Learn

Overview

Mandala Chain is a hybrid layer 1 blockchain project that addresses the need for secure and confidential data storage and transfer in the face of data breaches and cyberattacks. It uses the Substrate framework to create a customizable and modular architecture that provides users confidentiality, anonymity, and data protection.

Web 2 Integration with Web 3 Technology

Experience the best of both worlds with Mandala Chain. Our unique Web2 Layer Integration seamlessly bridges the gap between traditional web applications and the innovative world of Web3. Enjoy the familiarity and convenience of Web2 while harnessing the power of Web3 technology.

Trust in a decentralized world

At Mandala Chain, we believe in trust and decentralization. Our platform empowers individuals through decentralized identity verification, allowing users to confirm their identity without compromising privacy. With Mandala Chain, you can confidently verify your identity while ensuring the highest level of security.

The project will leverage zero-knowledge proofs (ZKP) technology to enable private transactions and verification without compromising privacy - creating a trustless, secure, private data storage and transfer environment. Mandala Chain is a solution to the problem of data privacy and security, offering secure and private data storage and transfer in the blockchain ecosystem. ZKP feature will be implemented in the near future when it reaches the adequate level of maturity for Substrate implementation. Stay tuned for our future updates.

KEPENG Coin

Kepeng Coin (Ticker: $KPG) is a cryptocurrency used for governance, gas fees, and node development in the Mandala Chain ecosystem. It is inspired by a traditional Balinese coin called Kepeng, which has cultural and spiritual significance and is still used in Balinese villages for governance and offerings. Kepeng Coin aims to honor this tradition by incorporating it into the Mandala Chain ecosystem to incentivize users to contribute to the network's growth and development. Kepeng Coin will serve as a governance token, allowing holders to participate in network decisions, and as a means of payment for transaction fees on the network.

Architecture and Components

A blockchain relies on a decentralized network of computers—called **nodes**—that communicate with each other.

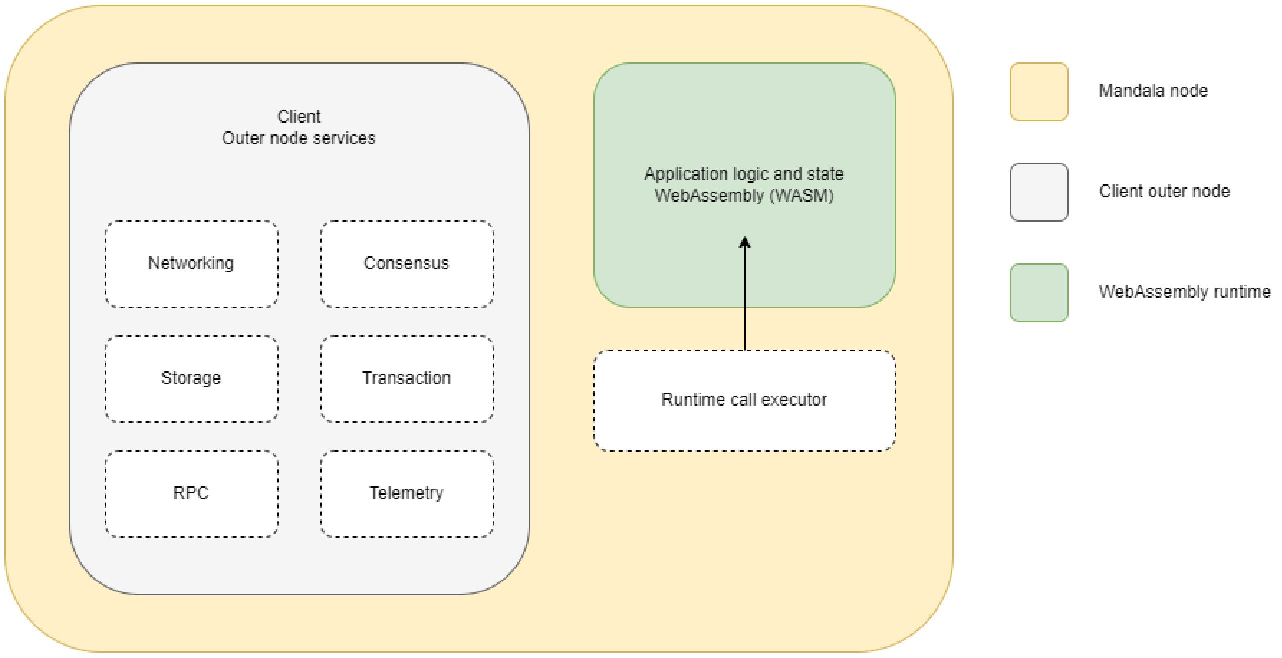
Because the node is a core component of any blockchain, it's essential to understand what makes a Substrate-based blockchain node unique. This includes the core services and libraries provided by default and how the node can be customized and extended to suit our goals.

Client and runtime

At a high level, a Mandala Chain node consists of two main parts:

- A **core client** with **outer node services** that handles network activity such as peer discovery, managing transaction requests, reaching consensus with peers and responding to RPCs (Remote Procedure Calls).
- A **runtime** that contains all of the business logic for executing the state transition function of the blockchain.

The following diagram illustrates this separation of responsibilities in simplified form to help you visualize the architecture of the Mandala Chain.



Client and Runtime

Client outer node services

The core client includes several outer node services responsible for the activity outside the runtime. For example, the outer node service in the core client handles peer discovery, manages the transaction pool, communicates with other nodes to reach consensus, and responds to RPC requests from the outside world.

Some of the most important activities that are handled by core client services involve the following components:

- **Storage**: The outer node persists in the evolving state of a Mandala Chain blockchain using a simple and highly efficient key-value storage layer. Note: a key-value storage system is a database where data is stored in pairs – a key (an identifier) and a value (the actual data).
- **Peer-to-peer networking**: The outer node uses the Rust implementation of the `libp2p` network stack to communicate with other network participants.
- **Consensus**: The outer node communicates with other network participants to ensure

they agree on the state of the blockchain.

- **Remote Procedure Call (RPC) API**: The outer node accepts inbound HTTP and WebSocket requests to allow blockchain users to interact with the network.
- **Telemetry**: The outer node collects and provides access to node metrics through an embedded Prometheus server.
- **Execution environment**: The outer node is responsible for selecting the execution environment—WebAssembly or native Rust—for the runtime to use and then dispatching calls to the runtime selected.

Mandala Chain provides default implementations for handling these activities through its core blockchain components. In principle, you can modify or replace the default implementation of any component with your own code. In practice, it's rare for an application to require changes to any of the underlying blockchain features, but Substrate allows you to make changes so you are free to innovate where you see fit.

Performing these tasks often requires the client node services to communicate with the runtime. This communication is handled by calling specialized **runtime API**s.

Runtime

The runtime determines whether transactions are valid or invalid and handles changes to the blockchain state. Requests coming from the outside come through the client into the runtime, and the runtime is responsible for the state transition functions and storing the resulting state.

Because the runtime executes the functions it receives, it controls how transactions are included in blocks and how blocks are returned to the outer node for gossiping or importing to other nodes. In essence, the runtime is responsible for handling everything that happens on-chain. It is also the core component of the node for building the Mandala Chain.

The Mandala Chain runtime is designed to compile to WebAssembly (WASM) byte code.

This design decision enables:

- Support for forkless upgrades.
- Multi-platform compatibility.
- Runtime validity checking.
- Validation proofs for relay chain consensus mechanisms.

Similar to how the outer node has a way to provide information to the runtime, the runtime uses specialized host functions to communicate with the outer node or the outside world.

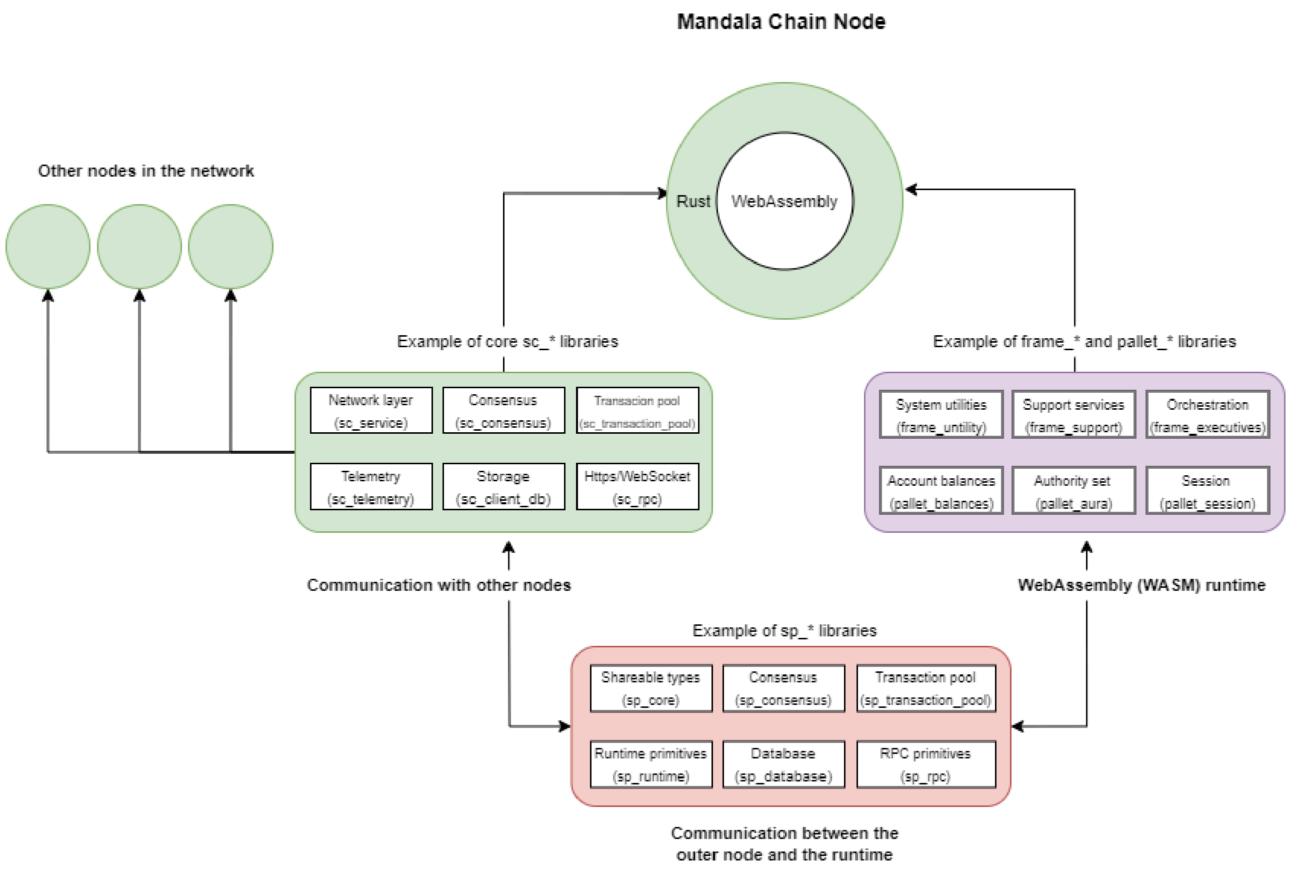
Core libraries

Many aspects of the blockchain are configured with a default implementation. For example, there are default implementations of the networking layer, database, and consensus mechanism that you can use as-is to get your blockchain running without a lot of customization. However, the libraries underlying the basic architecture provide a great deal of flexibility for defining your own blockchain components.

Much like the node consists of two main parts—the core client and the runtime—that provide different services, Substrate libraries are divided into three main areas of responsibility:

- Core client libraries for outer node services.
- FRAME libraries for the runtime.
- Primitive libraries for underlying functions and interfaces for communication between the libraries.

The following diagram illustrates how the libraries mirror the core client outer node and runtime responsibilities and how the library of **primitives** provides the communication layer between the two.


Mandala Chain Node
Rust
WebAssembly
Other nodes in the network
Example of core sc_* libraries
Network layer (sc_service)
Consensus (sc_consensus)
Transacion pool (sc_transaction_pool)
Telemetry (sc_telemetry)
Storage (sc_client_db)
Https/WebSocket (sc_rpc)
Example of frame_* and pallet_* libraries
System utilities (frame_untility)
Support services (frame_support)
Orchestration (frame_executives)
Account balances (pallet_balances)
Authority set (pallet_aura)
Session (pallet_session)
Communication with other nodes
WebAssembly (WASM) runtime
Example of sp_* libraries
Shareable types (sp_core)
Consensus (sp_consensus)
Transaction pool (sp_transaction_pool)
Runtime primitives (sp_runtime)
Database (sp_database)
RPC primitives (sp_rpc)
Communication between the outer node and the runtime

Core client libraries

The libraries that enable the blockchain node to handle its network responsibilities, including consensus and block execution, are Rust crates that use the `sc_` prefix in the crate name. For example, the `sc_service` library is responsible for building the networking layer for blockchains and managing the communication between the network participants and the transaction pool.

FRAME libraries for the runtime

The libraries that enable you to build the runtime logic and to encode and decode the information passed into and out of the runtime are Rust crates that use the `frame_` prefix in the crate name.

The `frame_*` libraries provide the infrastructure for the runtime. For example, the `frame_system` library provides a basic set of functions for interacting with other Substrate-based components and `frame_support` enables you to declare runtime storage items, errors, and events.

In addition to the infrastructure provided by the `frame_*` libraries, the runtime can include one or more `pallet_*` libraries. Each Rust crate that uses the `pallet_` prefix represents a single FRAME module. In most cases, you use the `pallet_*` libraries to assemble the functionality you want to incorporate in the blockchain to suit your project.

You can build the runtime without using the `frame_*` or `pallet_*` libraries using the **primitives** libraries. However, the `frame_*` or `pallet_*` libraries provide the most efficient path to composing the runtime.

Primitive libraries

At the lowest level of the Mandala Chain architecture, there are primitive libraries that give you control over underlying operations and enable communication between the core client services and the runtime. The primitive libraries are Rust crates that use the `sp_` prefix in the crate name.

The primitive libraries provide the lowest level of abstraction to expose interfaces that the core client or the runtime can use to perform operations or interact with each other.

For example:

- The `sp_arithmetic` library defines fixed point arithmetic primitives and types for the runtime to use.
- The `sp_core` library provides a set of shareable Substrate types.
- The `sp_std` library exports primitives from the Rust standard library to make them usable with any code that depends on the runtime.

Modular architecture

The separation of the core Mandala Chain libraries provides a flexible and modular architecture for writing the blockchain logic. The primitives library provides a foundation that both the core client and the runtime can build without communicating directly with each other. Primitive types and traits are exposed in their own separate crates, so they are available to the outer node services and runtime components without introducing cyclic dependency issues.

Ecosystem

Mandala Chain Layer 1 Networks

Mandala network (Mainnet):

- Mandala is the main network that acquires features after they are established and stable on the Niskala testnet.
- The functionality will be equal to the Niskala testnet, with a focus on the proven security and stability that have been battle-tested on the Niskala testnet.

Niskala Test network (Testnet):

- Niskala is a value-bearing test network that gets features before Mandala does.
- Functionality is equal to the future Mandala mainnet, with possible next-generation testing of features from time to time that will eventually migrate onto Mandala Mainnet.

Ecosystem Projects

For a project that wants to utilize Mandala Chain, please refer to this BUILD documentation.

Consensus Mechanism

Introduction

In the blockchain ecosystem, consensus mechanisms are crucial to maintaining the integrity and security of the network. A consensus mechanism is a process used by a network of nodes to agree upon a single data value or a shared state of the network. In other words, it ensures that all nodes in the blockchain have a unified view of the data.

Mandala Chain uses a hybrid consensus mechanism that bridges efficiency and security, critical for maintaining the integrity of the network.

Mandala Chain leverages **Aura (Authority Round)** for block production. This mechanism relies on a select group of collators who are responsible for producing new blocks. By using a **Proof-of-Authority (PoA)** setup, the process ensures that only authorized entities generate blocks, streamlining and securing the network operations.

For block finality, Mandala Chain utilizes **GRANDPA (GHOST-based Recursive ANcestor Deriving Prefix Agreement)**. This highly efficient protocol ensures that once a block is finalized, it becomes immutable and secure against changes, maintaining the stability of the ledger across the network.

Block Authors

There are validators on the Mandala Chain who participate in the consensus mechanism to produce the blocks based on validity statements from other validators. These validators are called ***block authors;*** they are selected by **AURA consensus** and can note up to one backable candidate for each block to include in the blockchain. A backable candidate included in the chain is considered backed in that fork of the chain.

In a Mandala Chain block, block authors will only include candidate receipts that have a parent candidate receipt in an earlier block. This ensures the Mandala Chain blocks follow a valid chain. Also, the block authors will only include a receipt for which they have an erasure coding chunk, ensuring that the system can perform the next round of availability and validity checks.

Block Production

In the Mandala Chain, block production is a critical process managed by **collators**. These key participants are responsible for assembling transactions and proposing new block candidates, ensuring the smooth operation of the blockchain network.

