaries are in the directory:

- prometheus - Prometheus main binary file
- promtool

The following two directories (which contain the web interface, configuration files examples and the license) are in the directory:

- consoles
- console_libraries

Copy the executable files to the `/usr/local/bin/` directory.

```
sudo cp ./prometheus /usr/local/bin/
sudo cp ./promtool /usr/local/bin/
```

Change the ownership of these files to the `prometheus` user.

```
sudo chown prometheus:prometheus /usr/local/bin/prometheus
sudo chown prometheus:prometheus /usr/local/bin/promtool
```

Copy the `consoles` and `console_libraries` directories to `/etc/prometheus`

```
sudo cp -r ./consoles /etc/prometheus
sudo cp -r ./console_libraries /etc/prometheus
```

Change the ownership of these directories to the `prometheus` user.

```
sudo chown -R prometheus:prometheus /etc/prometheus/consoles
sudo chown -R prometheus:prometheus /etc/prometheus/console_libraries
```

Once everything is done, run this command to remove `prometheus` directory.

```
cd .. && rm -rf prometheus*
```

Before using Prometheus, it needs some configuration. Create a YAML configuration file named `prometheus.yml` by running the command below.

```
sudo nano /etc/prometheus/prometheus.yml
```

The configuration file is divided into three parts which are `global`, `rule_files`, and `scrape_configs`.

- `scrape_interval` defines how often Prometheus scrapes targets, while `evaluation_interval` controls how often the software will evaluate rules.
- `rule_files` block contains information of the location of any rules we want the Prometheus server to load.
- `scrape_configs` contains the information which resources Prometheus monitors.

The configuration file should look like this below:

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s

rule_files:
  # - "first.rules"
  # - "second.rules"

scrape_configs:
  - job_name: "prometheus"
    scrape_interval: 5s
    static_configs:
      - targets: ["localhost:9090"]
  - job_name: "substrate_node"
    scrape_interval: 5s
    static_configs:
      - targets: ["localhost:9615"]
```

With the above configuration file, the first exporter is the one that Prometheus exports to monitor itself. As we want to have more precise information about the state of the Prometheus server we reduced the `scrape_interval` to 5 seconds for this job. The parameters `static_configs` and `targets` determine where the exporters are running. The second exporter is capturing the data from your node, and the port by default is `9615`.

You can check the validity of this configuration file by running `promtool check config /etc/prometheus/prometheus.yml`.

Save the configuration file and change the ownership of the file to `prometheus` user.

```
sudo chown prometheus:prometheus /etc/prometheus/prometheus.yml
```

Starting Prometheus

To test that Prometheus is set up properly, execute the following command to start it as the `prometheus` user.

```
sudo -u prometheus /usr/local/bin/prometheus --config.file /etc/prometheus/prometheus.yml --storage.tsdb.path
/var/lib/prometheus/ --web.console.templates=/etc/prometheus/consoles --
web.console.libraries=/etc/prometheus/console_libraries
```

The following messages indicate the status of the server. If you see the following messages, your server is set up properly.

```
level=info ts=2021-04-16T19:02:20.167Z caller=main.go:380 msg="No time or size retention was set so using the
default time retention" duration=15d
level=info ts=2021-04-16T19:02:20.167Z caller=main.go:418 msg="Starting Prometheus" version="(version=2.26.0,
branch=HEAD, revision=3cafc58827d1ebd1a67749f88be4218f0bab3d8d)"
level=info ts=2021-04-16T19:02:20.167Z caller=main.go:423 build_context="(go=go1.16.2, user=root@a67cafebe6d0,
date=20210331-11:56:23)"
level=info ts=2021-04-16T19:02:20.167Z caller=main.go:424 host_details="(Linux 5.4.0-42-generic #46-Ubuntu SMP
Fri Jul 10 00:24:02 UTC 2020 x86_64 ubuntu2004 (none))"
level=info ts=2021-04-16T19:02:20.167Z caller=main.go:425 fd_limits="(soft=1024, hard=1048576)"
level=info ts=2021-04-16T19:02:20.167Z caller=main.go:426 vm_limits="(soft=unlimited, hard=unlimited)"
level=info ts=2021-04-16T19:02:20.169Z caller=web.go:540 component=web msg="Start listening for connections"
address=0.0.0.0:9090
level=info ts=2021-04-16T19:02:20.170Z caller=main.go:795 msg="Starting TSDB ..."
level=info ts=2021-04-16T19:02:20.171Z caller=tls_config.go:191 component=web msg="TLS is disabled." http2=false
level=info ts=2021-04-16T19:02:20.174Z caller=head.go:696 component=tsdb msg="Replaying on-disk memory mappable
chunks if any"
level=info ts=2021-04-16T19:02:20.175Z caller=head.go:710 component=tsdb msg="On-disk memory mappable chunks
replay completed" duration=1.391446ms
level=info ts=2021-04-16T19:02:20.175Z caller=head.go:716 component=tsdb msg="Replaying WAL, this may take a
while"
level=info ts=2021-04-16T19:02:20.178Z caller=head.go:768 component=tsdb msg="WAL segment loaded" segment=0
maxSegment=4
level=info ts=2021-04-16T19:02:20.193Z caller=head.go:768 component=tsdb msg="WAL segment loaded" segment=1
maxSegment=4
level=info ts=2021-04-16T19:02:20.221Z caller=head.go:768 component=tsdb msg="WAL segment loaded" segment=2
maxSegment=4
level=info ts=2021-04-16T19:02:20.224Z caller=head.go:768 component=tsdb msg="WAL segment loaded" segment=3
maxSegment=4
level=info ts=2021-04-16T19:02:20.229Z caller=head.go:768 component=tsdb msg="WAL segment loaded" segment=4
maxSegment=4
level=info ts=2021-04-16T19:02:20.229Z caller=head.go:773 component=tsdb msg="WAL replay completed"
checkpoint_replay_duration=43.716µs wal_replay_duration=53.973285ms total_replay_duration=55.445308ms
level=info ts=2021-04-16T19:02:20.233Z caller=main.go:815 fs_type=EXT4_SUPER_MAGIC
level=info ts=2021-04-16T19:02:20.233Z caller=main.go:818 msg="TSDB started"
level=info ts=2021-04-16T19:02:20.233Z caller=main.go:944 msg="Loading configuration file"
filename=/etc/prometheus/prometheus.yml
level=info ts=2021-04-16T19:02:20.234Z caller=main.go:975 msg="Completed loading of configuration file"
filename=/etc/prometheus/prometheus.yml totalDuration=824.115µs remote_storage=3.131µs web_handler=401ns
query_engine=1.056µs scrape=236.454µs scrape_sd=45.432µs notify=723ns notify_sd=2.61µs rules=956ns
level=info ts=2021-04-16T19:02:20.234Z caller=main.go:767 msg="Server is ready to receive web requests."
```

Go to `http://SERVER_IP_ADDRESS:9090/graph` to check whether you are able to access the Prometheus interface or not. If it is working, exit the process by pressing on `CTRL + C`.

Next, we would like to automatically start the server during the boot process, so we have to create a new `systemd` configuration file with the following config.