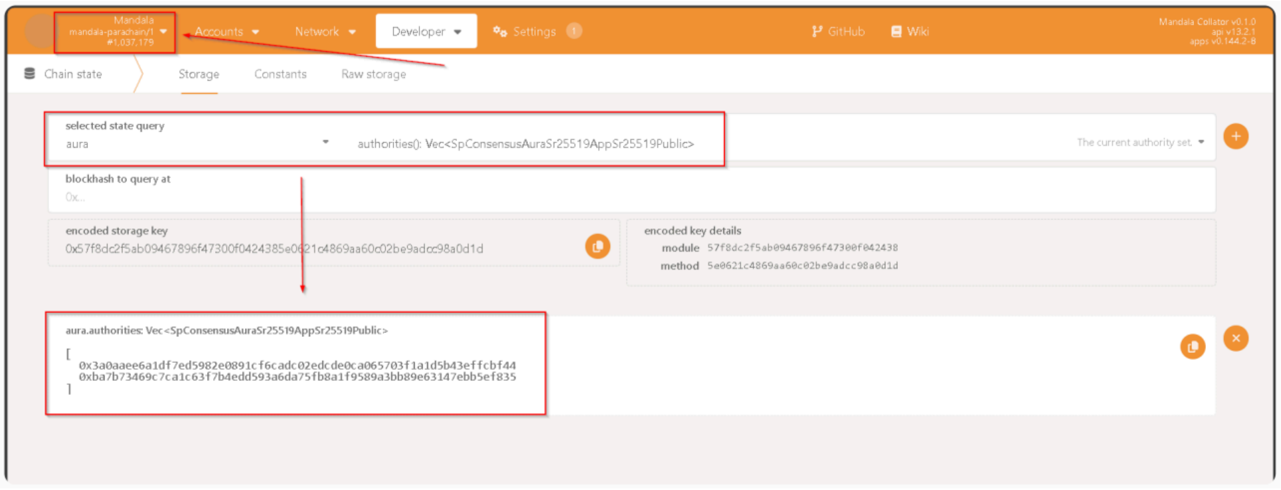
Key Aspects of Block Production

1. **Independent Operation**:
    - Block production functions separately from the finality mechanism. This separation allows for efficient and rapid generation of blocks without delays, contributing to the overall performance of the parachain.
2. **Consensus Mechanism**:
    - The Mandala Chain uses the **Aura (Authority Round)** pallet for producing blocks, based on the **Proof-of-Authority (PoA)** model. This model simplifies block production by using a fixed set of trusted collators to author blocks, ensuring security and efficiency.
3. **Deterministic Control**:
    - Aura's design enables a centralized (but trusted) group of collators to create blocks in a scheduled manner. This deterministic approach minimizes conflicts and helps predict network behavior, enhancing reliability.

Verification of Aura Implementation

To maintain the integrity and functionality of Aura in Mandala Chain, we follow a systematic verification process:

- **Aura Pallet Verification**: It's essential to ensure that the Aura pallet is present within the chain's runtime. This verification confirms that the pallet is properly integrated and actively managing block production.
- **Aura Keys Check**: The accuracy of Aura keys setup is crucial. We ensure that these keys are correctly configured within the Aura pallet, allowing authorized collators to carry out block production effectively.
- **ASTAR Reference**: We benchmark our process against ASTAR, which also utilizes Aura. Their successful methodology in maintaining Aura's integrity helps validate our approach and provides insights into optimizing our setup.



selected state query
aura
authorities(): Vec<SpConsensusAuraSr25519AppSr25519Public>
The current authority set.

blockhash to query at
0x...

encoded storage key
0x57f8dc2f5ab09467896f47300f0424385e0621c4869aa60c02be9adcc98a0d1d

encoded key details
module 57f8dc2f5ab09467896f47300f042438
method 5e0621c4869aa60c02be9adcc98a0d1d

```
aura.authorities: Vec<SpConsensusAuraSr25519AppSr25519Public>
[
  0x3a0aaee6a1df7ed5982e0891cf6cadc02edcde0ca065703f1a1d5b43effcbf44
  0xba7b73469c7ca1c63f7b4edd593a6da75fb8a1f9589a3bb89e63147ebb5ef835
]
```

When opening our Mandala Node in production, everything is running Aura as expected.

Block Finality

Block finality is a crucial aspect of the Mandala Chain, ensuring that once a block is validated, it becomes immutable and secure. This integrity is crucial for maintaining trust across the network.

How Block Finality Works

1. **Finality Mechanism**:
    - After collators create block candidates, these blocks are not immediately

considered finalized. Instead, validators from the relay chain use the **GRANDPA (GHOST-based Recursive ANcestor Deriving Prefix Agreement)** protocol to finalize them.

2. **Role of GRANDPA**:
   - GRANDPA is a finality protocol used across Polkadot parachains, providing a robust and reliable way to confirm block permanence. It ensures that even if forks occur, only the canonical chain is recognized and extends, eliminating conflicts and duplications.
3. **Nominated Proof-of-Stake (NPoS) System**:
   - Validators are selected via the NPoS system. In this system, nominators stake DOT tokens to endorse validators they trust.
   - Trusted validators are then responsible for executing the GRANDPA protocol, confirming the security and finality of blocks. This consensus process helps maintain the network's overall integrity.
4. **Fork Resolution**:
   - Forks can happen when multiple blocks reference the same parent block. The **fork-choice rule** is applied to decide which chain is legit.
   - In the case of GRANDPA, the longest chain rule is implemented, where the "best" chain is determined by its length. Validators vote to finalize this chain, making it the official record.
5. **Benefits of GRANDPA**:
   - It provides immediate finality once enough votes are collected. This means users can trust that the data on the blockchain won't be altered, enhancing security and trustworthiness.
   - It efficiently resolves forks, ensuring that only one valid chain continues, simplifying the processing and validation tasks across the network.

Through these mechanisms, block finality in the Mandala Chain is robustly managed, ensuring that transactions are secure, irreversible, and reliably recorded in the network’s ledger. This setup not only enhances trust but also optimizes the performance and scalability of the entire system.

Non-Permissionless Setup

Maintaining a **non-permissionless setup** is essential for ensuring the security and integrity of the Mandala Chain. This framework guarantees that only authorized validators can participate in block production, which is vital for preventing malicious activities and preserving trust in the network.

1. **Validator Selection Control**:

   - The framework includes a **collator selection pallet** that is intentionally configured to prevent **new candidate registrations**. By setting the number of new candidates to zero, we eliminate the possibility of unauthorized entities joining the network as collators. This ensures that only pre-approved, trustworthy participants can produce blocks, enhancing overall network security.

2. **Managing Collator Additions**:

   - New collator additions are carefully controlled through the **invulnerable pallet**. This mechanism allows us to add only those collators deemed "invulnerable," meaning they have passed rigorous trust and reliability assessments. By restricting collator entries to only those with demonstrated resilience and integrity, we mitigate risks associated with introducing potentially harmful actors into the ecosystem.

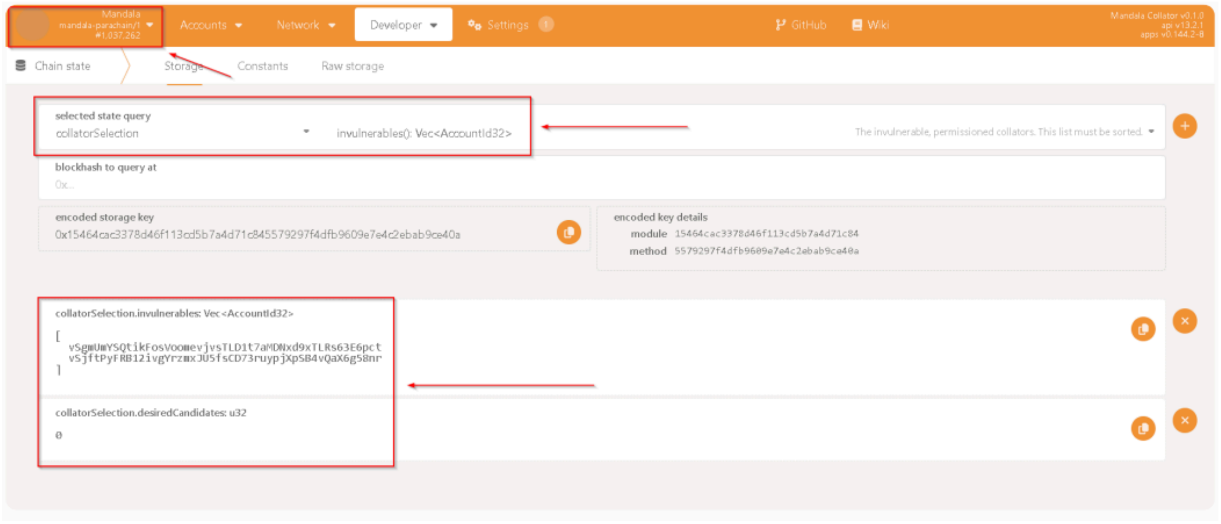
3. **Verification Process**:

   - To validate the effectiveness of this setup, the Mandala team conducts regular testing by deploying ephemeral Mandala node instances on the **zombienet**. This testing involves attempting to add new collators to observe if the configuration holds firm.
   - If attempts to introduce new collators fail, it confirms that the system is secure and functioning as intended. This proactive verification is crucial in maintaining a

strong posture against unauthorized access and reinforces confidence in our operational framework.

4. **Current Status**:

   - At present, the production environment has successfully configured candidate slots to zero, effectively halting any unauthorized additions. This configuration ensures that the validator set remains secure and reliable, maintaining the integrity of the Mandala Chain.
   - By consistently monitoring and updating this setup, we can adapt to emerging threats while safeguarding the network's foundational principles.



selected state query
collatorSelection invulnerables(): Vec<AccountId32>
The invulnerable, permissioned collators. This list must be sorted.

blockhash to query at
0x...

encoded storage key
0x15464cac3378d46f113cd5b7a4d71c845579297f4dfb9609e7e4c2ebab9ce40a

encoded key details
module 15464cac3378d46f113cd5b7a4d71c84
method 5579297f4dfb9609e7e4c2ebab9ce40a

collatorSelection.invulnerables: Vec<AccountId32>

```
[
  vSgmUmYSQtikFosVoomevjvsTLD1t7aMDNxd9xTLRs63E6pct
  vSjftPyFRB12ivgYrzmxJU5fsCD73ruypjXpSB4vQaX6g58nr
]
```

collatorSelection.desiredCandidates: u32

```
0
```

How the current state of Mandala Chain's collator pallet looks like.

Glossary

Active Nomination

A validator (or validators) that a nominator has selected to nominate and is actively validating this era. The nominator is placing their stake behind this validator for this era and will potentially receive staking rewards in return for doing so.

Attestation

In the Mandala Chain validity system, an *attestation* is a type of message that validators broadcast that says whether they think a block candidate is valid or invalid.

Authority

An authority is a generic term for the role in a blockchain that can participate in the consensus mechanisms. In GRANDPA, the authorities vote on chains they consider final. In BABE, the authorities are block producers. Authority sets can be chosen to be mechanisms such as Mandala Chain's PoA algorithm.

BABE

Blind Assignment for Blockchain Extension (BABE) is Mandala Chain's block production mechanism.

Bitfield Array

A bitfield array contains single-bit values which indicate whether a candidate is available. The number of items is equal of to the number of availability cores and each bit represents a vote on the corresponding core in the given order.

Block

A collection of data, such as transactions, that together indicate a state transition of the blockchain.

Blockspace

Blockspace is the capacity of a blockchain to finalize and commit operations. It represents a blockchain's security, computing, and storage capability as an end product. Blockspace produced by different blockchains can vary in quality, availability, and flexibility. Mandala Chain has a blockspace-centric architecture.

Block Explorer

An application that allows a user to explore the different blocks on a blockchain.

Blocks Nominations

This indicates that a validator does not currently allow any more nominations. This is controlled by the validator.

BLS

Boneh-Lynn-Shacham (BLS) signatures have a slow signing, very slow verification, require slow and much less secure pairing friendly curves, and tend towards dangerous malleability. Yet, BLS permits a diverse array of signature aggregation options far beyond any other known signature scheme, which makes BLS a preferred scheme for voting in consensus algorithms and threshold signatures.

Bounty

A mechanism that works in some sense as the reverse of a Treasury Proposal, allowing the Mandala Chain Council to indicate that there is a need to do some task for the Mandala Chain network and allowing users to receive KPG in return for working on that task.

Byzantine Fault Tolerance

The property of a system that is tolerant of Byzantine faults; a system where not only may individual subsystems fail, but it may not be clear if a particular subsystem has failed or not. That is, different observers of the system may not agree on whether or not the system has failed. Ensuring Byzantine fault tolerance is an important part of developing any distributed system.

Capacity

The maximum number of nominators signaling intent to nominate a validator (and thus could potentially actively nominate that validator in the next session). This maximum number will equal the number of nominators necessary to oversubscribe a validator. Any validator that is "at capacity" or higher may potentially be oversubscribed in the next session; a validator that is not at capacity cannot be oversubscribed unless more nominators select it before the next election.

Candidate

A candidate is a submitted block to the validators. A block stops being referred to as a candidate as soon as it has been finalized.

Collations

A collation is a data structure that contains the proposed block candidate, including an optional validation Runtime update and upward messages.

Commission

Validators and nominators get paid from block production on the network, where validators can set a variable commission rate, which is initially subtracted from the total rewards that the validator is entitled to (for that period), where the commission determines the rate of distribution for the remaining rewards set out for the nominators that are backing that validator.

Community Queue

The queue for proposals originating from individual accounts (i.e. not the Council) which are waiting to become referenda. Compare the External queue.

Consensus

The process of a group of entities to agree on a particular data value (such as the ordering and makeup of blocks on a blockchain). There are a variety of algorithms used for determining consensus. The consensus algorithm used by Mandala Chain is GRANDPA.

Curator

A person, group, or other entity charged with judging and verifying the successful completion of a Bounty.

Dapps

A generic term for a decentralized application, that is, one that runs as part of a distributed network as opposed to being run on a specific system or set of systems.

KPG

The native token for Mandala Chain. KPG serves some purposes: network governance (allowing them to vote on-chain upgrades and other exceptional events), and general operation (rewarding good actors and punishing bad actors).

Epoch

An epoch is a time duration in the BABE protocol that is broken into smaller time slots. Each slot has at least one slot leader who has the right to propose a block. In Garbha, it is the same duration as a session.

Era

A (whole) number of sessions, which is the period that the validator set (and each validator's active nominator set) is recalculated and where rewards are paid out.

Equivocation

Providing conflicting information to the network. BABE equivocation entails creating multiple blocks in the same slot. GRANDPA equivocation would consist of signing multiple conflicting chains.

External Queue

The queue for proposals originating with the Mandala Chain Council which are waiting to become referenda. Compare the Community queue.

Extrinsic

A SCALE encoded array consisting of a version number, signature, and varying data types indicating the resulting runtime function to be called, including the parameters required for that function to be executed. These state changes are invoked from the outside world, i.e. they are not part of the system itself. Extrinsics can take two forms, "**inherents**" and "transactions". For more technical details see the Mandala Chain spec (TODO).

Finality

The property of a block that cannot be reverted. Generally, created blocks are not final until some point in the future - perhaps never, in the case of "probabilistic finality". The Mandala Chain uses a deterministic finality gadget known as GRANDPA.

Finality Gadget

A mechanism that determines finality.

Frame

The collection of Substrate-provided pallets (Substrate Runtime Modules).

Genesis

The origin of a blockchain, also known as block 0. It can also be used to reference the initial state of the blockchain at origination.

EXAMPLE

In the *genesis* state Alice, Bob, and Charlie had 30 tokens each.

Governance

The process of determining what changes to the network are permissible, such as modifications to code or movement of funds. The governance system in Mandala Chain is on-chain and revolves around stakeholder voting.

Governance Council

An on-chain entity that consists of several on-chain accounts (starting at 6, eventually moving to the final value of 24). The Council can act as a representative for "passive" (non-voting) stakeholders. Council members have two main tasks: proposing referenda for the overall stakeholder group to vote on and cancelling malicious referenda.

GRANDPA Finality Gadget

GHOST-based Recursive ANcestor Deriving Prefix Agreement. It is the finality gadget for Mandala Chain, which allows asynchronous, accountable, and safe finality to the blockchain. For an overview of GRANDPA.

Hard Fork

A permanent diversion of a blockchain occurs quickly due to a high priority change in a consensus rule. Clients who follow a hard fork always need to upgrade their clients to continue following the upgraded chain. Hard forks are considered permanent divergences of a chain for which non-upgraded clients are following consensus rules incompatible to the ones followed by upgraded clients.

Hard Spoon

Defined by Jae Kwon of Cosmos as "a new chain that takes into account state from an existing chain; not to compete, but to provide broad access." A non-contentious blockchain that inherits the state of the underlying blockchain and creates a new branch of *the same blockchain*.

Inactive Nomination

A validator (or validators) that a nominator has selected to nominate, but is not actively validating this era. This type of nomination may become active in a future era.

Inherent

Extrinsics that are "inherently true." Inherents are not gossiped on the network and are put into blocks by the block author. They are not provably true the way that the desire to send funds is, therefore they do not carry a signature. A blockchain's ***runtime*** must have rules for validating inherents. For example, timestamps are inherents. They are validated by being within some margin that each validator deems reasonable.

Injected Account

An account that is not directly managed by the Mandala Chain UI but can be accessed through it, such as accounts controlled by the Polkadot.js extension.

Interoperability

The ability for some sort of system to exchange and make use of information often compared to "cross-chain" technologies.

Keep-Alive Check

The keep-alive check is used to indicate whether or not a transfer can allow the sending account to be reduced to less than the existential deposit, causing it to be reaped.

LIBP2P

An open-source library for encrypted peer-to-peer communications and other networking functions. More information at: https://libp2p.io/

Liveness

The property of a distributed system is that it will eventually come to some sort of consensus. A system stuck in an infinite loop would not be considered live, even if computations are taking place; a system that eventually provides a result, even if incorrect or it takes a long time, is considered to have liveness.

Mainnet

Short for "main network": the fully functional and acting chain that runs its own network.

Message

In Mandala Chain's XCMP protocol, a *message* is arbitrary data that is sent from one parachain (the egress chain) to another (the ingress chain) through a channel and ensured delivery by the validator set.

Message Queue

In Mandala Chain's XCMP protocol, a *message queue* is the list of messages waiting to be processed by a particular receiving parachain over a channel.

Metadata

Data that includes information about other data, such as information about a specific transaction.

Motion

A motion is essentially a "referendum" or "decision" being considered by the Council. The Council can vote on motions like approving Treasury Proposals or making proposals for the community to vote on.

Next Session

This indicates that the validator will be a member of the active set in the next session.

Node Explorer

A tool that gives you information about a node, such as the latest blocks sealed, finalized, and the current chain state as known by that node.

Non-fungible Token (NFT)

A non-fungible token is a token that does not hold the property of fungibility, which, in turn, means that it cannot be interchangeable and indistinguishable from other tokens. NFTs allow the tokenization of unique items and provide exclusive ownership for those tokens.

On-chain Governance

A governance system of a blockchain that is controlled by mechanisms on the blockchain.

On-chain governance allows decisions to be made transparently. Note that there are a variety of different algorithms for making these decisions, such as simple majority voting, adaptive quorum biasing, or identity-based quadratic voting.

Online Message

This is a message that is broadcast by a validator to verify to the network that the validator is online, even if they haven't published a block this epoch. This is sometimes referred to as "ImOnline".

Origin

The initiator of an extrinsic. A simple origin would be the account that is sending a token to another account. Mandala Chain also supports more complex origin types, such as the ***root origin***, from which privileged functions can be called.

Oversubscribed

If more than the maximum number of nominators nominate the same validator, it is "oversubscribed", and only the top staked nominators (ranked by the amount of stake, up to the maximum number of nominators) are paid rewards. Other nominators will receive no rewards for that era. The current maximum number of nominators is 512 on Mandala Chain, but it can be modified via governance.

Pallet

A Substrate runtime module.

Host

The environment in which a runtime module can be executed. Parachains must support the Mandala Chain Host - external chains that do not will have to use a bridge. Previously known as the Mandala Chain Runtime Environment.

Preimage

The on-chain proposals do not require the entire image of extrinsics and data (for instance the WASM code, in case of upgrades) to be submitted, but would rather just need that image's hash. That **preimage** can be submitted and stored on-chain against the hash later, upon the proposal's dispatch.

Proof of Stake (PoS)

A method of selecting participation in a consensus system, in which participants are chosen based on how many tokens they have at stake (at risk of loss due to misbehavior). Normally, Proof-of-Stake systems limit the number of participants.

Proof of Work (PoW)

A method of selecting participants in a consensus system, typically the longest chain rule, in which participants try to solve a puzzle like finding a partial pre-image of a hash. Normally, a Proof-of-Work system can have any number of participants.

Proposal

A potential function call to be voted on in a referendum. Proposals modify the behavior of the Mandala Chain network, from minor parameter tuning up to replacing the runtime code.

Protocol

A system of rules that allows two or more entities of a communications system to transmit information. The protocol defines the rules, syntax, semantics, and synchronization of communication and possible recovery methods.

Random Seed

A random seed is a pseudo-random number available on-chain. It is used in various places of the Mandala Chain protocol, most prominently in BABE the block production mechanism.

Referendum

A vote on whether or not a proposal should be accepted by the network. Referenda may be initiated by the Governance Council, by a member of the public, or as the result of a previous proposal. Stakeholders vote on referenda, weighted by both the size of their stake (i.e. number of KPG held) and the amount of time they are willing to lock their tokens.

Re-Genesis

Re-Genesis is the process of exporting the current chain state, and creating a new chain that builds on it. Re-Genesis will involve stop-the-world migration, which results in a period of time when no actual blocks are added to the blockchain. In a way, re-genesis can be viewed as a hard fork process. A formal design of Re-Genesis on Substrate is still under development - Re-Genesis Rationale and Design.

Remarks

Remarks are extrinsics with no effect. They provide additional information to external inputs, acting as *notes*. Remarks are stored alongside block records and do not change the chain's storage; the information is not stored in the chain's trie, but along blocks.

Rococo

The testnet set aside for testing parachains, cumulus, and related technology.

Root Origin

A system-level origin in Substrate. This is the highest privilege level and can be thought of as the superuser of the runtime origin. To learn about more raw origins in Substrate, visit Substrate Docs

Runtime

The state transition function of a blockchain. It defines a valid algorithm for determining the state of the next block given the previous state.

Runtime Module

A module that implements specific transition functions and features one might want to have in their runtime. Each module should have domain-specific logic. For example, a Balances module has logic to deal with accounts and balances. In Substrate, modules are called "pallets".

Safety

The property of a distributed system indicating that a particular state transition will not be reverted. GRANDPA provides *deterministic* safety. That is, for a state changed marked as "safe" or "final", one would require a hard fork to revert that change.

Scalability

While an ambiguous concept, [blockchain] scalability can be understood as the ability for the network to scale in capabilities (e.g. processing more transactions) when needed.

Session

A session is a Substrate implementation term for a period that has a constant set of validators. Validators can only join or exit the validator set at a session change.

Session Certificate

A message containing a signature on the concatenation of all the Session keys.

Session Key

Hot keys that are used for performing network operations by validators, for example, signing GRANDPA commit messages.

Soft Fork

A backward compatible change to client code causes upgraded clients to start mining a new chain. Requires a "vote-by-hashrate" of a majority of miners to enact successfully. Soft forks are considered temporary divergences in a chain since non-upgraded clients do not follow the new consensus rules but upgraded clients are still compatible with old consensus rules.

Software Development Kit (SDK)

A collection of software tools (and programs) packaged together that can be used to develop software.

Staking

The act of bonding tokens for Mandala Chain (KPG) by putting them up as "collateral" for a chance to produce a valid block (and thus obtain a block reward). Validators and nominators stake their KPG in order to secure the network.

State transition function

A function that describes how the state of a blockchain can be transformed. For example, it may describe how tokens can be transferred from one account to another.

Substrate

A modular framework for building blockchains. Mandala Chain is built using Substrate. Chains built with Substrate will be easy to connect as parachains. For developers, see the Substrate GitHub repository.

System Parachains

Parachains that are part of the Mandala Chain core protocol. These are allocated a parachain execution core by governance rather than auction.

Tabling

In Mandala Chain governance, bringing a proposal to a vote via referendum. Note that this is the British meaning of "tabling", which is different from the US version, which means "to postpone" a measure.

Teleport

Send an asset from an account on one chain to an account on a different chain. This occurs by burning an amount on the sending chain and minting an equivalent amount on the destination chain.

Testnet

Short for "test network": an experimental network where testing and development takes place. Networks are often executed on a testnet before they are deployed to a mainnet.

Tokenization

The process of replacing sensitive data with non-sensitive data.

Tracks

Each Origin is associated with a single referendum class and each class is associated with a Track. The Track outlines the lifecycle for the proposal and is independent from other class's tracks. Having independent tracks allows the network to tailor the dynamics of referenda based upon their implied privilege level.

Tranche

Validators use a subjective, tick-based system to determine when the approval process should start. A validator starts the tick-based system when a new availability core candidates have been proposed, which can be retrieved via the Runtime API, and increments the tick every 500 milliseconds. Each tick/increment is referred to as a “tranche”, represented as an integer, starting at 0.

Transfer

Send an asset from one account to another. This generally refers to transfers that occur only on the same chain.

Transaction

An extrinsic that is signed. Transactions are gossiped on the network and incur a transaction fee. Transactions are "provably true", unlike inherents. For example, one can prove that Alice wants to send funds to Bob by the fact that she signed a transfer-funds message with her private key.

Validator

A node that secures the Chain by staking KPG, validating proofs, and voting on consensus along with other validators.

Voting

The process of stakeholders determining whether or not a referendum should pass. Votes are weighted both by the number of KPG that the stakeholder account controls and the amount of time they are willing to lock their KPG.

Waiting Nomination

The nominator has nominated this validator, but the validator was not elected into the active validator set this era and thus cannot produce blocks for the canonical chain. If the validator does get into the active set in a future era, this may turn into an active or inactive nomination.

Wallet

A program that allows one to store private keys and sign transactions for Mandala Chain or other blockchain networks.

Wasm

The abbreviation for WebAssembly.

Watermark

In Mandala Chain's parachain messaging scheme, the *watermark* is the minimum processed send-height of the receiving parachain. All messages on all channels that are sending to this parachain at or before the watermark are guaranteed to be processed.

Web3 Foundation

A Switzerland-based foundation that nurtures and stewards technologies and applications in the fields of decentralized web software protocols, particularly those that utilize modern cryptographic methods to safeguard decentralization, to the benefit and for the stability of the Web3 ecosystem.

WebAssembly

An instruction format for a virtual, stack-based machine. Mandala Chain Runtime Modules are compiled to WebAssembly. Also known as Wasm.

Weights

A permission-less system needs to implement a mechanism to measure and limit usage in order to establish an economic incentive structure, to prevent the network overload, and to mitigate DoS vulnerabilities. This mechanism must enforce a limited time-window for block producers to create a block and include limitations on block size, to prevent execution of certain extrinsics which are deemed too expensive and could decelerate the network. This is handled by the weight system, where the cost of the transactions (referred to as extrinsics) are determined before execution. Checkout this section of the Substrate docs covering transaction weights and fees.

Witness

Cryptographic proof statements of data validity.

Whitelist Pallet

Allows one Origin to escalate the privilege level of another Origin for a certain operation. In terms of OpenGov, it allows the Fellowship to authorise a new origin (which we will call Whitelisted-Root) to be executed with Root-level privileges.

Network Token

In the Mandala Chain network, the native token of Madya Mandala is Kepeng Token, with the ticker KPG, whereas the testnet token of Niskala testnet is KPGT. Both KPG and KPGT tokens have 18 decimals.

Utility

The Kepeng Token (KPG) is the utility token for Madya Network and has two primary functions.

1. Transactions
2. On-chain Governance

Transactions

Every on-chain transaction requires the sender to pay fees. Part of the fee is burned, and part is deposited to the treasury.

On-chain Governance

At Mandala Chain, we believe in the power of decentralized decision-making. That's why we're committed to implementing on-chain governance in the future, where every community member has a voice in shaping the network's future. The Kepeng token plays a crucial role in this process, serving as the primary means for facilitating governance activities such as voting and referenda.

The Kepeng Testnet Token (KPGT) is a utility token for testing the features of the Niskala testnet. More details can be found in the Faucet section.

Smart Contracts

Overview

Mandala Chain is a Substrate-based blockchain platform. Developers can create and deploy smart contracts on Mandala Chain using two types of technologies: WebAssembly (Wasm) and Ethereum Virtual Machine (EVM). This dual compatibility allows developers to build decentralized applications (DApps) using their preferred tools and languages. Mandala Chain commited to ensure smooth transitions for developers that comes from other blockchain environments.

Types of Smart Contracts on Mandala Chain

WebAssembly (Wasm) Smart Contracts

Wasm smart contracts on Mandala Chain leverage the `pallet-contracts` API, a sandboxed environment for deploying and executing contracts. These contracts are written in languages that compile to Wasm, such as **ink!** (Rust-based) or **ask!** (AssemblyScript-based).

Key Features:

- **Compatibility with `pallet-contracts` API**: Ensures seamless integration with the runtime environment.
- **Sandboxed Execution**: Uses the Wasm interpreter `wasmi` for secure and correct execution of untrusted smart contract code.
- **Two-Step Deployment Process**:

i. **Upload Contract Code**: The Wasm code is uploaded to the chain with a unique identifier (`code_hash`).
ii. **Instantiate Contract**: Create an address and storage for the contract based on the uploaded code.

This decoupled approach reduces on-chain storage requirements and allows multiple instances of a single contract code, saving costs for developers. For example, standardized token contracts like PSP22 or PSP34 can be reused without re-uploading the code.

To learn more about building Wasm smart contracts, refer to the Wasm Smart Contracts section.

EVM Smart Contracts

Mandala Chain leverage EVM-frontier to support EVM smart contracts, enabling developers to use Solidity or any other language that compiles to EVM-compatible bytecode. This feature provides a familiar environment for Ethereum developers while leveraging Mandala Chain's robust infrastructure.

Key Features:

- **Solidity Compatibility**: Write contracts in Solidity or other supported languages.
- **EVM Bytecode Support**: Deploy contracts compiled into EVM-compatible bytecode.
- **Low Friction Development**: Develop, test, and execute smart contracts that is compatible with the existing Ethereum developer toolchain.

To learn more about building EVM smart contracts on Mandala Chain, refer to the EVM Smart Contracts section (upcoming)

Why Choose Mandala Chain?

Mandala Chain bridges the gap between Web2 development and blockchain technology by offering:

- Flexibility in choosing between Wasm and EVM environments.
- Developer-friendly tools and documentation tailored for both ecosystems.
- A scalable and secure platform built on Substrate's proven architecture.

Start your journey into blockchain development with Mandala Chain today!

Smart Contract vs Layer-1 blockchain, what's the difference?

In this article we will discuss the differences between developing a smart contract and a layer 1 blockchain.

Layers of Abstraction

When you write a smart contract, you create instructions that associate with and deploy to a specific chain address.

In comparison, a runtime module (known as a *pallet*) on a layer 1 blockchain is the entire logic of a chain's state transitions (called a *state transition function*).

Smart contracts must consciously implement upgradeability, while layer 1 blockchains can swap out their code entirely through a root command or via the governance pallet.

When you build a smart contract, it will eventually be deployed to a target chain with its own environment. Layer 1 blockchains allow the developer to declare the environment of their own chain, even allowing others to write smart contracts for it.

Gas Fees

Smart contracts must find a way to limit their own execution, or else full nodes are vulnerable to DOS attacks. An infinite loop in a smart contract, for example, could consume the computational resources of an entire chain, preventing others from using it. The halting problem shows that even with a powerful enough language, it is impossible to know ahead of time whether or not a program will ever cease execution. Some platforms, such as Bitcoin, get around this constraint by providing a very restricted scripting language. Others, such as Ethereum, "charge" the smart contract "gas" for the rights to execute their code. If a smart contract does get into a state where execution will never halt, it eventually runs out of gas, ceases execution, and any state transition that the smart contract would have made is rolled back.

Layer 1 blockchains can implement arbitrarily powerful programming languages and contain no gas notion for their native logic. This means that some functionality is easier to implement for the developer, but some constructs, such as a loop without a terminating condition, should never be implemented. Leaving certain logic, such as complex loops that could run indefinitely, to a non-smart contract layer or even trying to eliminate it is a wiser choice. Layer 1 blockchains try to be proactive, while smart contract platforms are event-driven.

The Mandala Chain and other Substrate-based blockchains typically use the ***weight-fee*** model and **not** a ***gas-metering*** model.

Build

Why Build on Mandala Chain?

Mandala Chain is an advanced interoperable smart contract platform that seamlessly integrates with both the Polkadot and Ethereum blockchain ecosystems. It offers native support for both WebAssembly (Wasm) and Ethereum Virtual Machine (EVM) environments, operating on its Layer 1 scaling solutions and Parachain-based networks. This dual compatibility ensures that developers can leverage the strengths of both blockchain technologies, enabling versatile and scalable decentralized applications.

Mandala Chain Key Features

Mandala Chain is equipped with a range of features that make it a powerful platform for blockchain development:

☑️ **Decentralized Identity Verification:** Mandala Chain offers decentralized identity verification, allowing users to confirm their identities without revealing personal information.

☑️ **Cross-chain communication:** Mandala Chain supports cross-chain communication, enabling apps and services to communicate securely across different blockchains.

☑️ **Scalability:** Mandala Chain is designed to handle a high volume of transactions, ensuring that the network can scale to meet the needs of users.

☑️ **Governance:** Mandala Chain has a decentralized governance model, giving users a voice in the decision-making process and ensuring transparency and accountability.

☑️ **Interoperability:** Mandala Chain provides interoperability between different blockchains, enabling seamless communication and exchange of data and value.

✅ **Smart Contract:** Smart contracts on Mandala Chain enable automated execution of agreements without intermediaries, improving efficiency and reducing costs.

✅ **DApp Ecosystem:** Mandala Chain is designed to support the development of decentralized applications, creating a thriving ecosystem of innovative apps and services.

✅ **Security & Privacy Protection:** Mandala Chain uses advanced encryption techniques to protect user privacy, ensuring that sensitive data is kept confidential and secure.

Introduction

Overview

To make use of this documentation effectively, you should possess a general understanding of programming basics. The programming languages used throughout are mainly Rust, Solidity, and JavaScript, for which previous knowledge is not necessary but would be highly beneficial. In conjunction with improving your understanding of the material contained within these guides, we recommend that you review supplemental material covering these languages in order to improve your overall understanding of the topics and practical code examples provided.