```
sudo nano /etc/systemd/system/prometheus.service
```

```
[Unit]
  Description=Prometheus Monitoring
  Wants=network-online.target
  After=network-online.target

[Service]
  User=prometheus
  Group=prometheus
  Type=simple
  ExecStart=/usr/local/bin/prometheus \
  --config.file /etc/prometheus/prometheus.yml \
  --storage.tsdb.path /var/lib/prometheus/ \
  --web.console.templates=/etc/prometheus/consoles \
  --web.console.libraries=/etc/prometheus/console_libraries
  ExecReload=/bin/kill -HUP $MAINPID

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

Once the file is saved, execute the command below to reload `systemd` and enable the service so that it will be loaded automatically during the operating system's startup.

```
sudo systemctl daemon-reload && systemctl enable prometheus && systemctl start prometheus
```

Prometheus should be running now, and you should be able to access its front again end by re-visiting `IP_ADDRESS:9090/`.

Installing Grafana

In order to visualize your node metrics, you can use Grafana to query the Prometheus server. Run the following commands to install it first.

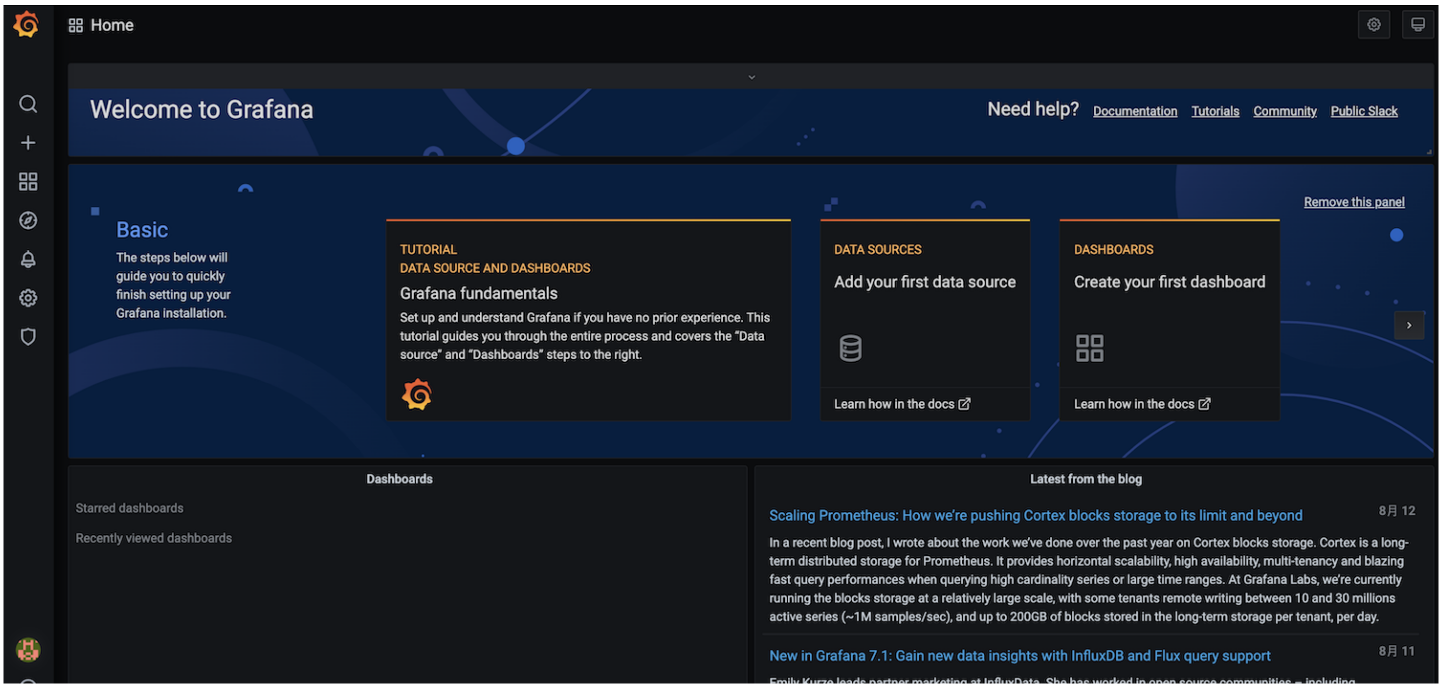
```
sudo apt-get install -y adduser libfontconfig1
wget https://dl.grafana.com/oss/release/grafana_7.5.4_amd64.deb
sudo dpkg -i grafana_7.5.4_amd64.deb
```

If everything is fine, configure Grafana to auto-start on boot and then start the service.

```
sudo systemctl daemon-reload
sudo systemctl enable grafana-server
sudo systemctl start grafana-server
```

You can now access it by going to the `http://SERVER_IP_ADDRESS:3000/login`. The default user and password is admin/admin.

> Note: If you want to change the port on which Grafana runs (3000 is a popular port), edit the file `/usr/share/grafana/conf/defaults.ini` with a command like `sudo vim /usr/share/grafana/conf/defaults.ini` and change the `http_port` value to something else. Then restart grafana with `sudo systemctl restart grafana-server`.



The steps below will guide you to quickly finish setting up your Grafana installation.

Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.

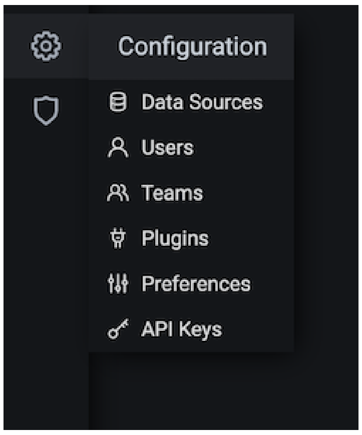
Scaling Prometheus: How we're pushing Cortex blocks storage to its limit and beyond 8月 12

In a recent blog post, I wrote about the work we've done over the past year on Cortex blocks storage. Cortex is a long-term distributed storage for Prometheus. It provides horizontal scalability, high availability, multi-tenancy and blazing fast query performances when querying high cardinality series or large time ranges. At Grafana Labs, we're currently running the blocks storage at a relatively large scale, with some tenants remote writing between 10 and 30 millions active series (~1M samples/sec), and up to 200GB of blocks stored in the long-term storage per tenant, per day.

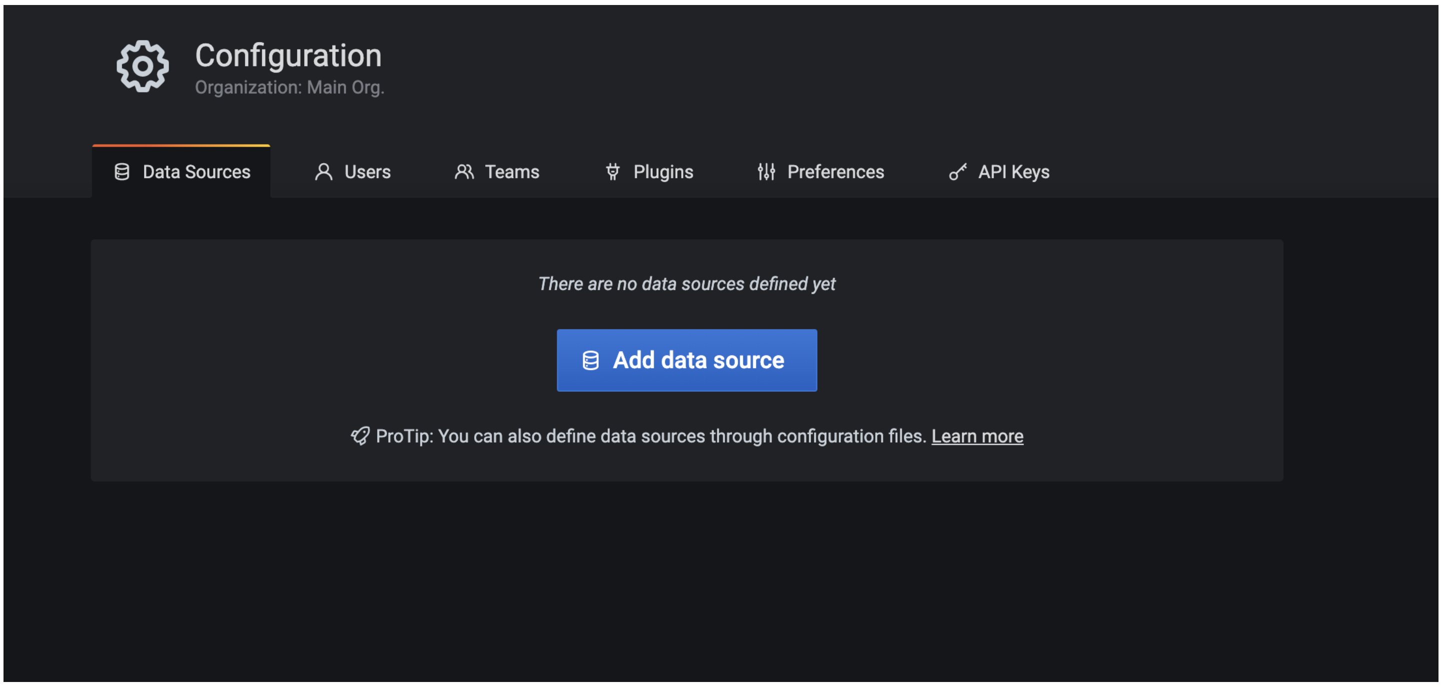
New in Grafana 7.1: Gain new data insights with InfluxDB and Flux query support 8月 11