Do I need blockchain knowledge to follow this documentation?

Your blockchain knowledge will be useful, but if you are reading this, that means you are on the right track.

I'm a Polkadot builder. Do I need this?

If you are already a builder on the Polkadot/Kusama ecosystem, you can most likely skip the Introduction chapter. But it is recommended that you read about available networks.

Do I need to be a developer to understand the Introduction chapter?

To use this introduction chapter, you do not need any programming skills, and it will be useful later when you step into more advanced topics.

Mandala Chain Networks

The Mandala Chain Networks are designed to provide a robust and flexible blockchain infrastructure, supporting both test and main network environments. This documentation outlines the key components and configurations of the Mandala Chain Networks.

Testnets

Testnets are essential for developers and testers to experiment with new features and applications without affecting the main network. They provide a safe environment to test upgrades, smart contracts, and other blockchain functionalities.

Niskala

Niskala is the designated testnet for the Mandala Chain. It serves as a sandbox environment where developers can deploy and test their applications. Key features of the Niskala testnet include:

- **Free Transactions:** Transactions on Niskala are free, allowing developers to test without incurring costs.
- **Frequent Resets:** The network is periodically reset to ensure a clean testing environment and to incorporate the latest updates.
- **Realistic Conditions:** Niskala mimics the conditions of the mainnet, providing a realistic testing ground for applications.

To connect to the Niskala testnet, use the following RPC endpoint in here

Mainnets

Mainnets are the live networks where real transactions occur, and they hold actual value.

The Mandala Chain mainnet is designed for high performance, security, and scalability.

Mandala Chain

- The Mandala Chain is the primary mainnet, offering a secure and efficient blockchain platform for various applications.
- The functionality will be equal to the Niskala testnet, with a focus on the proven security and stability that have been battle-tested on the Niskala testnet.

For further assistance or inquiries, please contact the Mandala Chain support team at [email protected].

Use

In the following sections you will find all the help you need to create & manage accounts, and transact on Niskala Network.

Account

Niskala is an Ethereum-compatible blockchain network that operates within the Polkadot ecosystem. Niskala replaces the default Substrate-style accounts and keys with Ethereum-st…

Wallets

Connect your wallet to Niskala

Account

Niskala is an Ethereum-compatible blockchain network that operates within the Polkadot ecosystem. Niskala replaces the default Substrate-style accounts and keys with Ethereum-style accounts and keys. This compatibility allows users to interact with their Niskala accounts using MetaMask and other Ethereum tools.

Additionally, Niskala supports interaction through Polkadot.js Apps, which natively handle H160 addresses and ECDSA keys. This dual compatibility provides flexibility and ease of use for developers and users alike.

Substrate EVM Compatible Blockchain

Niskala, as a Substrate-based blockchain, can offer a full EVM implementation. This capability allows for the execution of Solidity-based smart contracts with minimal to no changes. By integrating the EVM pallet into its runtime, Niskala supports EVM functionality, while the Ethereum Pallet with Frontier ensures Ethereum RPC compatibility. This setup enables Niskala to offer Ethereum compatibility, similar to other parachains in the Polkadot ecosystem.

Address Mapping

In this configuration, users have Ethereum-style addresses (H160 format) on a Substrate-based chain. These addresses are 42-character hexadecimal strings, including the "0x" prefix, and are linked to private keys used for signing transactions on the Ethereum side of the chain. However, these addresses are mapped to Substrate-style addresses (H256 format) within the Substrate Balance pallet.

A user, such as Alice, only knows the private key of her H160 address and not the mapped H256 version. Consequently, she can perform read-only operations through Substrate’s API but cannot send transactions with her H256 address. To fully operate on the Substrate side of the chain, including staking, balances, and governance, Alice needs a separate H256 address with a different private key.

Niskala Unified Accounts

Niskala aims to establish a fully Ethereum-compatible environment within the Polkadot ecosystem, prioritizing an exceptional user experience. This commitment goes beyond the fundamental features of Ethereum, incorporating additional functionalities such as on-chain governance, staking, and cross-chain integrations.

With the unified account system, users, like Bob, only need one H160 address and its associated private key to perform all the aforementioned tasks, encompassing both EVM and Substrate operations.

In this new configuration, Bob retains just one private key linked to a single address. He no longer needs to transfer balances between multiple accounts and can access all features using this one account and private key. This single account has been standardized to align with Ethereum's H160 address format and ECDSA key standards.

Wallets

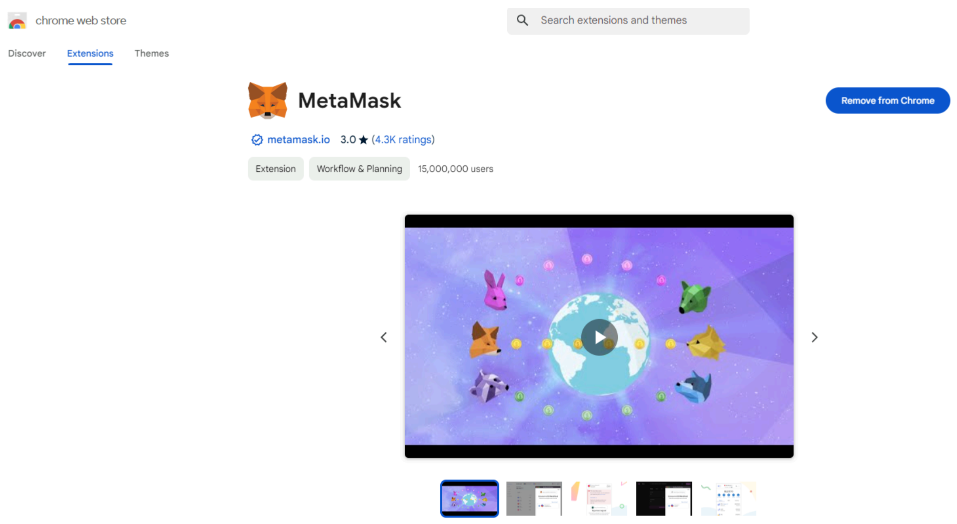
Connect your wallet to Niskala

MetaMask App

MetaMask is a popular Ethereum wallet and browser extension that allows users to manage their accounts, send transactions, and interact with decentralized applications (dApps).

Here’s a step-by-step guide on how to set up and use MetaMask with Niskala:

Step 1: Install MetaMask

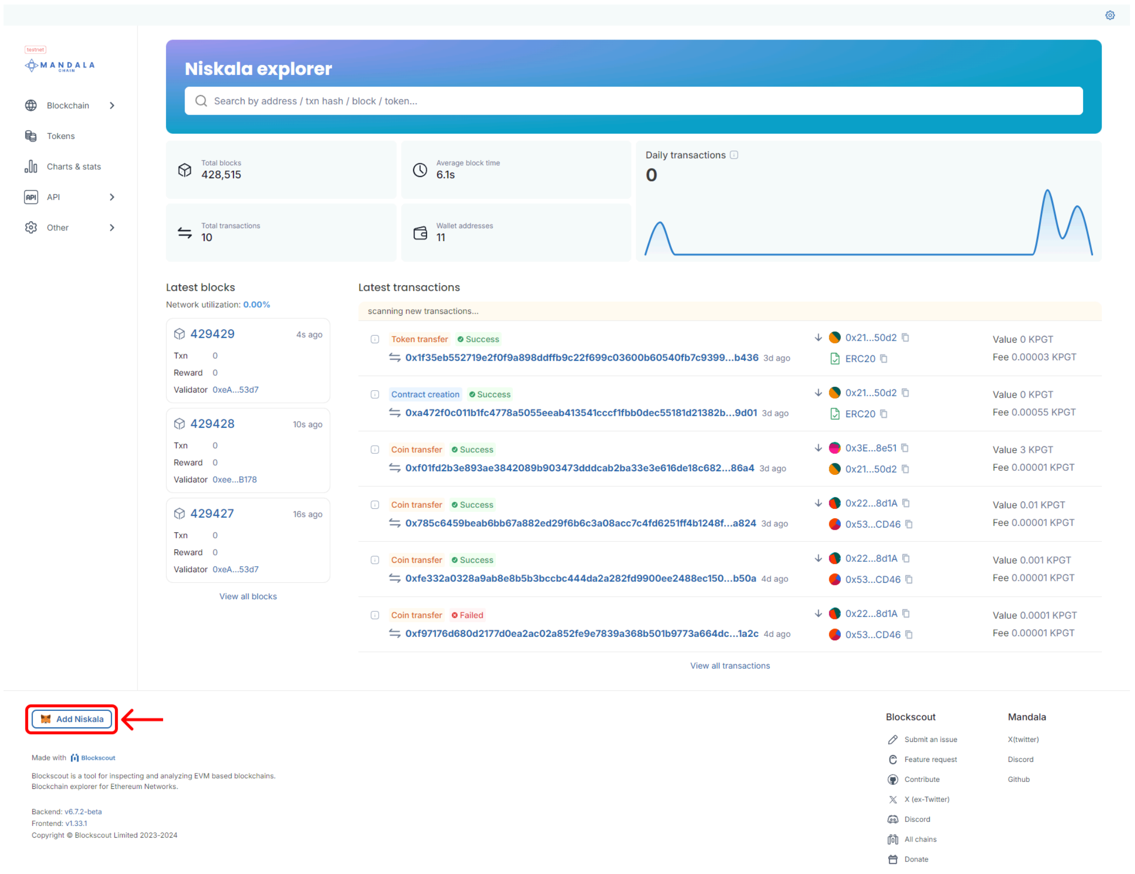


1. **Download MetaMask**: Visit the MetaMask website and download the extension for your preferred browser (Chrome, Firefox, Brave, or Edge).
2. **Install the Extension**: Follow the installation instructions provided by your browser to add MetaMask as an extension.
3. **Create or Import an Account**: Once installed, open MetaMask and either create a

new Ethereum account or import an existing one using your seed phrase.

Step 2: Connect MetaMask to Niskala

1. **Open MetaMask**: Click on the MetaMask icon in your browser to open the extension.
2. **Add Niskala Network**: Click the "Add Niskala" button and verify network details.



Niskala explorer

Search by address / txn hash / block / token...

Total blocks 428,515

Average block time 6.1s

Daily transactions 0

Total transactions 10

Wallet addresses 11

Latest blocks

Network utilization: 0.00%

429429 4s ago
Txn 0
Reward 0
Validator 0xeA...53d7

429428 10s ago
Txn 0
Reward 0
Validator 0xee...B178

429427 16s ago
Txn 0
Reward 0
Validator 0xeA...53d7

View all blocks

Latest transactions

scanning new transactions...

Token transfer Success
0x1f35eb552719e2f0f9a898ddffb9c22f699c03600b60540fb7c9399...b436 3d ago
0x21...50d2
ERC20
Value 0 KPGT
Fee 0.00003 KPGT

Contract creation Success
0xa472f0c011b1fc4778a5055eeab413541cccf1fbb0dec55181d21382b...9d01 3d ago
0x21...50d2
ERC20
Value 0 KPGT
Fee 0.00055 KPGT

Coin transfer Success
0xf01fd2b3e893ae3842089b903473dddcab2ba33e3e616de18c682...86a4 3d ago
0x3E...8e51
0x21...50d2
Value 3 KPGT
Fee 0.00001 KPGT

Coin transfer Success
0x785c6459beab6bb67a882ed29f6b6c3a08acc7c4fd6251ff4b1248f...a824 3d ago
0x22...8d1A
0x53...CD46
Value 0.01 KPGT
Fee 0.00001 KPGT

Coin transfer Success
0xfe332a0328a9ab8e8b5b3bccbc444da2a282fd9900ee2488ec150...b50a 4d ago
0x22...8d1A
0x53...CD46
Value 0.001 KPGT
Fee 0.00001 KPGT

Coin transfer Failed
0xf97176d680d2177d0ea2ac02a852fe9e7839a368b501b9773a664dc...1a2c 4d ago
0x22...8d1A
0x53...CD46
Value 0.0001 KPGT
Fee 0.00001 KPGT

View all transactions

Add Niskala

Made with Blockscout

Blockscout is a tool for inspecting and analyzing EVM based blockchains.
Blockchain explorer for Ethereum Networks.

Backend: v6.7.2-beta
Frontend: v1.33.1
Copyright © Blockscout Limited 2023-2024

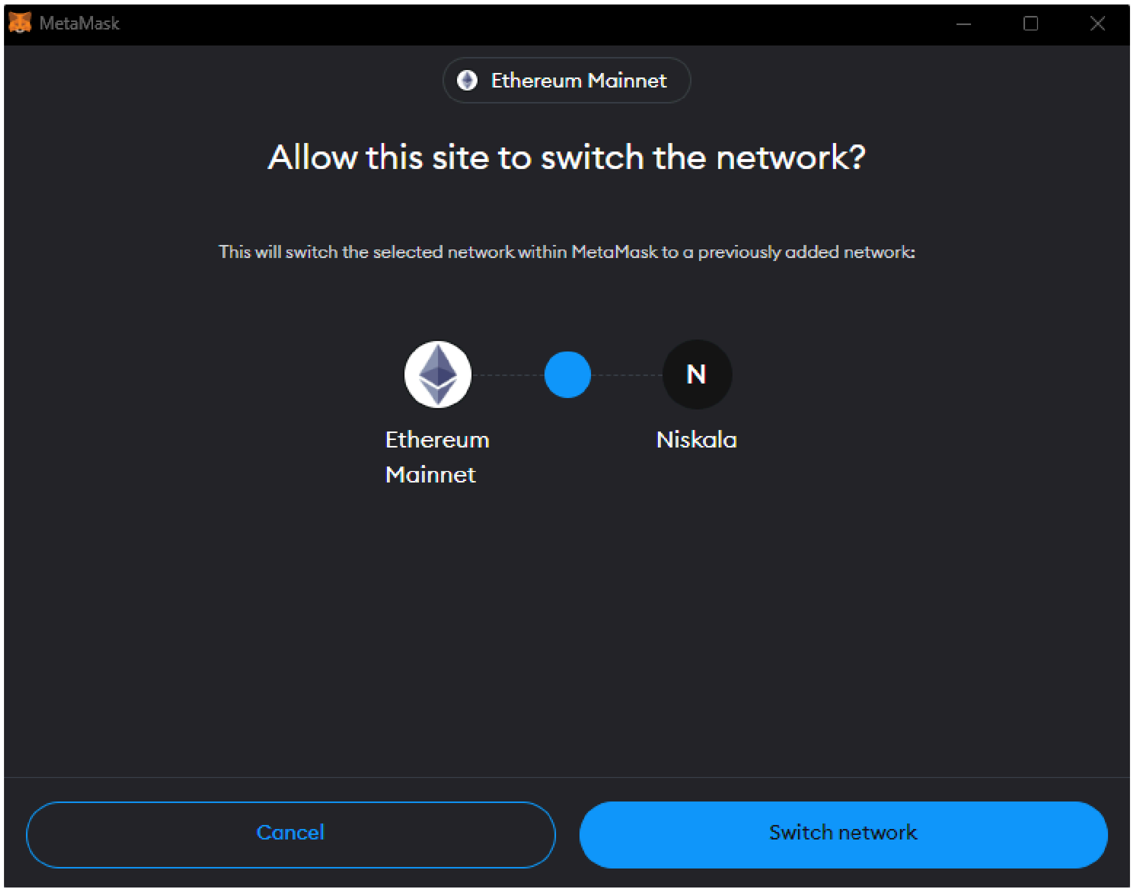
Blockscout
Submit an issue
Feature request
Contribute
X (ex-Twitter)
Discord
All chains
Donate

Mandala
X(twitter)
Discord
Github

- **Network Name**: Niskala Network
- **Network RPC URL**: `https://mlg1.mandalachain.io/`
- **Chain ID**: `6025`
- **Currency Symbol**: KPGT

Step 3: Interact with Niskala

1. **Approve and Switch to Niskala Network**: Ensure that you are connected to the Niskala network.



Allow this site to switch the network?

This will switch the selected network within MetaMask to a previously added network:

2. **Manage Your Account**: You can now view your Niskala account balance, send transactions, and interact with dApps on the Niskala network.