Emily Kurze leads partner marketing at InfluxData. She has worked in open source communities – including

In order to visualize the node metrics, click *settings* to configure the `Data Sources` first.



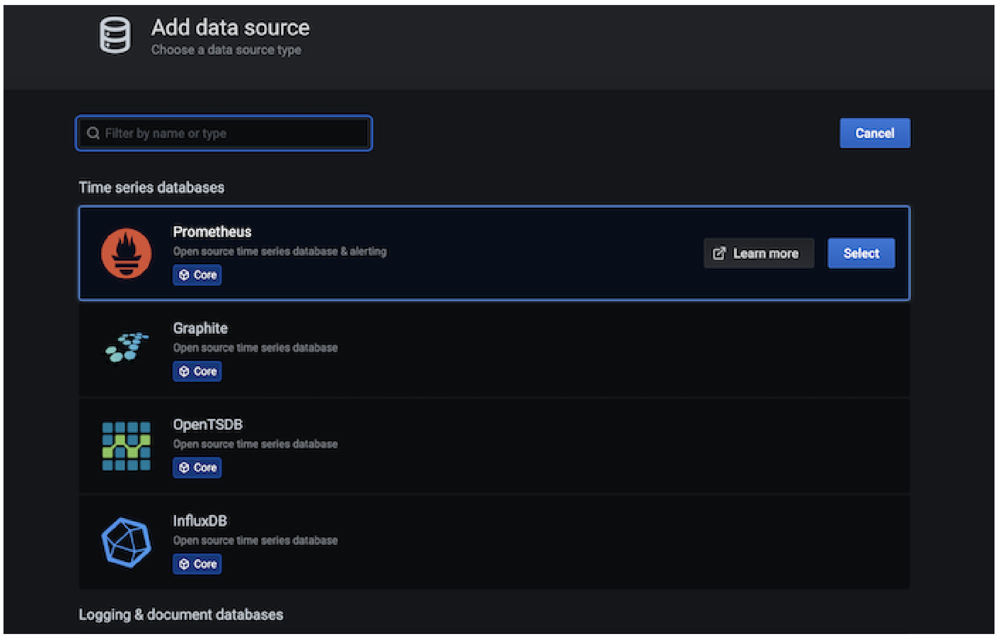
Click `Add data source` to choose where the data is coming from.



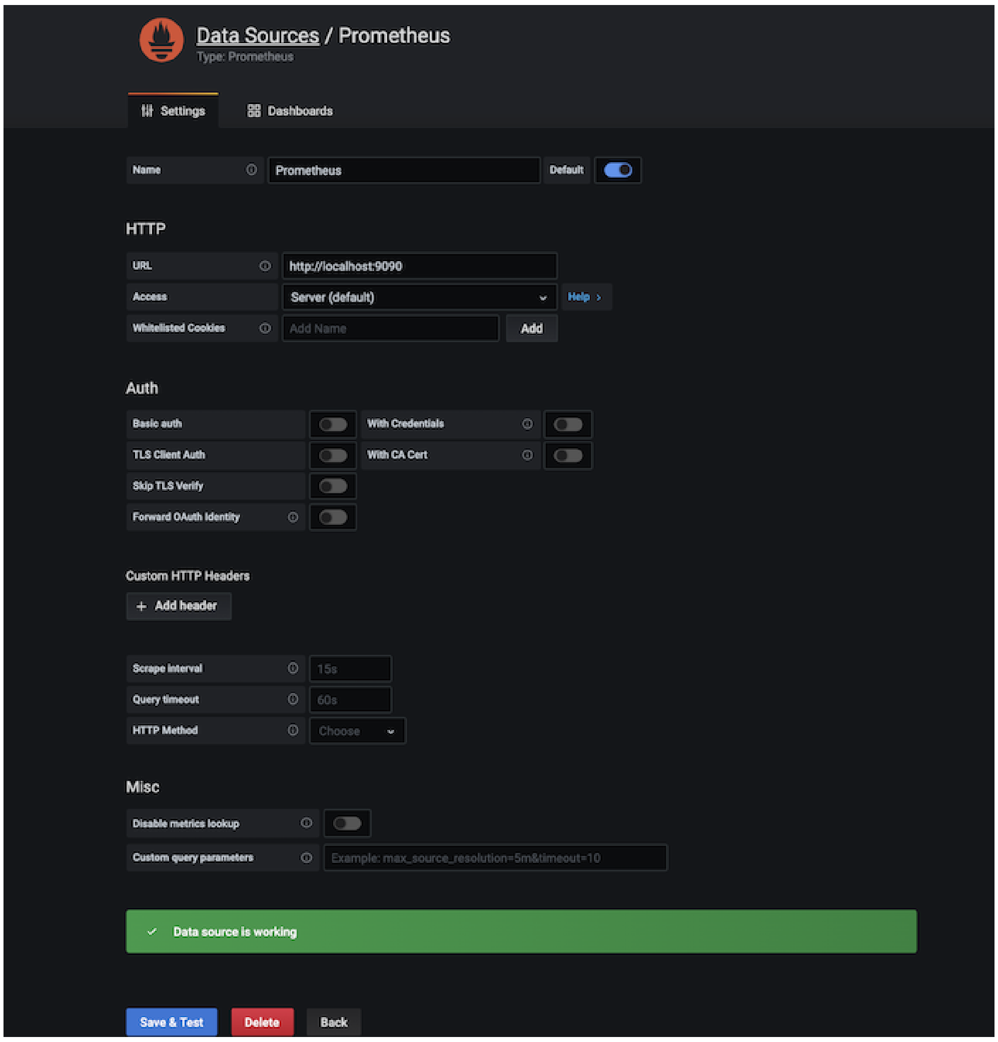
There are no data sources defined yet

ProTip: You can also define data sources through configuration files. Learn more

Select `Prometheus`.