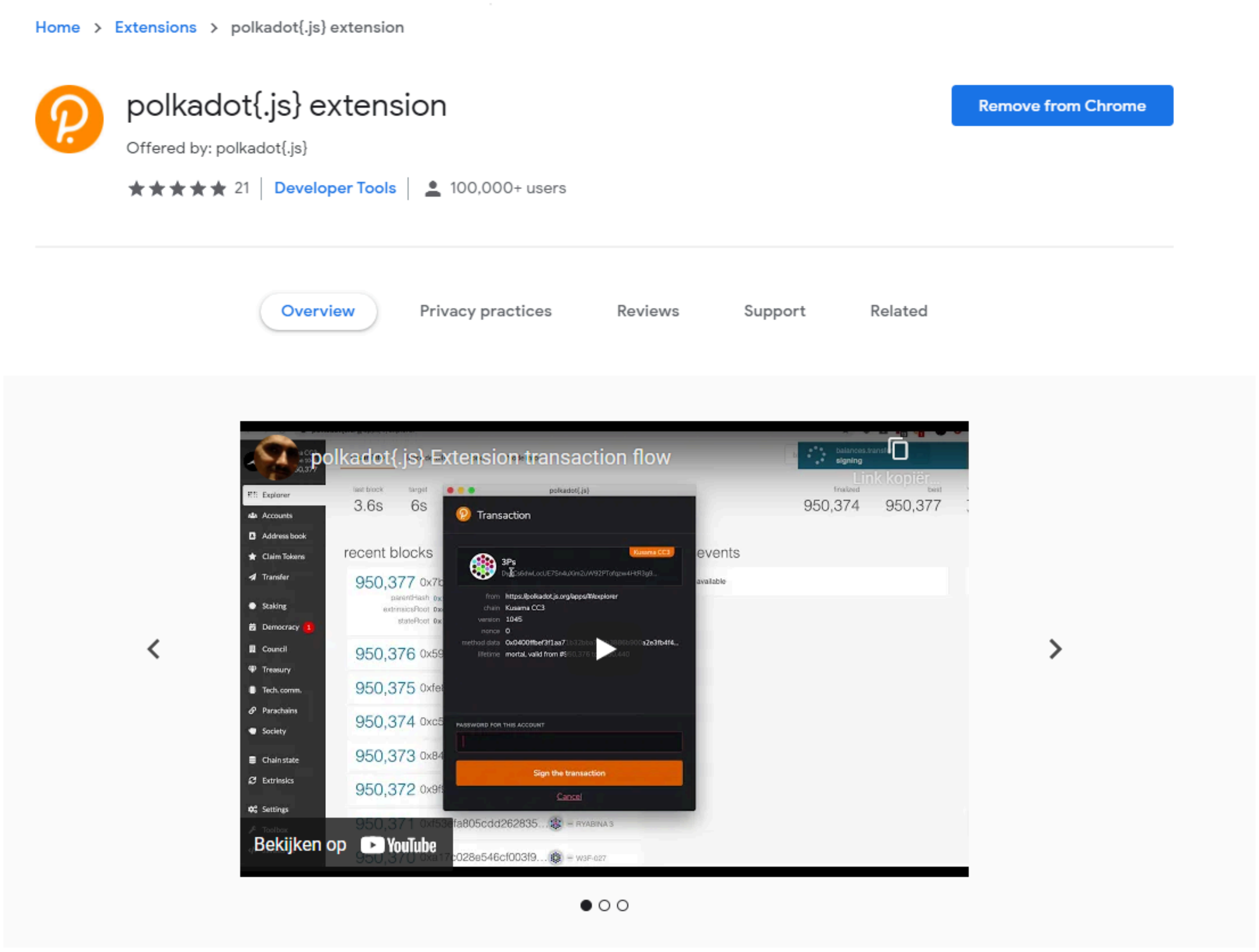
By following this guide, users can effectively manage their Niskala accounts using MetaMask and take advantage of the blockchain's Ethereum compatibility.

Polkadot JS App

Polkadot.js Browser Plugin

Here’s a step-by-step guide on how to set up and use Polkadot.js with Niskala:

The browser plugin is available for both Google Chrome (and Chromium-based browsers like Brave) and Firefox. After installing the plugin, you should see the orange and white Polkadot.js logo in your browser menu bar.



Home > Extensions > polkadot{.js} extension

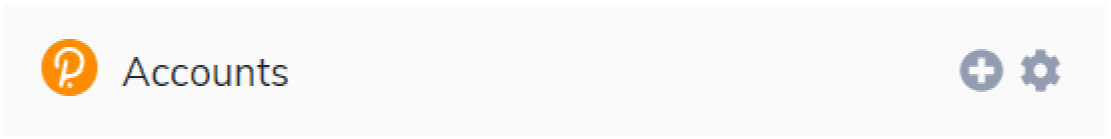
polkadot{.js} extension

Offered by: polkadot{.js}

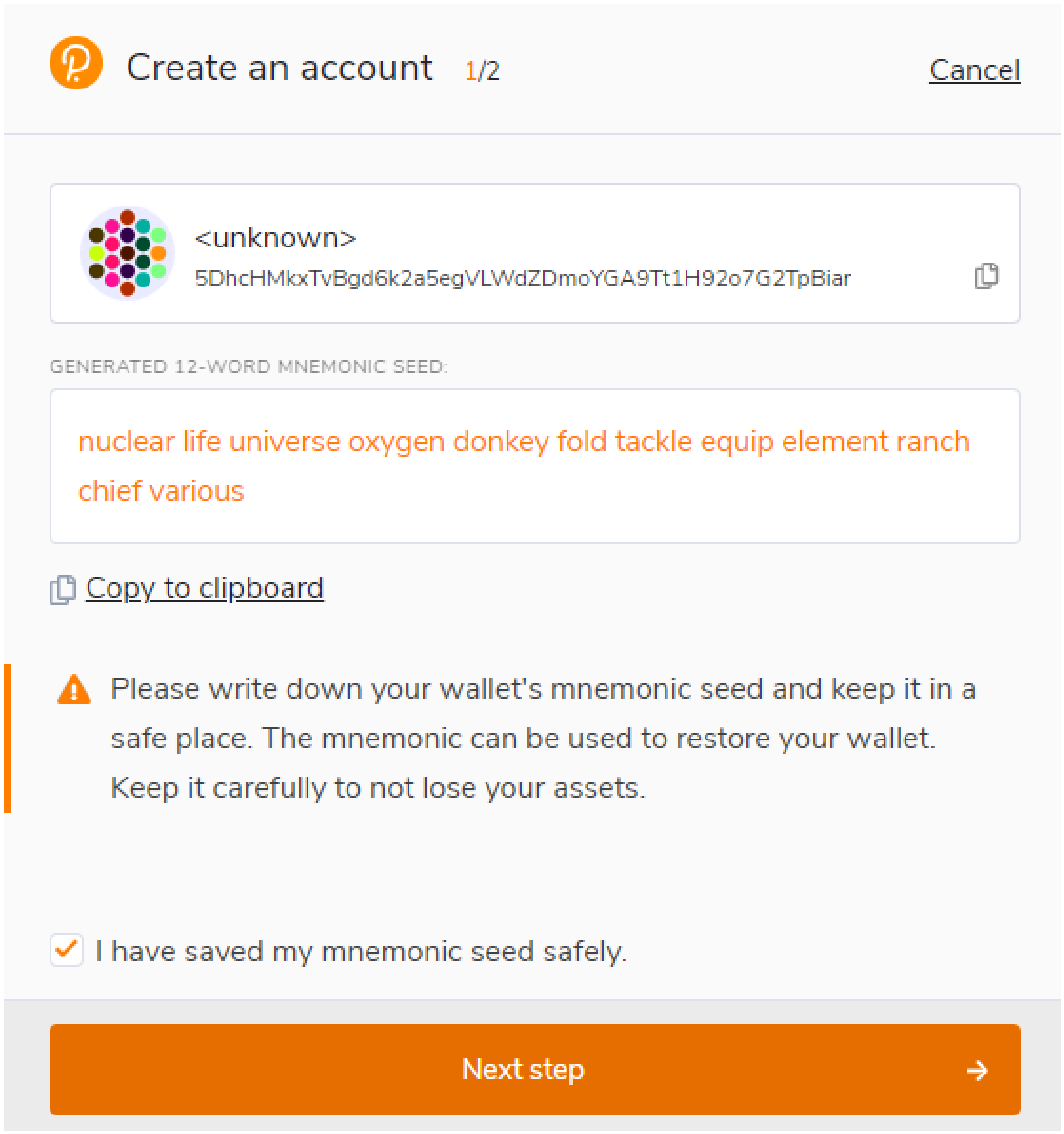
★★★★★ 21 | Developer Tools | 100,000+ users

Create Account

Open the Polkadot.js browser extension by clicking the logo on the top bar of your browser. You will see a browser popup.



Click the big plus button - "Create new account." The Polkadot.js plugin will then use system randomness to make a new seed for you and display it to you in the form of twelve words.


Create an account 1/2

Cancel

<unknown>

5DhcHMkxTvBgd6k2a5egVLWdZDmoYGA9Tt1H92o7G2TpBiar

GENERATED 12-WORD MNEMONIC SEED:

nuclear life universe oxygen donkey fold tackle equip element ranch chief various

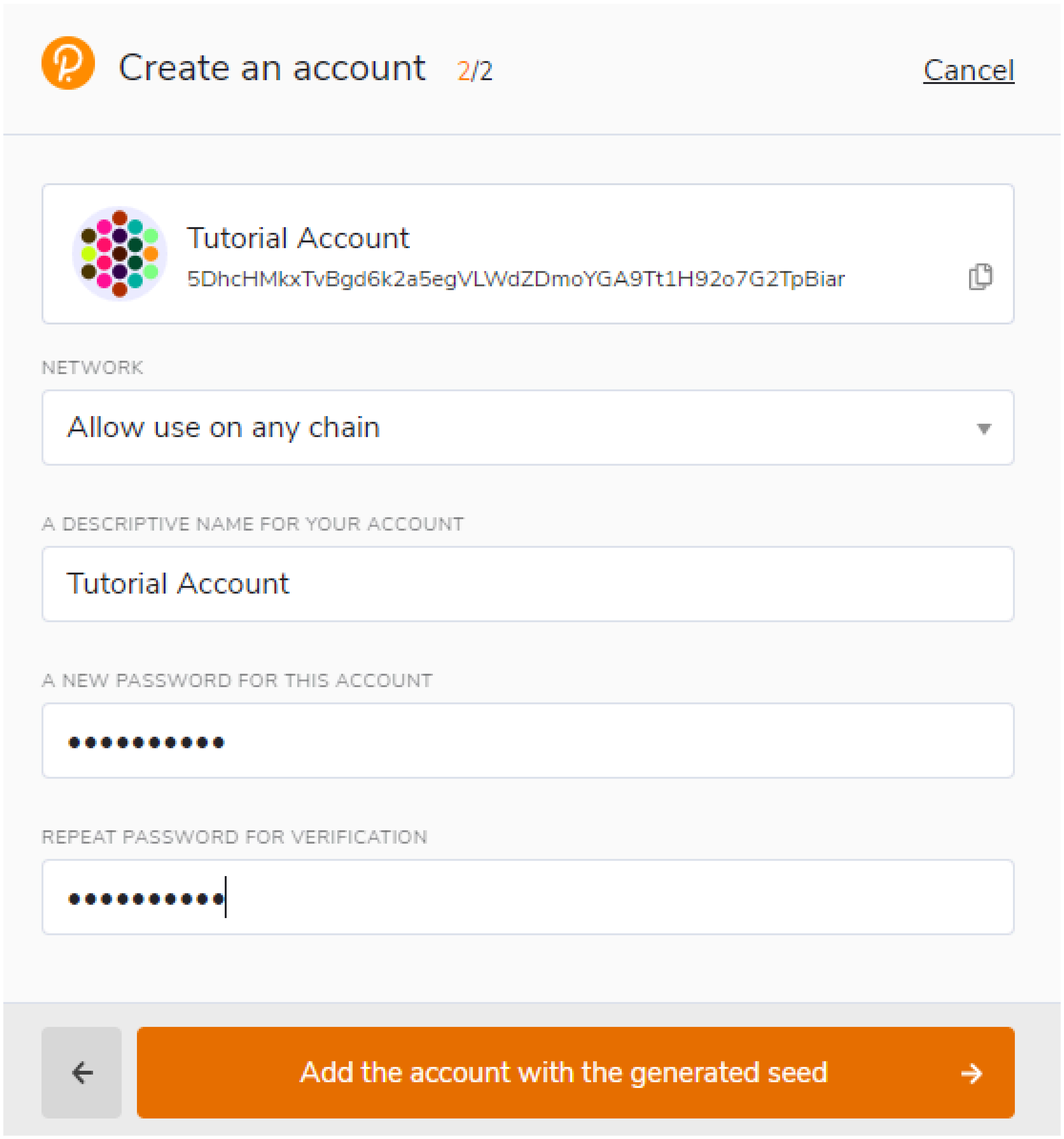
Copy to clipboard

Please write down your wallet's mnemonic seed and keep it in a safe place. The mnemonic can be used to restore your wallet. Keep it carefully to not lose your assets.

✓ I have saved my mnemonic seed safely.

Next step →

You need to back up these words. Please store the seed somewhere safe and secure. If you cannot access your account via Polkadot.js for some reason, you can re-enter your seed through the "Add account menu" by selecting "Import account from pre-existing seed".


Create an account 2/2

Cancel

Tutorial Account
5DhcHMkxTvBgd6k2a5egVLWdZDmoYGA9Tt1H92o7G2TpBiar

NETWORK

Allow use on any chain

A DESCRIPTIVE NAME FOR YOUR ACCOUNT

Tutorial Account

A NEW PASSWORD FOR THIS ACCOUNT

●●●●●●●●●●●

REPEAT PASSWORD FOR VERIFICATION

●●●●●●●●●●●

It is best to create an account that is allowed on *any chain* in the Mandala Chain ecosystem. This account can then be used for Niskala. Your account will automatically change format when connected to a chain.

A **descriptive name** is arbitrary and for your use only. It is not stored on the blockchain and will not be visible to other users who look at your address via a block explorer. If you're juggling multiple accounts, it helps to make this as descriptive and detailed as needed.

The **password** will be used to encrypt this account's information. You will need to re-enter it when using the account for any outgoing transaction or to cryptographically sign a message.

> **DANGER!** Note that this password does **NOT** protect your ***seed phrase***. If someone knows the twelve words in your mnemonic seed, they still have control over your account even if they do not know the password.

After clicking on "Add the account with the generated seed", your account is created. We recommend also saving your account as a JSON file somewhere safe.

Support

In case you have any problems, our team can provide support. Please remember that we will NEVER DM you first and we will never ask for any funds or tokens! If you get approached by someone *pretending* to be part of the team, do NOT engage with them. Instead, please reach out to us using our official support email: [email protected]

Build Environment

Setting up various environments is a crucial step in the development and deployment of blockchain applications. Reviewing the following sections will provide valuable insights into the purpose and specific requirements of each environment.

For example, if you are building and testing WebAssembly (Wasm) smart contracts, and ink! environment.

When you're ready to deploy a smart contract to production, this section will guide you through configuring an RPC endpoint, ensuring a seamless transition from development to deployment.

Network RPC endpoints

What is RPC Node?

- RPC nodes can be queried for information and used to initiate transactions.
- The purpose of RPC nodes is to allow decentralized applications and clients to communicate with a Blockchain network.
- RPC nodes listen for requests, respond with the necessary data, or execute the requested transaction.
- Common usage of RPC nodes includes querying the Blockchain for information, sending transactions, and executing smart contract functions.
- RPC nodes are important for decentralized applications to function and interact with a Blockchain network, allowing for decentralized exchange and other use cases.

Public Endpoints

> **INFO**
>
> The free endpoints below are dedicated to end users, they can be used to interact with dApps or deploy/call smart contracts.
>
> They limit the rate of API calls, so they are not suitable for high demand, such as a dApp UI constantly scraping blockchain data or an indexer.

To meet the demands of a production dApp, we provide you with our integration layer called Dhatu. :::

	Public endpoint Mandala
Network	Mandala Chain
HTTPS	node1.mandalachain.io
WebSocket	wss://node1.mandalachain.io
Symbol	KPG

Running A Local Network

Now, let's spin up a local network on a standalone node.

Get the Latest Binary

You can obtain the latest binary in one of the following ways:

- Download the latest binary from GitHub.
- Build it from the source.

If you would like to download the binary, visit the Release page of the Madya Mandala Github repository. There, you can find the pre-built binaries for MacOS and Ubuntu, as well as Docker images. If you prefer to build it from the source, this readme can guide you through the process.

After you obtain the binary, you can rename the file to `madya`, and add execution permission by running the following command:

```
chmod +x ./madya-solo
```

You should then be able to execute the binary. To see whether you can run the node, let's check the binary version.

```
./madya --version
# madya-solo xxx
```

Run the Local Network

You are now ready to run the local network, using the following command:

```
./madya-solo --port 30333 --rpc-port 9944 --rpc-cors all --alice
--dev
```

What this command means:

- Use port 30333 for P2P TCP connection
- Use port 9944 for WebSocket/Http connection
- Accept any origin for HTTP and WebSocket connections
- Enable Alice session keys
- Launch network in development mode

You can see the full list of the command options using the `help` subcommand:

```
./madya-solo help
# madya-solo xxx
#
# MadyaSolo <[email protected]>
# Madya solo
# ...
```

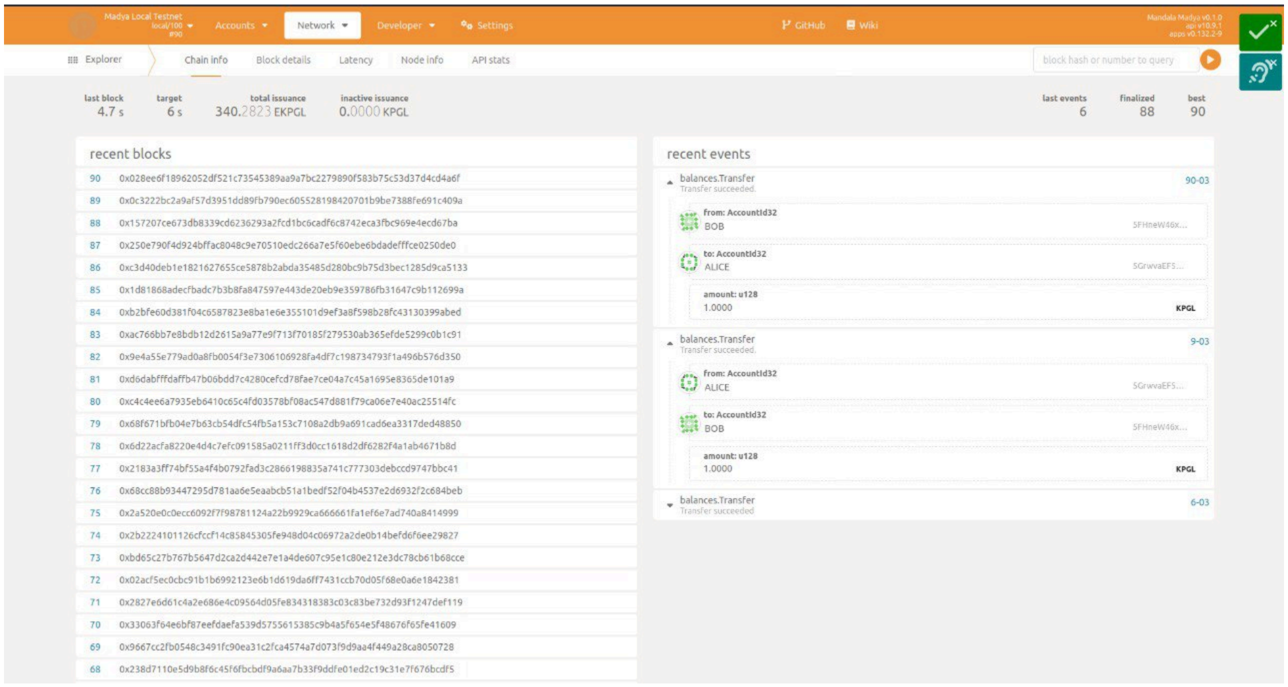
When you have successfully launched the local network, you will see the messages in your terminal.

Access Your Local Network via Polkadot.js Apps Portal

Visit the following URL:

https://polkadot.js.org/apps/?rpc=ws%3A%2F%2F127.0.0.1%3A9944#/explorer

There, you will see the following screen:



last block	target	total issuance	inactive issuance	last events	finalized	best
4.7 s	6 s	340.2823 EKPGL	0.0000 KPGL	6	88	90

recent blocks

90	0x028ee6f18962052df521c73545389aa9a7bc2279890f583b75c53d37d4cd4a6f
89	0x0c3222bc2a9af57d3951dd89fb790ec605528198420701b9be7388fe691c409a
88	0x157207ce673db8339cd6236293a2fcd1bc6cadf6c8742eca3fbc969e4ecd67ba
87	0x250e790f4d924bffac8048c9e70510edc266a7e5f60ebe6bdadefffce0250de0
86	0xc3d40deb1e1821627655ce5878b2abda35485d280bc9b75d3bec1285d9ca5133
85	0x1d81868adecfbadc7b3b8fa847597e443de20eb9e359786fb31647c9b112699a
84	0xb2bfe60d381f04c6587823e8ba1e6e355101d9ef3a8f598b28fc43130399abed
83	0xac766bb7e8bdb12d2615a9a77e9f713f70185f279530ab365efde5299c0b1c91
82	0x9e4a55e779ad0a8fb0054f3e7306106928fa4df7c198734793f1a496b576d350
81	0xd6dabfffdaffb47b06bdd7c4280cefcd78fae7ce04a7c45a1695e8365de101a9
80	0xc4c4ee6a7935eb6410c65c4fd03578bf08ac547d881f79ca06e7e40ac25514fc
79	0x68f671bfb04e7b63cb54dfc54fb5a153c7108a2db9a691cad6ea3317ded48850
78	0x6d22acfa8220e4d4c7efc091585a0211ff3d0cc1618d2df6282f4a1ab4671b8d
77	0x2183a3ff74bf55a4f4b0792fad3c2866198835a741c777303debccd9747bbc41
76	0x68cc88b93447295d781aa6e5eaabcb51a1bedf52f04b4537e2d6932f2c684beb
75	0x2a520e0c0ecc6092f7f98781124a22b9929ca666661fa1ef6e7ad740a8414999
74	0x2b2224101126cfccf14c85845305fe948d04c06972a2de0b14befd6f6ee29827
73	0xbd65c27b767b5647d2ca2d442e7e1a4de607c95e1c80e212e3dc78cb61b68cce
72	0x02acf5ec0cbc91b1b6992123e6b1d619da6ff7431ccb70d05f68e0a6e1842381
71	0x2827e6d61c4a2e686e4c09564d05fe834318383c03c83be732d93f1247def119
70	0x33063f64e6bf87eefdaefa539d5755615385c9b4a5f654e5f48676f65fe41609
69	0x9667cc2fb0548c3491fc90ea31c2fca4574a7d073f9d9aa4f449a28ca8050728
68	0x238d7110e5d9b8f6c45f6fbcbdf9a6aa7b33f9ddfe01ed2c19c31e7f676bcdf5

recent events

balances.Transfer
Transfer succeeded. 90-03
from: AccountId32 BOB 5FHneW46x...
to: AccountId32 ALICE 5GrwvaEF5...
amount: u128 1.0000 KPGL

balances.Transfer
Transfer succeeded. 9-03
from: AccountId32 ALICE 5GrwvaEF5...
to: AccountId32 BOB 5FHneW46x...
amount: u128 1.0000 KPGL

balances.Transfer
Transfer succeeded 6-03

This represents your local network. In this local network, some native tokens have already been issued to a few accounts. Let's visit the Account page from the Accounts Tab.

There, you can see that ALICE and BOB have around 1,000,000 tokens. In the following section, you will deploy your smart contract and interact with it by paying the transaction fees using these tokens.

Test Tokens (Faucets)

A faucet is a dedicated platform where users can obtain test tokens to facilitate development and testing on the Niskala network. In this documentation, we will explore effective methods to acquire KPGT tokens.

How to Get KPGT

To receive testnet tokens, please follow these steps:

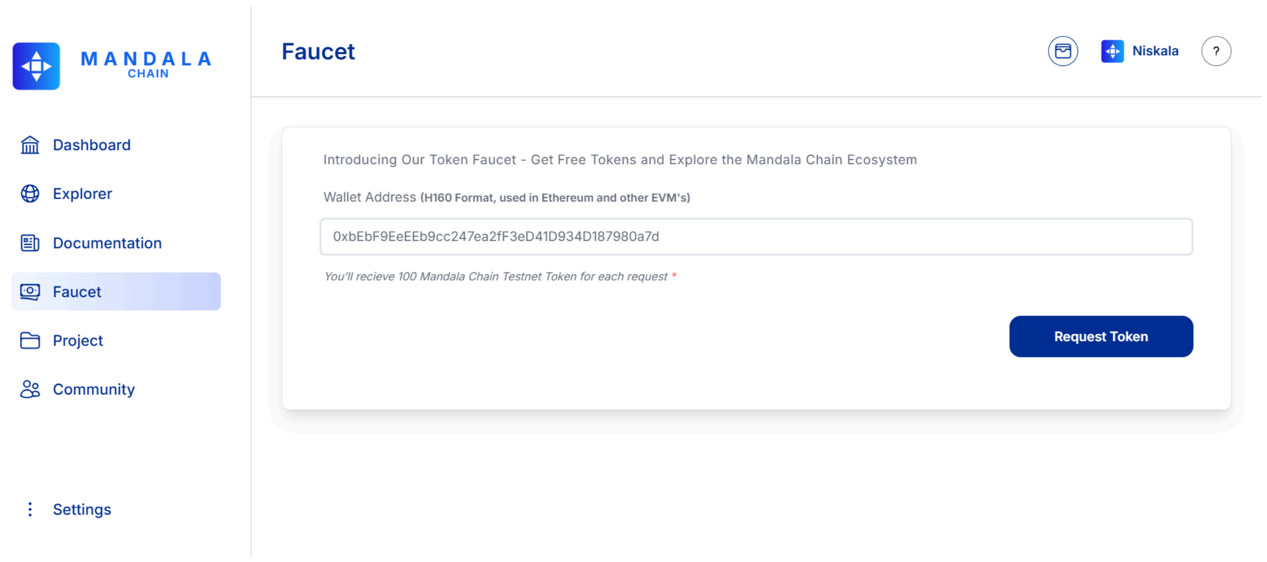
1. Visit the Mandala Chain Website

- Open your web browser
- Enter the URL: hub.mandalachain.io

2. Navigate to the Faucet Page

- Once on the site, look for the **Faucet** menu in the left sidebar
- Click on it to go to the faucet page

3. Enter Your Wallet Address



Introducing Our Token Faucet - Get Free Tokens and Explore the Mandala Chain Ecosystem

Wallet Address (H160 Format, used in Ethereum and other EVM's)

0xbEbF9EeEEb9cc247ea2fF3eD41D934D187980a7d

You'll recieve 100 Mandala Chain Testnet Token for each request *

- On the faucet page, you will find a field labeled **Wallet Address**
- Make sure your wallet address is in H160 format, which is used in Ethereum and other EVMs
- Enter your wallet address into the provided field

4. Request Tokens

- After entering your wallet address, click the **Request Token** button
- A pop-up will appear confirming your request:
    - Pop-up message: "Your request has been successful. You can check the result through the link below."
    - You will see options to **Check** or **Close**

5. Confirm Receipt

- Check your wallet to ensure that the tokens have been received

- You will receive 100 Niskala Testnet Tokens for each request (*this process may take a few minutes*)

Additional Tips

- Make sure your wallet is connected to **Niskala** to receive the tokens
- If you do not receive tokens after some time, try again or check your wallet settings

Important Security Notice: Our team will **NEVER** DM you first, and we **NEVER** ask for your **money** in exchange for faucets!

If someone approaches you pretending to be part of the team, do NOT engage with them. Instead, please use the official Discord faucet channel mentioned above.

WASM Smart Contracts

The Wasm section provides an in-depth exploration of the WebAssembly (Wasm) stack as implemented on the Mandala Chain networks. It delves into advanced topics to enhance your understanding and capabilities in working with Wasm.

Additionally, this section includes a series of comprehensive tutorials designed to guide you through the process of building and deploying Wasm smart contracts, equipping you with the skills needed to leverage the full potential of Wasm on these platforms.

📄 WASM Smart Contract Stack

Smart Contract Runtime Environment

📄 eDSLs

Embedded Domain-Specific Languages (eDSLs), are tools used to improve the blockchain and smart contract development experience by making it easier to write and understand c…

📄 Ink! Development

Ink! is a Rust eDSL developed by Parity. It specifically targets smart contract development for Substrate’s pallet-contracts.

🗃 Development Tutorials

2 items

WASM Smart Contract Stack

Smart Contract Runtime Environment

Mandala Chain runtimes are based on Substrate, and both networks incorporate `pallet-contracts,` a sandboxed environment used to deploy and execute WebAssembly (WASM) smart contracts. Any language that compiles to WASM may be deployed and run on this Wasm Virtual Machine. However, the code should be compatible with the `pallet-contracts` API.

To avoid unnecessary complexity, and writing boilerplate code, the most appropriate method of building will involve the use of an eDSL specifically targeting `pallet-contracts`, such as ink! (based on Rust), or ask! (based on AssemblyScript), or possibly others as the ecosystem grows.

After compilation, a WASM blob can then be deployed and stored on-chain.

Execution Engine

Pallet-contracts uses wasmi as a Wasm interpreter to execute WASM smart contract blobs. Although there is a faster JIT interpreter such as wasmtime available in the native runtime, smart contracts are an untrusted environment that requires a higher degree of correctness of interpretation, which makes *wasmi* a more suitable option.

Two-step Deployment of Contracts

The contract code (WASM blob), and contract address and storage are decoupled from one another, so we require two steps to deploy a new contract on-chain:

1. First, upload the Wasm contract on-chain (every contract Wasm code has a `code_hash` as an identifier).
2. Second, instantiate the contract - it will create an address and storage for that contract.
3. Anyone can instantiate a contract based on its `code_hash`.

There are several benefits of decoupling the contract code from the address/storage:

- To save space on-chain. Since a contract can have several constructors and instantiations, a redeployment will create a new instance based on the same underlying code. Think about standardized tokens, like PSP22 & PSP34, that will have one `code_hash` & `blob` living on-chain, and as many instantiations as needed, rather than uploading code with each new instantiation (for example, on Ethereum).
- To instantiate a new contract using code within an existing smart contract (see the delegator example), `code_hash` is all that is needed.
- Some standard contracts, such as (PSP22 and PSP34) will only be uploaded on-chain once, preventing users from having to pay gas costs for uploading new code.
- Update contract code for an address: replace the contract code at the specified address with a new code (see set_code_hash). Storage and balances will be preserved.

For More Information About `pallet-contracts`

- `pallet-contracts` in Rust docs
- `pallet-contracts` on Github

Client APIs

The only library available to communicate with smart contracts is Polkadot.js API.

> INFO
>
> This API provides application developers the ability to query a node and interact with the Polkadot or Substrate chains using Javascript.

Parity also developed a web application to interact with contracts called contracts-ui.

The Wasm Stack vs. Ethereum

	Ethereum	Mandala Chain
L1 Architecture	Ethereum clients	Substrate (will support EVM)
Smart Contract Runtime Environment	EVM	WASM pallet-contract + EVM frontier
Gas Model	fixed price per instruction	weight + storage fees + loading fees
Smart Contract DSLs	Solidity and Vyper	ink! (Rust), ask! (AssemblyScript), Solidity, and Vyper
Standards	EIPs	PSPs and EIPs

eDSLs

Embedded Domain-Specific Languages (eDSLs), are tools used to improve the blockchain and smart contract development experience by making it easier to write and understand code. EDSLs are programming languages or libraries that are designed to be used within the context of another programming language to provide a more expressive and intuitive way to write smart contracts. In other words, an eDSL allows developers to write smart contracts at a higher level, which makes the code easier to read and interpret and less prone to error.

For example, instead of using pure Rust to write Wasm smart contracts or blockchain logic, a Rust eDSL, such as Substrate, can be used instead as an eDSL specifically targeting development within those domains. Substrate allows developers to express the intent of their code in a more natural way, making it easier to understand and maintain.

EDSLs can also provide features such as error checking, debugging, and testing, which can further improve the development experience within the realm of their specific domains.

`Ink!`

Ink! is an eDSL written in Rust and developed by Parity. It specifically targets Substrate's `pallet-contracts` API.

Ink! offers Rust procedural macros and a list of crates to help facilitate development and save time by avoiding boilerplate code.

Check out the official documentation here and `Ink!` GitHub repo here.

`Ask!`

Ask! is a framework for AssemblyScript developers that allows them to write Wasm smart contracts for `pallet-contracts`. Its syntax is similar to TypeScript.

This project is funded by the Polkadot treasury - link here, and is still under development.

Check out the official GitHub here.

Ink! Development

Ink! is a Rust eDSL developed by Parity. It specifically targets smart contract development for Substrate's `pallet-contracts`.

Ink! offers Rust procedural macros and a list of crates to facilitate development and allows developers to avoid writing boilerplate code.

It is currently the most widely supported eDSL, and will be highly supported in the future. (by Parity and builders community).

Ink! offers a broad range of features such as:

- idiomatic Rust code
- Ink! Macros & Attributes - #[ink::contract]
- `Trait` support
- Upgradeable contracts - Delegate Call
- Chain Extensions (interact with Substrate pallets inside a contract)
- Off-chain Testing - `#[ink(test)]`

Installation procedures are available in ink! Environment section.

Documentation

- Ink! Github repo
- Ink! Intro repo
- Ink! Official Documentation
- Ink! Rust doc

Development Tutorials

Overview

In the following sections you can find hands-on tutorials designed to help developers get the most out of Mandala Chain Network.

Prerequisites

This tutorial targets developers with no experience in ink! and a **basic** level in Rust.

Tutorial	Difficulty
NFT contract with PSP34	Intermediate ink! - Basic Rust
Implement Uniswap V2 core DEX	Advanced ink! - Basic Rust

To follow this tutorial, you will need:

- To set up your ink! environment.
- Basic Rust knowledge. Learn Rust.

What will we do?

In this tutorial, we will implement the most basic contract: Flipper in ink!.

What will we use?

- ink! 4.0.0

What will you learn?

- Anatomy of an ink! contract
- Define contract storage
- Callable functions
- Unit test your contract

Basic Contract

Each contract should be in its **own crate**. In a folder, create two files:

- Cargo.toml: The manifest.
- lib.rs: The default library file.

Inside the Cargo.toml you will need to specify parameters in the `[package]`, `[dependencies]`, `[lib]` type, and `[features]` sections:

```
[package]
name = "my_contract"
version = "0.1.0"
authors = ["Your Name <[email protected]>"]
edition = "2021"

[dependencies]
ink = { version = "4.0.0", default-features = false}
ink_metadata = { version = "4.0.0", features = ["derive"],
optional = true }

scale = { package = "parity-scale-codec", version = "3", default-
features = false, features = ["derive"] }
scale-info = { version = "2.3", default-features = false, features
= ["derive"], optional = true }

[lib]
path = "lib.rs"

[features]
default = ["std"]
std = [
    "ink/std",
```

In the library file - ink! has a few minimum requirements:

- `#![cfg_attr(not(feature = "std"), no_std)]` at the beginning of each contract file.
- a module with `#[ink::contract]`.
- a (storage) struct - that can be empty - with `#[ink(storage)]`.
- at least one constructor with `#[ink(constructor)]`.
- at least one fn with `#[ink(message)]`.