Add data source
Choose a data source type
Filter by name or type
Cancel
Time series databases
Prometheus
Open source time series database & alerting
Core
Learn more
Select
Graphite
Open source time series database
Core
OpenTSDB
Open source time series database
Core
InfluxDB
Open source time series database
Core
Logging & document databases

The only thing you need to input is the `URL` that is `https://localhost:9090` and then click `Save & Test`. If you see `Data source is working`, your connection is configured correctly.



Data Sources / Prometheus
Type: Prometheus

Settings | Dashboards

Name: Prometheus | Default: on

HTTP

URL: http://localhost:9090
Access: Server (default) | Help ›
Whitelisted Cookies: Add Name | Add

Auth

Basic auth: off | With Credentials: off
TLS Client Auth: off | With CA Cert: off
Skip TLS Verify: off
Forward OAuth Identity: off

Custom HTTP Headers
+ Add header

Scrape interval: 15s
Query timeout: 60s
HTTP Method: Choose

Misc

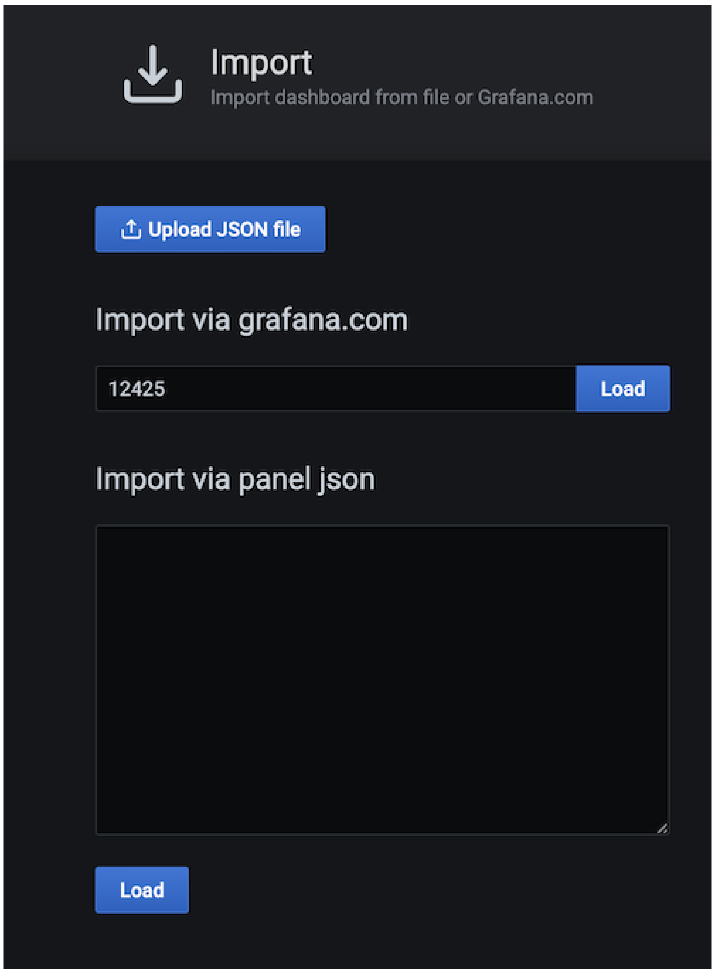
Disable metrics lookup: off
Custom query parameters: Example: max_source_resolution=5m&timeout=10

✓ Data source is working

Save & Test | Delete | Back

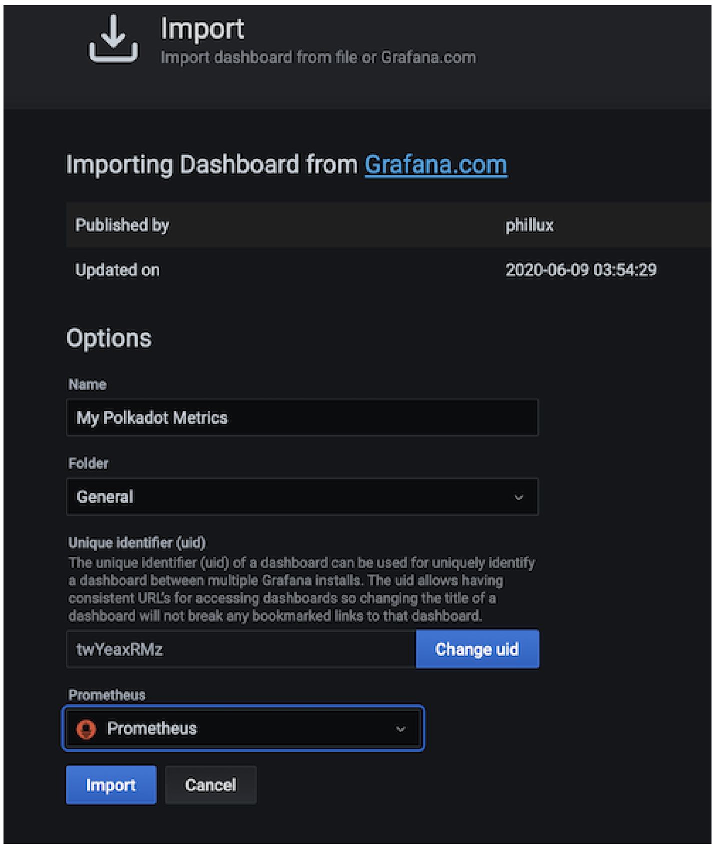
Next, import the dashboard that lets you visualize your node data. Go to the menu bar on the left and mouse hover "+" then select `Import`.

`Import via grafana.com` - It allows you to use a dashboard that someone else has created and made public. You can check what other dashboards are available via https://grafana.com/grafana/dashboards. In this guide, we use "My Polkadot Metrics", so input "12425" under the id field and click `Load`.



12425

Once it has been loaded, make sure to select "Prometheus" in the Prometheus dropdown list. Then click `Import`.



Importing Dashboard from Grafana.com

Published by	phillux
Updated on	2020-06-09 03:54:29

Options

Name: My Polkadot Metrics

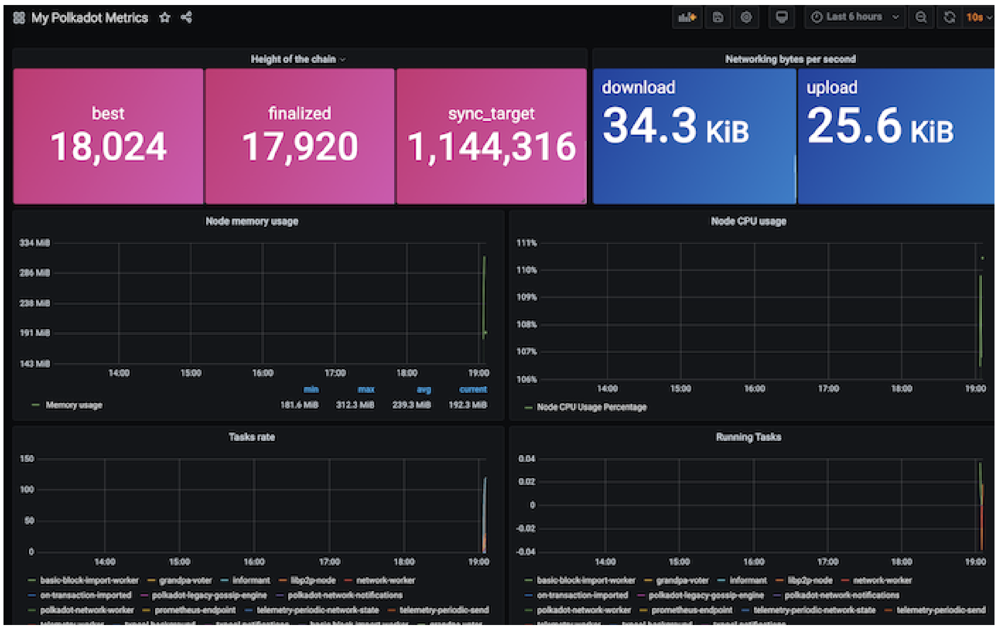
Folder: General

Unique identifier (uid)
The unique identifier (uid) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The uid allows having consistent URL's for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

twYeaxRMz

Prometheus: Prometheus

In the meantime, start your Polkadot node by running `./polkadot`. If everything is done correctly, you should be able to monitor your node's performance such as the current block height, CPU, memory usage, etc. on the Grafana dashboard.