In the lib.rs the minimum implementation is:

```
#![cfg_attr(not(feature = "std"), no_std)]

#[ink::contract]
mod my_contract {

    #[ink(storage)]
    pub struct MyContract {}

    impl MyContract {
        #[ink(constructor)]
        pub fn new() -> Self {
            Self {}
        }

        #[ink(message)]
        pub fn do_something(&self) {
            ()
        }
    }
}
```

The flipper smart contract is most basic example provided by ink! team.

Flipper Contract

This is step-by-step explanation of the process behind building an ink! smart contract, using a simple app called Flipper. The examples provided within this guide will help you develop an understanding of the basic elements and structure of ink! smart contracts.

What is Flipper?

Flipper is a basic smart contract that allows the user to toggle a boolean value located in storage to either `true` or `false`. When the flip function is called, the value will change from one to the other.

Prerequisites

Please refer to the previous section for the list of prerequisites.

Flipper Smart Contract

In a new project folder, execute the following:

```
$ cargo contract new flipper # flipper is introduced from the
beginning.
```

```
$ cd flipper/
$ cargo contract build #build flipper app
```

💡 If you receive an error such as:

```
ERROR: cargo-contract cannot build using the "stable" channel.
Switch to nightly.
```

Execute:

```
$ rustup default nightly
```

to reconfigure the default Rust toolchain to use the nightly build, or

```
$ cargo +nightly contract build
```

to use the nightly build explicitly, which may be appropriate for developers working exclusively with ink!

Once the operation has finished and the Flipper project environment has been initialized, we can perform an examination of the file and folder structure. Let's dive a bit deeper into the project structure:

The File Structure of Flipper

- `target`: Contains build / binary information.
- `Cargo.lock`: The lock file for the dependency package.
- `Cargo.toml`: The package configuration.
- `lib.rs`: The contract logic.

Flipper Contract `lib.rs`

```
#![cfg_attr(not(feature = "std"), no_std)]

#[ink::contract]
```

Contract Structure `lib.rs`

```
#![cfg_attr(not(feature = "std"), no_std)]

use ink_lang as ink;

#[ink::contract]
mod flipper {

        // This section defines storage for the contract.
    #[ink(storage)]
    pub struct Flipper {
    }

        // This section defines the functional logic of the
contract.
    impl Flipper {
    }

        // This section is used for testing, in order to verify
contract validity.
    #[cfg(test)]
    mod tests {
    }
}
```

Storage

```
    #[ink(storage)]
    pub struct Flipper {

    }
```

This annotates a struct that represents the **contract's internal state.** (details):

```
#[ink(storage)]
```

Storage types:

- Rust primitives types
  - `bool`
  - `u{8,16,32,64,128}`
  - `i{8,16,32,64,128}`
  - `String`
- Substrate specific types
  - `AccountId`
  - `Balance`
  - `Hash`
- ink! storage type
  - `Mapping`
- Custom data structure details

This means the contract (Flipper) stores a single `bool` value in storage.

```
#[ink(storage)]
pub struct Flipper {
    value: bool,
}
```

Callable Functions

At the time the contract is deployed, a constructor is responsible for **bootstrapping the initial state** into storage. For more information.

```
#[ink(constructor)]
```

The addition of the following function will initialize `bool` to the specified `init_value`.

```
#[ink(constructor)]
pub fn new(init_value: bool) -> Self {
    Self { value: init_value }
}
```

Contracts can also contain multiple constructors. Here is a constructor that assigns a default value to `bool`. As in other languages, the default value of `bool` is `false`.

```
#[ink(constructor)]
pub fn default() -> Self {
    Self::new(Default::default())
}
```

The following will permit a function to be **publicly dispatchable**, meaning that the function can be called through a message, which is a way for contracts and external accounts to interact with the contract. Find more information here). Note that all public functions **must** use the `#[ink(message)]` attribute.

```
#[ink(message)]
```

The `flip` function modifies storage items, and `get` function retrieves a storage item.

```
#[ink(message)]
pub fn flip(&mut self) {
    self.value = !self.value;
```

💡 If you are simply *reading* from contract storage, you will only need to pass `&self`, but if you wish to *modify* storage items, you will need to explicitly mark it as mutable `&mut self`.

```
impl Flipper {

        #[ink(constructor)]
        pub fn new(init_value: bool) -> Self {
            Self { value: init_value }
        }

        /// Constructor that initializes the `bool` value to
`false`.
        ///
        /// Constructors can delegate to other constructors.
        #[ink(constructor)]
        pub fn default() -> Self {
            Self::new(Default::default())
        }

        /// A message that can be called on instantiated
contracts.
        /// This one flips the value of the stored `bool` from
`true`
        /// to `false` and vice versa.
        #[ink(message)]
        pub fn flip(&mut self) {
            self.value = !self.value;
        }

        /// Simply returns the current value of our `bool`.
        #[ink(message)]
        pub fn get(&self) -> bool {
            self.value
        }
    }
```

Test

```
#[cfg(test)]
    mod tests {
        /// Imports all the definitions from the outer scope so we
can use them here.
        use super::*;

        /// Imports `ink_lang` so we can use `#[ink::test]`.
        use ink_lang as ink;

        /// We test if the default constructor does its job.
        #[ink::test]
        fn default_works() {
            let flipper = Flipper::default();
            assert_eq!(flipper.get(), false);
        }

        /// We test a simple use case of our contract.
        #[ink::test]
        fn it_works() {
            let mut flipper = Flipper::new(false);
            assert_eq!(flipper.get(), false);
            flipper.flip();
            assert_eq!(flipper.get(), true);
        }
    }
```

Compile, Deploy, and Interact with Contracts

Follow this guide to deploy your contract using Polkadot UI. Once deployed, you will be able to interact with it.

EVM Smart Contracts

The EVM smart contracts section provides guides on how to develop and deploy EVM Smart Contracts on Mandala Chain networks. It delves into advanced topics to enhance your understanding and capabilities in working with Wasm.

Additionally, this section includes a series of comprehensive tutorials designed to guide you through the process of building and deploying EVM smart contracts to Mandala Chain, making it easy for any developers who already familiar with EVM ecosystem to build on Mandala Chain.

Your First Contract

Introduction

In this tutorial, you will learn how to develop and deploy a smart contract on Niskala.

This tutorial is aimed at those who have never touched Mandala Chain before, and want to get basic, but comprehensive, understanding of deploying smart contract on Niskala.

This tutorial should take about 30 minutes, after which you will officially be a dApp developer! To complete this tutorial, you will need:

- Basic knowledge of Solidity and Polkadot;
- Familiarity with software development tools and CLIs;
- Basic knowledge about MetaMask;
- Basic knowledge about javascript frontend web framework (Vue);

What you will do in this Tutorial:

1. Learn about Niskala.
2. Run a Collator node as a local development node.
3. Configure MetaMask.
4. Obtain some test tokens.

Configure MetaMask

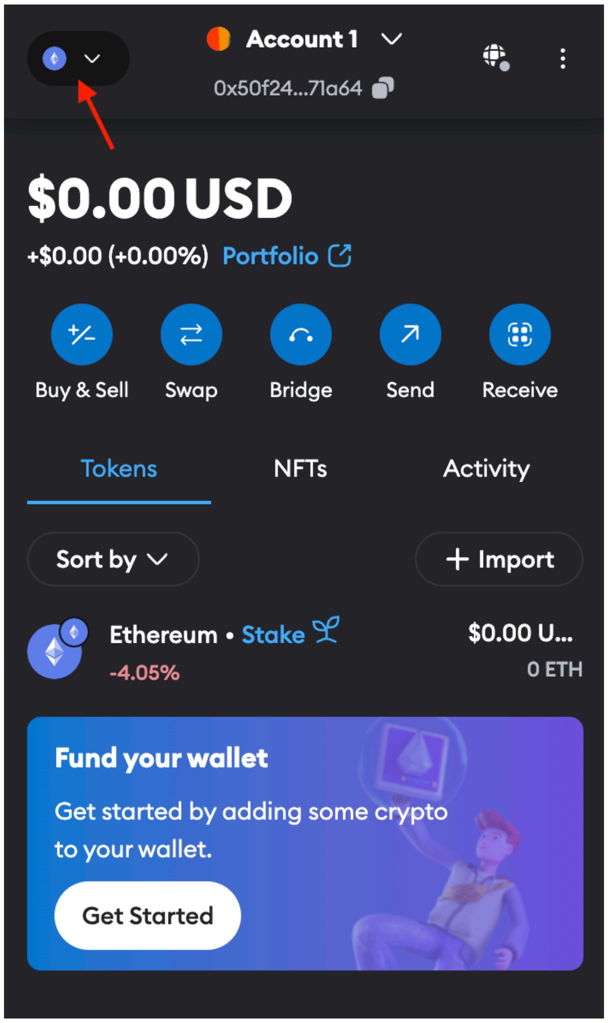
Add network to MetaMask

INFO

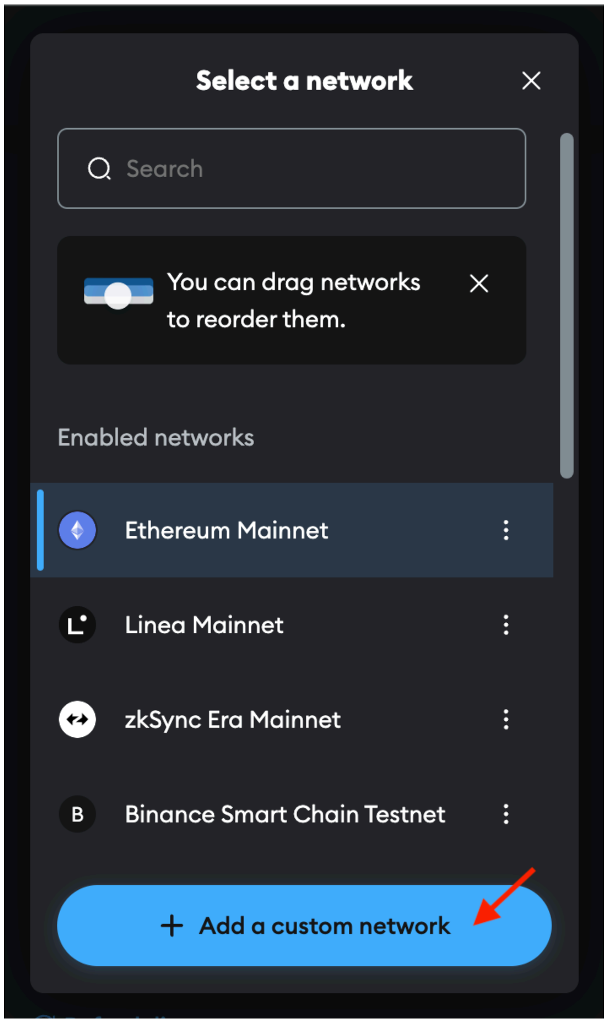
Note: Before following the instructions below, ensure that you have already installed MetaMask on your browser. If not, follow the installation guide here.

This guide provides a step-by-step process to connect MetaMask to the Niskala Network.

1. **Open MetaMask**
   Click on the MetaMask icon in your browser to open the extension.



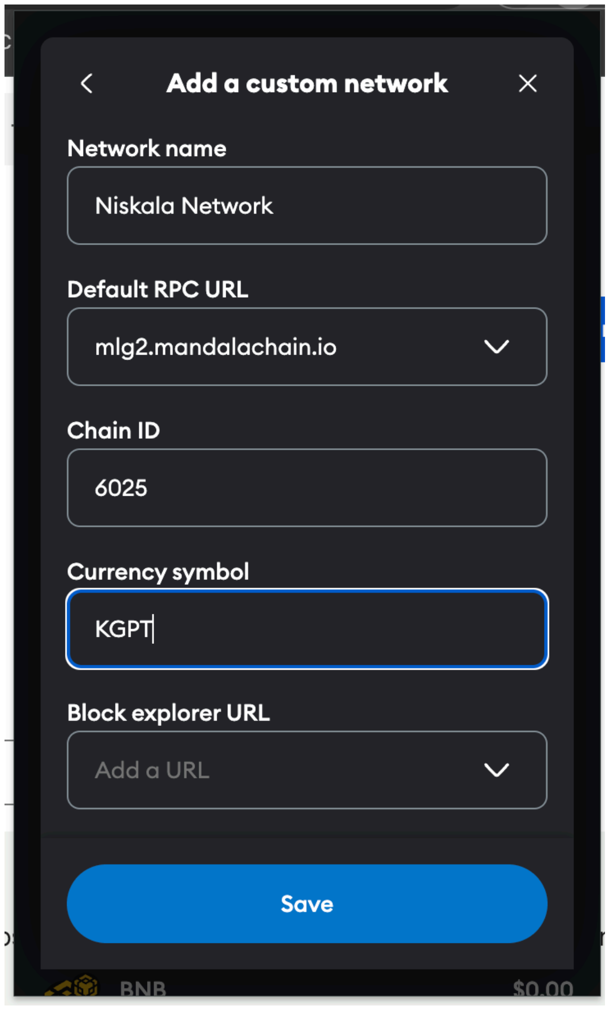
2. **Add a Custom Network**
   Click on the network dropdown icon in the top-left corner of MetaMask, then select the `Add a custom network` button.



3. **Enter Niskala Network Details**
   Fill in the following details in the custom network form:

Property	Value
Network	Niskala Network
HTTPS	https://mlg1.mandalachain.io/ (testnet) https://mlg2.mandalachain.io/ (devnet)

Property	Value
Chain ID	6025
Symbol	KPGT



Network name
Niskala Network

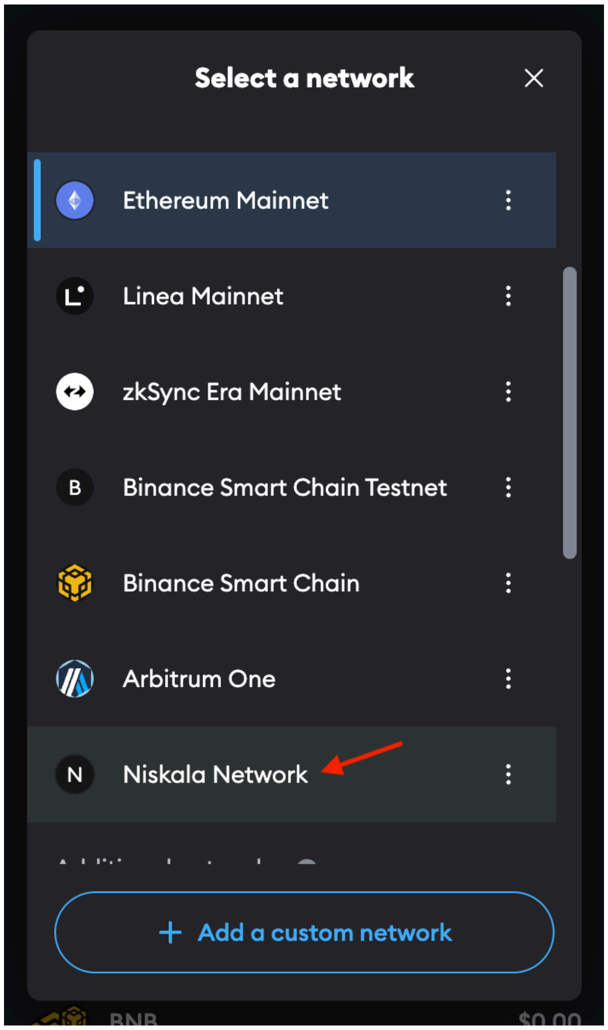
Default RPC URL
mlg2.mandalachain.io

Chain ID
6025

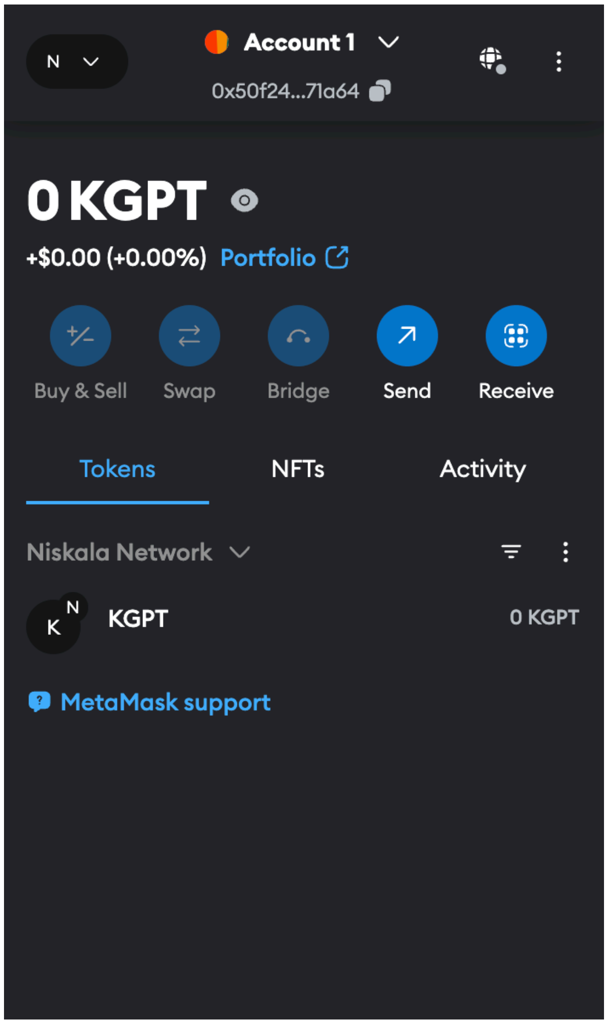
Currency symbol
KGPT

Block explorer URL
Add a URL

4. **Switch to Niskala Network**
   After saving, select `Niskala Network` from the network dropdown menu to switch to it.