Installing and Configuring Alertmanager (Optional)

In this section, let's configure the Alertmanager that helps to predict the potential problem or notify you of the current problem in your server. Alerts can be sent in Slack, Email, Matrix, or others. In this guide, we will show you how to configure the email notifications using Gmail if your node goes down.

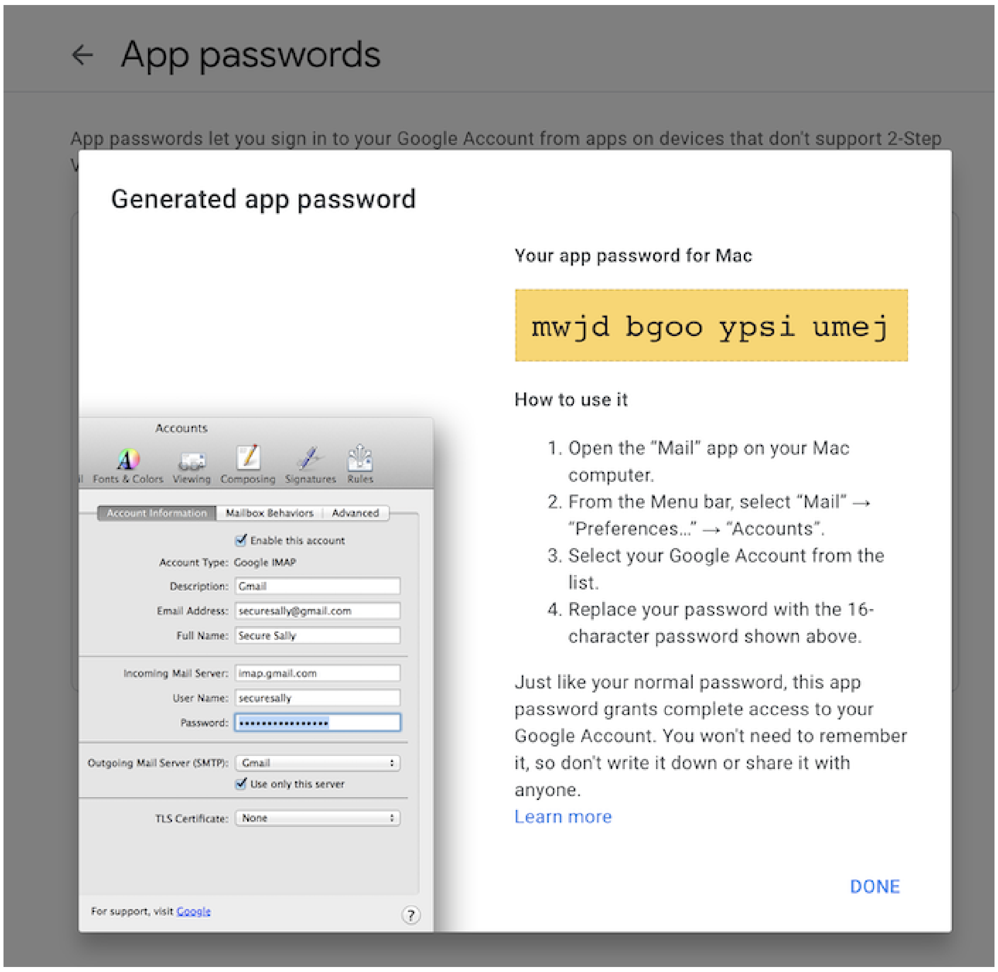
First, download the latest binary of AlertManager and unzip it by running the command below:

```
wget https://github.com/prometheus/alertmanager/releases/download/v0.21.0/alertmanager-0.21.0.linux-amd64.tar.gz
tar -xvzf alertmanager-0.21.0.linux-amd64.tar.gz
mv alertmanager-0.21.0.linux-amd64.tar.gz/alertmanager /usr/local/bin/
```

Gmail Setup

To allow AlertManager to send an email to you, you will need to generate something called an `app password` in your Gmail account. For details, click here to follow the whole setup.

You should see something like below:


App passwords

App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step

Generated app password

Your app password for Mac

mwjd bgoo ypsi umej

How to use it

1. Open the "Mail" app on your Mac computer.
2. From the Menu bar, select "Mail" → "Preferences..." → "Accounts".
3. Select your Google Account from the list.
4. Replace your password with the 16-character password shown above.

Just like your normal password, this app password grants complete access to your Google Account. You won't need to remember it, so don't write it down or share it with anyone.
Learn more

Accounts

Account Information | Mailbox Behaviors | Advanced

[✓] Enable this account
Account Type: Google IMAP
Description: Gmail
Email Address: securesally@gmail.com
Full Name: Secure Sally
Incoming Mail Server: imap.gmail.com
User Name: securesally
Password: ••••••••••••••••
Outgoing Mail Server (SMTP): Gmail
[✓] Use only this server
TLS Certificate: None

For support, visit Google

Copy and save it somewhere else first.

AlertManager Configuration

There is a configuration file named `alertmanager.yml` inside the directory that you just extracted in the previous command, but that is not of our use. We will create our `alertmanager.yml` file under `/etc/alertmanager` with the following config.

> Ensure to change the ownership of "/etc/alertmanager" to `prometheus` by executing
>
> sudo chown -R prometheus:prometheus /etc/alertmanager

```
global:
 resolve_timeout: 1m

route:
 receiver: 'gmail-notifications'

receivers:
- name: 'gmail-notifications'
  email_configs:
  - to: YOUR_EMAIL
    from: YOUR_EMAIL
    smarthost: smtp.gmail.com:587
    auth_username: YOUR_EMAIL
    auth_identity: YOUR_EMAIL
    auth_password: YOUR_APP_PASSWORD
    send_resolved: true
```

With the above configuration, alerts will be sent using the the email you set above. Remember to change `YOUR_EMAIL` to your email and paste the app password you just saved earlier to the `YOUR_APP_PASSWORD`.

Next, create another `systemd` configuration file named `alertmanager.service` by running the command `sudo nano /etc/systemd/system/alertmanager.service` with the following config.

SERVER_IP - Change to your host IP address amd make sure port 9093 is opened

```
[Unit]
Description=AlertManager Server Service
Wants=network-online.target
After=network-online.target

[Service]
User=root
Group=root
Type=simple
ExecStart=/usr/local/bin/alertmanager --config.file /etc/alertmanager/alertmanager.yml --web.external-
url=http://SERVER_IP:9093 --cluster.advertise-address='0.0.0.0:9093'


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

To the start the Alertmanager, run the following commands:

```
sudo systemctl daemon-reload && sudo systemctl enable alertmanager && sudo systemctl start alertmanager && sudo
systemctl status alertmanager
```

```
● alertmanager.service - AlertManager Server Service
   Loaded: loaded (/etc/systemd/system/alertmanager.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2020-08-20 22:01:21 CEST; 3 days ago
 Main PID: 20592 (alertmanager)
    Tasks: 70 (limit: 9830)
   CGroup: /system.slice/alertmanager.service
```

You should see the process status is "active (running)" if you have configured properly.

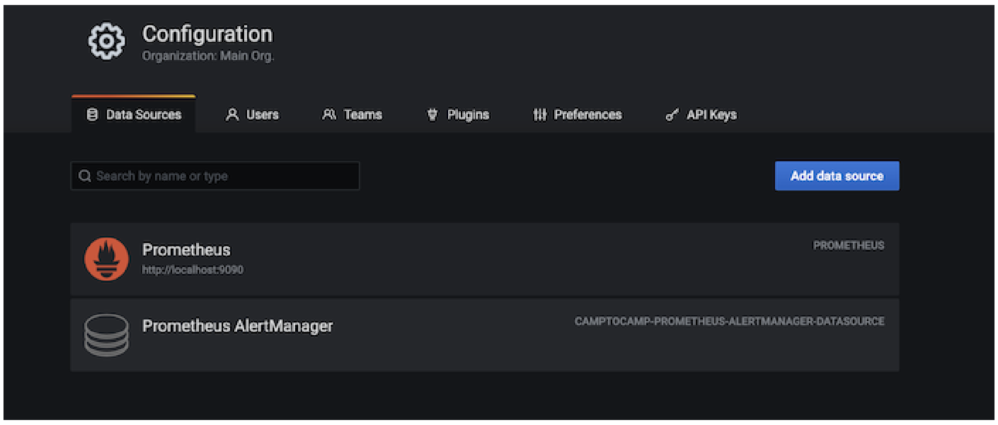
There is a Alertmanager plugin in Grafana that can help you to monitor the alert information. To install it, execute the command below:

```
sudo grafana-cli plugins install camptocamp-prometheus-alertmanager-datasource
```

And restart Grafana once the plugin is successfully installed.

```
sudo systemctl restart grafana-server
```

Now go to your Grafana dashboard `SERVER_IP:3000` and configure the Alertmanager datasource.



Go to Configuration -> Data Sources, search "Prometheus AlertManger" if you cannot find it at the top.

Fill in the `URL` to your server location followed by the port number used in the Alertmanager.

Then click "Save & Test" at the bottom to test the connection.

To monitor the alerts, let's import dashboard "8010" that is used for Alertmanager. And make sure to select the "Prometheus AlertManager" in the last column. Then click "Import".

You will end up having the follwing:

AlertManager Integration

To let the Prometheus server be able to talk to the Alertmanger, we will need to add the following config in the `etc/prometheus/prometheus.yml`.

```
rule_files:
  - 'rules.yml'

alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - localhost:9093
```

That is the updated `etc/prometheus/prometheus.yml`.

```
global:
  scrape_interval:     15s
  evaluation_interval: 15s

rule_files:
  - 'rules.yml'

alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - localhost:9093

scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'substrate_node'
    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:9615']
```

We will need to create a new file called "rules.yml" under `/etc/prometheus/` that is defined all the rules we would like to detect. If any of the rules defined in this file is fulfilled, an alert will be triggered. The rule below checks whether the instance is down. If it is down for more than 5 minutes, an email notification will be sent. If you would like to learn more about the details of the rule defining, go here. There are other interesting alerts you may find useful here.

```
groups:
  - name: alert_rules
    rules:
      - alert: InstanceDown
        expr: up == 0
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "Instance [{{ $labels.instance }}] down"
          description: "[{{ $labels.instance }}] of job [{{ $labels.job }}] has been down for more than 1
minute."
```

Change the ownership of this file to `prometheus` instead of `root` by running:

```
sudo chown prometheus:prometheus rules.yml
```

To check the rules defined in the "rules.yml" is syntactically correct, run the following command:

```
sudo –u prometheus promtool check rules rules.yml
```

Finally, restart everthing by running:

```
sudo systemctl restart prometheus && sudo systemctl restart alertmanager
```

Now if one of your target instances down, you will receive an alert on the AlertManager and Gmail like below.

How to Chill

Stakers can be in any one of the three states: validating, nominating, or chilling. When a staker wants to temporarily pause their active engagement in staking, but does not want to unbond their funds, they can choose to "chill" their involvement and keep their funds staked.

An account can step back from participating in active staking by clicking "Stop" under the `Network > Staking > Account actions` page in PolkadotJS Apps or by calling the `chill` extrinsic in the staking pallet. When an account chooses to chill, they will become inactive in the next era. The call must be signed by the *controller* account, not the *stash*.

> Note: If you need a refresher on the different responsibilities of the stash and controller account when staking, take a look at the accounts section in the general staking guide.

Chilling as a Nominator

When you chill after being a nominator, your nominations will be reset. This means that when you decide to start nominating again you will need to select validators to nominate once again. These can be the same validators if you prefer, or a completely new set. Just be aware - your nominations will not persist across chills.

Your nominator will remain bonded when it is chilled. When you are ready to nominate again, you will not need to go through the whole process of bonding again, rather you will issue a new nominate call that specifies the new targets to nominate.

Chilling as a Validator

When you voluntarily chill after being a validator, your nominators will not automatically go away. As long as your nominators make no action, you will still have the nominations when you choose to become an active validator once again. However, if your nominators decide to nominate other validators then these nominations will take priority when the validator comes back. It may also be the case that your nominators change their entire nomination targets (all 16 of the allowed nominations). In this case your nominators would need to explicitly specify your validator as a target when your validator comes back.

When you become an active validator you will also need to reset your validator preferences (commission, etc.). These can be configured as the same values that were set previously or something totally different.

Involuntary Chills

If a validator was unresponsive or found to have committed a slashable offense within two eras, the validator will be removed from the active set in a process known as *involuntary chilling.* When a validator has been involuntarily chilled, it is necessary for the nominators that were previously nominating that validator to re-issue the nominate call.

Nominators who have the option to renominate an involuntarily chilled validator will have a display row to do so using Polkadot-JS Apps. This row is displayed in the "Account Actions" tab for the nominator under a heading that says "Redenomination required". If your validator has been involuntarily chilled, you will need to request your nominators to re-issue the nominate call in order to start nominating you again.

How to Stop Validating

If you wish to remain a validator or nominator (e.g. you're only stopping for planned downtime or server maintenance), submitting the `chill` extrinsic in the `staking` pallet should suffice. It is only if you wish to unbond funds or reap an account that you should continue with the following.

To ensure a smooth stop to validation, make sure you should do the following actions:

- Chill your validator
- Purge validator session keys
- Unbond your tokens

These can all be done with PolkadotJS Apps interface or with extrinsics.

Chill Validator

To chill your validator or nominator, call the `staking.chill()` extrinsic. See the How to Chill page for more information. You can also claim your rewards at this time.

Purge validator session keys

Purging the validator's session keys removes the key reference to your stash. This can be done through the `session.purgeKeys()` extrinsic with the controller account.

> NOTE: **If you skip this step, you will not be able to reap your stash account**, and you will need to rebond, purge the session keys, unbond, and wait the unbonding period again before being able to transfer your tokens. See Unbonding and Rebonding for more details.

Unbond your tokens

Unbonding your tokens can be done through the `Network > Staking > Account actions` page in PolkadotJS Apps by clicking the corrosponding stash account dropdown and selecting "Unbond funds". This can also be done through the `staking.unbond()` extrinsic with the controller account.

Participate in Democracy

The public referenda chamber is one of the three bodies of on-chain governance as it's instantiated in Polkadot and Kusama. The other two bodies are the council and the technical committee.

Public referenda can be proposed and voted on by any token holder in the system as long as they provide a bond. After a proposal is made, others can agree with it by *seconding* it and putting up tokens equal to the original bond. Every launch period, the most seconded proposal will be moved to the public referenda table where it can be voted upon. Voters who are willing to lock up their tokens for a greater duration of time can do so and get their vote amplified. For more details on the governance system please see here.

This guide will instruct token holders how to propose and vote on public referenda using the Democracy module as it's implemented in Kusama.

Important Parameters

The important parameters to be aware of when voting using the Democracy module are as follow:

Launch Period - How often new public referenda are launched.

Voting Period - How often votes for referenda are tallied.

Emergency Voting Period - The minimum voting period for a fast-tracked emergency referendum.

Minimum Deposit - The minimum amount to be used as a deposit for a public referendum proposal.

Enactment Period - The minimum period for locking funds *and* the period between a proposal being approved and enacted.

Cooloff Period - The period in blocks where a proposal may not be re-submitted after being vetoed.

Proposing an Action

Proposing an action to be taken requires you to bond some tokens. In order to ensure you have enough tokens to make the minimum deposit you can check the parameter in the chain state. The bonded tokens will only be released once the proposal is tabled (that is, brought to a vote); there is no way for the user to "revoke" their proposal and get the bond back before it has become a referendum. Since it is essentially impossible to predict definitely when a proposal may become a referendum (if ever), this means that any tokens bonded will be locked for an indeterminate amount of time.

> Proposals cannot be revoked by the proposer, even if they never turn into a referendum. It is important to realize that there is no guarantee that DOT you use for proposing or seconding a proposal will be returned to that account in any given timeframe.

On Polkadot Apps you can use the "Democracy" tab to make a new proposal. In order to submit a proposal, you will need to submit what's called the preimage hash. The preimage hash is simply the hash of the proposal to be enacted. The easiest way to get the preimage hash is by clicking on the "Submit preimage" button and configuring the action that you are proposing.

For example, if you wanted to propose that the account "Dave" would have a balance of 10 tokens your proposal may look something like the below image. The preimage hash would be `0xa50af1fadfca818feea213762d14cd198404d5496bca691294ec724be9d2a4c0`. You can copy this preimage hash and save it for the next step. There is no need to click Submit Preimage at this point, though you could. We'll go over that in the next section.

Now you will click on the "Submit proposal" button and enter the preimage hash in the input titled "preimage hash" and *at least* the minimum deposit into the "locked balance" field. Click on the blue "Submit proposal" button and confirm the transaction. You should now see your proposal appear in the "proposals" column on the page.

Now your proposal is visible by anyone who accesses the chain and others can second it or submit a preimage. However, it's hard to tell what exactly this proposal does since it shows the hash of the action. Other holders will not be able to make a judgement for whether they second it or not until someone submits the actual preimage for this proposal. In the next step you will submit the preimage.

Submitting a Preimage

The act of making a proposal is split from submitting the preimage for the proposal since the storage cost of submitting a large preimage could be pretty expensive. Allowing for the preimage submission to come as a separate transaction means that another account could submit the preimage for you if you don't have the funds to do so. It also means that you don't have to pay so many funds right away as you can prove the preimage hash out-of-band.

However, at some point before the proposal passes you will need to submit the preimage or else the proposal cannot be enacted. The guide will now show you how to do this.

Click on the blue "Submit preimage" button and configure it to be the same as what you did before to acquire the preimage hash. This time, instead of copying the hash to another tab, you will follow through and click "Submit preimage" and confirm the transaction.

Once the transaction is included you should see the UI update with the information for your already submitted proposal.

Seconding a Proposal

Seconding a proposal means that you are agreeing with the proposal and backing it with an equal amount of deposit as was originally locked. The bonded tokens will be released once the proposal is tabled (that is, brought to a vote), just like the original proposer's bond. By seconding a proposal you will move it higher up the rank of proposals. The most seconded proposal — in value, not number of supporters — will be brought to a referendum every launch period.

It is important to note that there is no way to stop or cancel seconding a proposal once it has been done. Therefore, the DOT that was seconded will be reserved until the proposal is tabled as a referendum. This is an indeterminate amount of time, since there is no guarantee that a proposal will become a referendum for a given period, as other proposals may be proposed and tabled before it.

Note that it is possible for a single account to second a proposal multiple times. This is by design; it is the value, not the number of seconds *per se*, that counts in terms of weighting. If there were a limit of one second per account, it would be trivial for a user with, for example, 1000 DOT to create ten accounts with 100 DOT instead of a single account with 1000 DOT. Thus, no restrictions are made on the number of times a single account can second a proposal.

To second a proposal, navigate to the proposal you want to second and click on the "Second" button.

You will be prompted with the full details of the proposal (if the preimage has been submitted!) and can then broadcast the transaction by clicking the blue "Second" button.

Once successful you will see your second appear in the dropdown in the proposal details.

Voting on a Proposal

At the end of each launch period, the most seconded proposal will move to referendum. During this time you can cast a vote for or against the proposal. You may also lock up your tokens for a greater length of time to weigh your vote more strongly. During the time your tokens are locked, you are unable to transfer them, however they can still be used for further votes. Locks are layered on top of each other, so an eight week lock does not become a 15 week lock if you vote again a week later, rather another eight week lock is placed to extend the lock just one extra week.

To vote on a referendum, navigate to the "Democracy" tab of Polkadot Apps. Any active referendum will show in the "referenda" column. Click the blue button "Vote" to cast a vote for the referendum.

If you would like to cast your vote for the proposal select the "Aye, I approve" option. If you would like to cast your vote against the proposal in referendum you will select "Nay, I do not approve" option.

The second option is to select your conviction for this vote. The longer you are willing to lock your tokens, the stronger your vote will be weighted. The timeline for the conviction starts after the voting period ends; tokens used for voting will always be locked until the end of the voting period, no matter what conviction you vote with. Unwillingness to lock your tokens means that your vote only counts for 10% of the tokens that you hold, while the maximum lock up of 896 days means you can make your vote count for 600% of the tokens that you hold.

When you are comfortable with the decision you have made, click the blue "Vote" button to submit your transaction and wait for it to be included in a block.

Unlocking Locked Tokens

Like vesting, the tokens that are locked in democracy are unlocked lazily. This means that you, the user, must explicitly call an unlock extrinsic to make your funds available again after the lock expires. Unbonding is another term you hear a lot in Polkadot, it means withdrawing your DOT that was used in staking. To know more about it, please see here.

You can do this from the "Accounts" page in Polkadot-JS Apps, unless you use Ledger (see below). First check that your account has a "democracy" lock by opening the details on your balance. In the example below the account has 150 KSM locked in democracy.

Now you can click the menu button on Apps and find the option that says "Clear expired democracy locks". After selecting this option you may confirm the transaction and your locks will be cleared when successful.

With a Ledger hardware wallet or Unlocking Very Old Locks

If you do not see an option to clear expired democracy votes, it may be that the lock is very old. Or, if you are using the Ledger hardware wallet, you will not be able to issue the batch Unlock action from the UI.

Instead, you must clear the lock by directly issuing the correct extrinsics.

Navigate to the Extrinsics page and submit the following extrinsic: `democracy.removeVote(index)` using the account that you voted with. For the index number (ReferendumIndex), enter the number of the referendum for which you voted ("12" in the image below).

The number of the referendum for which you voted is visible in an explorer such as Polkascan.

You need to press the "Submit Transaction" button to submit the extrinsic.

Now submit the following extrinsic: `democracy.unlock(target)`, where target is your your account address.

If you return to the Accounts page, you should see that the democracy lock has been released.

Note that this applies only to locked DOT that were used for voting on referenda. In order to unlock DOT locked by voting for members of the Polkadot Council, you need to go to the Council page, click "Vote", and then click on "Unvote All".

Delegate a Vote

If you are too busy to keep up and vote on upcoming referenda, there is an option to delegate your vote to another account whose opinion you trust. When you delegate to another account, that account gets the added voting power of your tokens along with the conviction that you set. The conviction for delegation works just like the conviction for regular voting, except your tokens may be locked longer than they would normally since locking resets when you undelegate your vote.

The account that is being delegated to does not make any special action once the delegation is in place. They can continue to vote on referenda how they see fit. The difference is now when the Democracy system tallies votes, the delegated tokens now are added to whatever vote the delegatee has made.

You can delegate your vote to another account and even attach a "Conviction" to the delegation. Navigate to the "Extrinsics" tab on Polkadot Apps and select the options "democracy" and "delegate". This means you are accessing the democracy pallet and choosing the delegate transaction type to send. Your delegation will count toward whatever the account you delegated for votes on until you explicitly undelegate your vote.

In the first input select the account you want to delegate to and in the second input select the amount of your conviction. Remember, higher convictions means that your vote will be locked longer. So choose wisely!

After you send the delegate transaction, you can verify it went through by navigating to the "Chain State" tab and selecting the "democracy" and "delegations" options. You will see an output similar to below, showing the addresses to which you have delegated your voting power.

Undelegate a Vote

You may decide at some point in the future to remove your delegation to a target account. In this case, your tokens will be locked for the maximum amount of time in accordance with the conviction you set at the beginning of the delegation. For example, if you chose "2x" delegation for four weeks lock up time, your tokens will be locked for 4 weeks after sending the `undelegate` transaction. Once your vote has been undelegated, you are in control of making votes with it once again. You can start to vote directly, or chose a different account to act as your delegate.

The `undelegate` transaction must be sent from the account that you wish to clear of its delegation. For example, if Alice has delegated her tokens to Bob, Alice would need to be the one to call the `undelegate` transaction to clear her delegation.

The easiest way to do this is from the "Extrinsics" tab of Polkadot Apps. Select the "democracy" pallet and the "undelegate" transaction type. Ensure that you are sending the transaction from the account you want to clear of delegations. Click "Submit transaction" and confirm.

Voting with a Governance Proxy

Making a vote on behalf of a stash requires a "proxy" transaction from the Proxy pallet. When you choose this transaction from the "Extrinsics" tab, it will let you select "vote" from the Democracy pallet, and you will specify the index of the referendum that is being voted, the judgement (i.e. "Aye" for approval or "Nay" for rejection), and the conviction, just like a normal vote.

For more material on adding and removing Governance proxies, as well as other types, please see the Proxy page.

Interpreting On-Chain Voting Data

Consider the following example showcasing how votes would be displayed on a block explorer.

```
Nay 0.1x => 0
Nay 1x => 1
Nay 2x => 2
Nay 3x => 3
Nay 4x => 4
Nay 5x => 5
Nay 6x => 6
Aye 0.1x => 128
Aye 1x => 129
Aye 2x => 130
Aye 3x => 131
Aye 4x => 132
Aye 5x => 133
Aye 6x => 134
```

At first glance, it may be difficult to interpret what you voted on. We need to take a step back and consider the "voting data" at the binary level.

The vote is stored as a byte using a bitfield data structure and displayed on the block explorer as a decimal integer. The bitfield stores both the conviction and aye/nay boolean, where the boolean is represented using the MSB of the byte. This would mean that the grouping of the 7 remaining bits is used to store the conviction.

Join the Council

The council is an elected body of on-chain accounts that are intended to represent the passive stakeholders of Polkadot and/or Kusama. The council has two major tasks in governance: proposing referenda and vetoing dangerous or malicious referenda. For more information on the council, see the governance page. This guide will walk you through entering your candidacy to the council.

Submit Candidacy

Submitting your candidacy for the council requires a small bond of DOT / KSM. Unless your candidacy wins, the bond will be forfeited. You can receive your bond back if you manually renounce your candidacy before losing. Runners-up are selected after every round and are reserved members in case one of the winners gets forcefully removed.

> Currently the bond for submitting a council candidacy on Polkadot is 100 DOT, and 0.1666 KSM on Kusama.

It is a good idea to announce your council intention before submitting your candidacy so that your supporters will know when they can start to vote for you. You can also vote for yourself in case no one else does.

Go to Polkadot Apps Dashboard and navigate to the "Council" tab. Click the button on the right that says "Submit Candidacy."

After making the transaction, you will see your account appear underneath the row "Candidates."

It is a good idea now to lead by example and give yourself a vote.

Voting on Candidates

Next to the button to submit candidacy is another button titled "Vote." You will click this button to make a vote for yourself (optional).

The council uses Phragmén approval voting, which is also used in the validator elections. This means that you can choose up to 16 distinct candidates to vote for and your stake will equalize between them. For this guide, choose to approve your own candidacy by clicking on the switch next to your account and changing it to say "Aye."

Winning

If you are one of the lucky ones to win a council election you will see your account move underneath the row "Members".

Now you are able to participate on the council by making motions or voting proposals. To join in on the active discussions, join the Polkadot Direction channel.

Voting for Councillors

The council is an elected body of on-chain accounts that are intended to represent the passive stakeholders of Polkadot and/or Kusama. The council has two major tasks in governance: proposing referenda and vetoing dangerous or malicious referenda. For more information on the council, see the governance page. This guide will walk you through voting for councillors in the elections.

Voting for Councillors

Voting for councillors requires you to lock 5 DOT on Polkadot or 0.0083 KSM on Kusama for the duration of your vote.

> Warning: If your balance is vesting, you cannot use unvested tokens for this lock. You will have to wait until you have at least that many **free** tokens to vote.

Like the validator elections, you can approve up to 16 different councillors and your vote will be equalized among the chosen group. Unlike validator elections, there is no unbonding period for your reserved tokens. Once you remove your vote, your tokens will be liquid again.

> Warning: It is your responsibility not to put your entire balance into the reserved value when you make a vote for councillors. It's best to keep *at least* enough DOT/KSM to pay for transaction fees.

Go to the Polkadot-JS Apps Dashboard and click on the "Council" tab. On the right side of the window there are two blue buttons, click on the one that says "Vote."

Since the council uses approval voting, when you vote you signal which of the candidates you approve of and your voted tokens will be equalized among the selected candidates. Select up to 16 council candidates by moving the slider to "Aye" for each one that you want to be elected. When you've made the proper configuration submit your transaction.

You should see your vote appear in the interface immediately after your transaction is included.

Removing your Vote

In order to get your reserved tokens back, you will need to remove your vote. Only remove your vote when you're done participating in elections and you no longer want your reserved tokens to count for the councillors that you approve.

Go to the "Governance" > "Council" tab on the Polkadot-JS Apps Dashboard.

Under the "Council overview" tab, click on "Vote".

Issue the "Unvote all" option.

When the transaction is included in a block you should have your reserved tokens made liquid again and your vote will no longer be counting for any councillors in the elections starting in the next term.