5. **Manage Your Account**
   You can now view your account balance, send transactions, and interact with dApps on the Niskala Network.


N
Account 1
0x50f24...71a64
0 KGPT
+$0.00 (+0.00%) Portfolio
Buy & Sell
Swap
Bridge
Send
Receive
Tokens
NFTs
Activity
Niskala Network
KGPT
0 KGPT
MetaMask support

Developer Tooling

Deploying and interacting with EVM-based smart contracts on Mandala Chain is as easy as any other EVM-compatible network. Getting started requires just two steps:

- Configuring (and funding) your Ethereum account on the respective network.
- Adding Mandala networks to your Ethereum client.

> **INFO**
>
> In this tutorial, we will use Niskala Testnet for demonstration purpose

Foundry

Introduction

Foundry is a comprehensive smart contract development toolchain created by Paradigm. It simplifies the process of building, testing, and deploying decentralized applications (dApps) on Ethereum and other EVM-compatible chains like Mandala.

Set Up

We recommend you to follow guide from Foundry Book for installation.

To verify that the installation, run following commands

```
forge --version
cast --version
anvil --version
```

You should see following output

```
forge 0.2.0 (6b07c77 2024-12-18T00:22:12.633674000Z)
cast 0.2.0 (6b07c77 2024-12-18T00:22:12.615412000Z)
anvil 0.2.0 (6b07c77 2024-12-18T00:22:12.584294000Z)
```

Initializing your project

If you're starting your Foundry project from scratch, we recommend you read the Foundry Book.

Contract Environment

Block Explorer

Overview

Verify Smart Contract

The EVM block explorer for all Mandala networks (Mandala, Niskala) is Blockscout and you can verify a smart contract by following Blockscout documentation for contract verification.

Block Explorer

Overview

Block explorers provide functionality like Google for blockchain, enabling developers and users to access information such as balances, contracts, tokens, transactions, and API services.

Explorers

Niskala Testnet

BlockScout is the best explorer for developers who are building in our EVM environment, it has all the features as EtherScan.

Blockscout: https://niskala.mandalachain.io/

Polkadot JS https://polkadot.js.org/apps/?rpc=wss%3A%2F%2Fmlg2.mandalachain.io#/explorer

Verify Smart Contract

The EVM block explorer for all Mandala networks (Mandala, Niskala) is **Blockscout** and you can verify a smart contract by following Blockscout documentation for contract verification.

FAQ

What are the names of the native tokens in the Mandala ecosystem?

KPGT (Niskala - Testnet tokens)

KPG (Utama Network - Mainnet tokens)

How do I connect to Mandala networks, RPCs, Network name, Chain ID?

You can visit this page.

How can I get test tokens (KPGT)?

To receive testnet tokens:

1. Join our Discord server: https://discord.com/invite/MwUQgQZgxm
2. Navigate to the dedicated faucet channel
3. Use the command `/drip <your_public_key>` in the chat

Is it possible to import Substrate (Polkadot) addresses to Metamask?

No. Polkadot (Substrate framework) uses a 256 bit address, while Metamask uses a 160 bit address.

Run a Node

Node Commands

The following sections summarize the commands of Mandala Chain nodes you need for different cases. For any more details, you can consult the help page:

Run a Collator Node

This guide provides a detailed explanation of collator nodes within the Mandala Network and step-by-step instructions for setting up and running a collator node.

Run a Full Node

Overview

Run a Validator Node

Overview

Run an Archive Node

Overview

Validator Requirement

How to become a validator

Node Commands

The following sections summarize the commands of Mandala Chain nodes you need for different cases. For any more details, you can consult the help page:

```
madya-solo --help
```

Collator

Binary service file

```
[Unit]
Description=Madya Solo

[Service]
User=madya
Group=madya

ExecStart=/usr/local/bin/madya-solo \
  --pruning archive \
  --validator \
  --name {VALIDATOR_NAME} \
  --chain madya \
  --base-path /var/lib/madya \
  --trie-cache-size 0 \
  --telemetry-url 'wss://telemetry.polkadot.io/submit/ 0' \

Restart=always
RestartSec=10
```

```
[Unit]
Description=Vraja Solo

[Service]
User=madya
Group=madya

ExecStart=/usr/local/bin/madya-solo \
  --pruning archive \
  --validator \
  --name {VALIDATOR_NAME} \
  --chain solo\
  --base-path /var/lib/niskala\
  --trie-cache-size 0 \
  --telemetry-url 'wss://telemetry.polkadot.io/submit/ 0' \

Restart=always
RestartSec=10

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

Docker

```
docker run -d \
--name madya-container \
-u $(id -u madya):$(id -g madya) \
-p 30333:30333 \
-v "/var/lib/madya/:/data" \
mandalachain/madya-solo:latest \
madya-solo \
--pruning archive \
--validator \
--name {VALIDATOR_NAME} \
docker run -d \
--name vraja-container \
-u $(id -u madya):$(id -g madya) \
-p 30333:30333 \
-v "/var/lib/madya/:/data" \
mandalachain/madya-solo:latest \
madya-solo \
--pruning archive \
--validator \
--name {VALIDATOR_NAME} \
--chain niskala \
--base-path /data \
--trie-cache-size 0 \
--telemetry-url 'wss://telemetry.polkadot.io/submit/ 0' \
```

Archive node as RPC endpoint

Binary

```
[Unit]
Description=Madya Solo node

[Service]
User=madya
Group=madya

ExecStart=/usr/local/bin/madya-solo \
  --pruning archive \
  --rpc-cors all \
  --name {NODE_NAME} \
  --chain madya \
  --base-path /var/lib/madya \
  --rpc-external \
[Unit]
Description=Niskala Archive node

[Service]
User=madya
Group=madya

ExecStart=/usr/local/bin/madya-solo \
  --pruning archive \
  --rpc-cors all \
  --name {NODE_NAME} \
  --chain niskala \
  --base-path /var/lib/niskala \
  --rpc-external \
  --rpc-methods Safe \
  --rpc-max-request-size 1 \
  --rpc-max-response-size 1 \
  --telemetry-url 'wss://telemetry.polkadot.io/submit/ 0' \

Restart=always
RestartSec=10

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

Docker

```
docker run -d \
--name madya-container \
-u $(id -u madya):$(id -g madya) \
-p 30333:30333 \
-p 9944:9944 \
-v "/var/lib/madya/:/data" \
mandalachain/madya-collator:latest \
docker run -d \
--name vraja-container \
-u $(id -u madya):$(id -g madya) \
-p 30333:30333 \
-p 9944:9944 \
-v "/var/lib/madya/:/data" \
mandalachain/madya-collator:latest \
madya-collator \
--pruning archive \
--rpc-cors all \
--name {NODE_NAME} \
--chain niskala \
--base-path /data \
--rpc-external \
--rpc-methods Safe \
--rpc-max-request-size 1 \
--rpc-max-response-size 1 \
--telemetry-url 'wss://telemetry.polkadot.io/submit/ 0' \
```

Specific cases command args

External monitoring

```
--prometheus-external
```

Full command documentation

To see full node command binary embedded documentation, please use the help option.

```
$ ./madya-solo -h
```

\

Run a Collator Node

This guide provides a detailed explanation of collator nodes within the Mandala Network and step-by-step instructions for setting up and running a collator node.

Overview

What is a Collator Node?

Collator node serves a critical role in the Mandala Network, where it produces blocks on parachain. Collator node collects user transactions and aggregates them into block candidates. These blocks are then provided to the Relay Chain validators along with **Proofs of Validity (PoV)**, ensuring that the proposed blocks meet the network's rules and standards. Validators then will be responsible for the finalization.

Collators maintain the parachain's state and ensure smooth communication between the parachain and the Relay Chain.

> **INFO**
>
> Collators do not provide security guarantees for parachains. Instead, they rely on the Relay Chain's security model to finalize blocks. For more info learn here

---

Role of Collator Node

Collator nodes serve several essential functions in the Mandala Network:

- **Block Production**: Aggregate transactions into valid block candidates and propose them to Relay Chain validators.

- **State Maintenance**: Maintain the full state of the parachain to ensure validators can verify blocks efficiently.
- **Network Bridging**: Act as intermediaries between parachains and the Relay Chain, facilitating cross-chain communication.
- **Consensus Participation**: Operate within the Aura consensus mechanism to produce blocks deterministically while leaving finality to validators using GRANDPA.

In Mandala's non-permissionless setup, only pre-approved collators are allowed, ensuring network security and reliability.

Getting Started

Project Setup

1. Clone the Mandala-Node repository.

```
git clone https://github.com/MandalaChain/Mandala-Node.git
```

2. Navigate to the directory

```
cd Mandala-Node
```

Requirement

To run Mandala-Node, you will need to have Rust installed in your machine. Depending on your operating system and Rust version, there might be additional packages required to compile the node.

If you haven't had rust on your machine or encounter version error, run following scripts:

```
.maintain/scripts/install-rust-toolchain.sh
```

Successful stup will produce following output:

```
installed toolchains
--------------------

stable-aarch64-apple-darwin (default)
nightly-aarch64-apple-darwin
1.75.0-aarch64-apple-darwin
solana

installed targets for active toolchain
--------------------------------------

aarch64-apple-darwin
wasm32-unknown-unknown

active toolchain
----------------

stable-aarch64-apple-darwin (default)
rustc 1.83.0 (90b35a623 2024-11-26)

Default host: aarch64-apple-darwin
rustup home:  /Users/[USERNAME]/.rustup

installed toolchains
--------------------

stable-aarch64-apple-darwin (default)
nightly-aarch64-apple-darwin
1.75.0-aarch64-apple-darwin
```

Running a node

1. Download Zombienet Zombienet is used to run Mandala Node on development chain since it requires a relay chain to connect to. Later we will spin up a 2-node rococo local testnet instance using zombienet.

   ```
   .maintain/scripts/download-zombienet.sh
   ```

   If successful, you will find *zombienet* binaries on these location `.maintain/zombienet/binaries`

2. Compile Mandala Polkadot

   ```
   .maintain/scripts/compile-mandala-polkadot.sh
   ```

   INFO

   Compiling the Mandala Polkadot takes a significant amount of resource and time. On Macbook Air Apple M1 (2020) it took approximately 2 hours for the compilation to finish. Please wait patiently and let the code cook.

   If successful, you will find *mandala* binaries on these location `.maintain/zombienet/binaries`

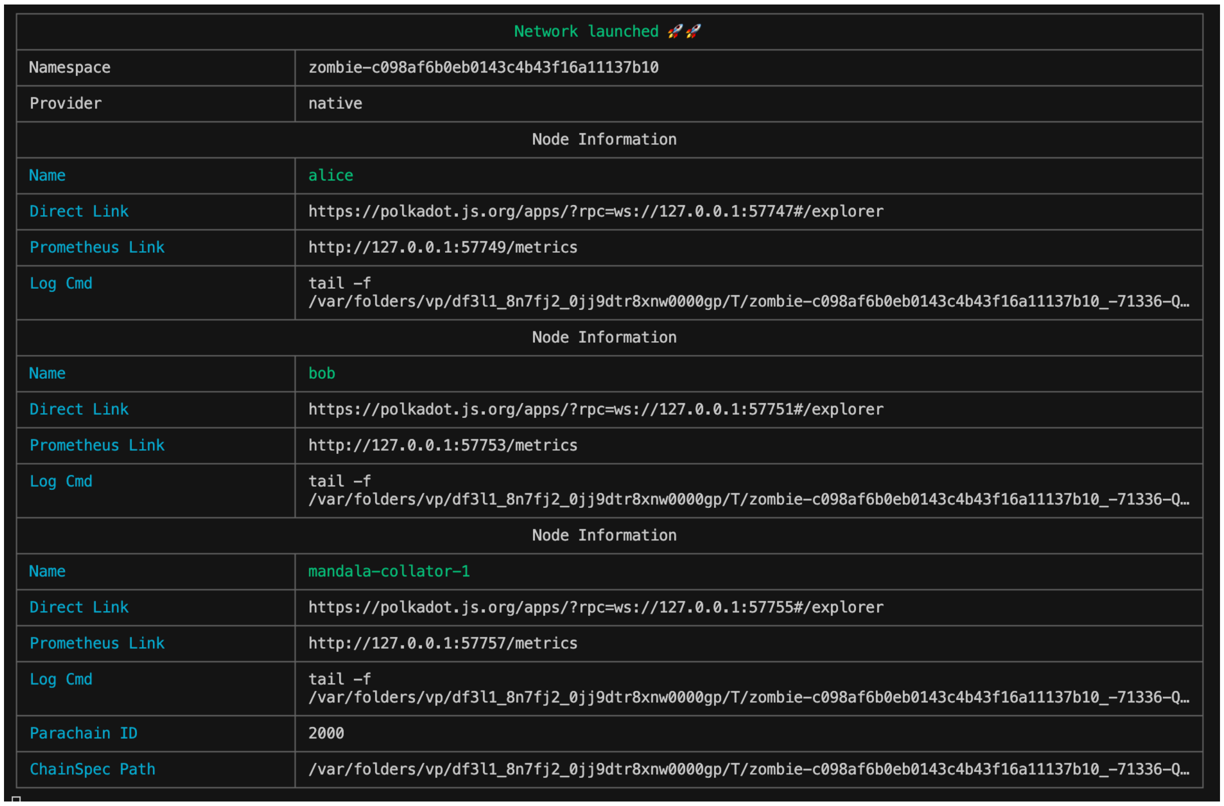
3. Start the network Run following commands:

   ```
   .maintain/scripts/start-zombienet.sh <chain-type>
   ```

   There are 4 chain types available to run

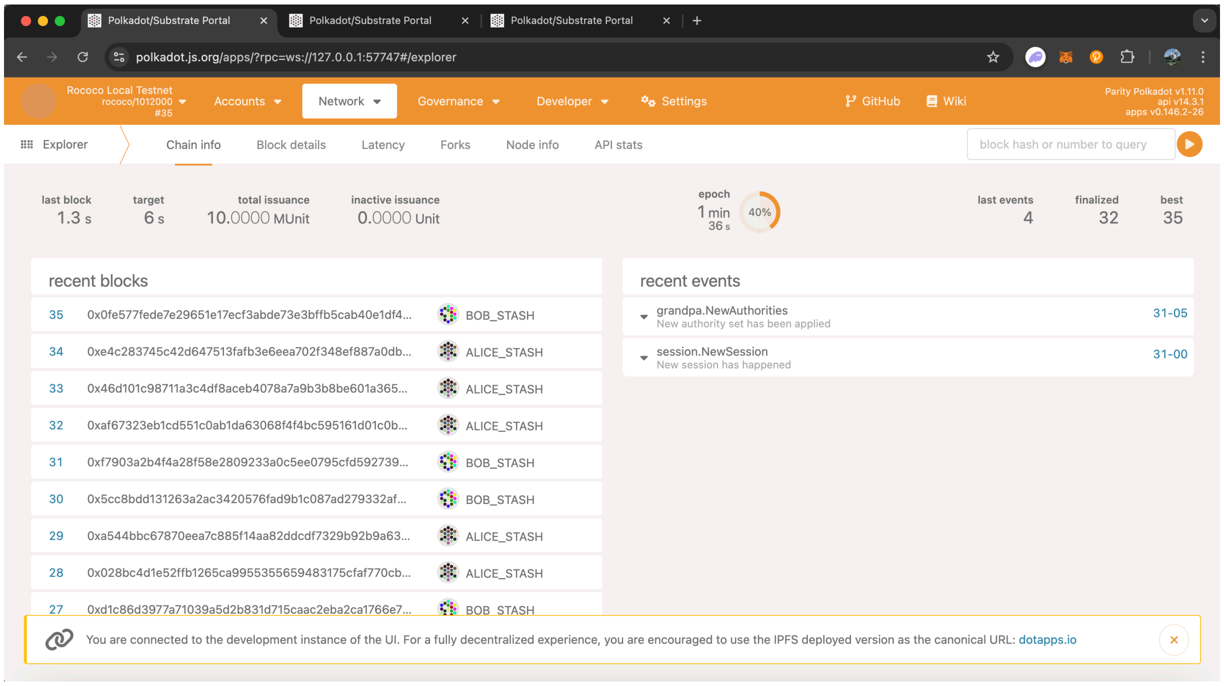
    - `local`: Local development chain (Mandala)
    - `mainnet`: Production chain (Mandala)
    - `dev`: Development chain (Niskala)
    - `live`: Live network (Niskala)

   Successful run will produce following result:



```
                              Network launched 🚀🚀
Namespace          zombie-c098af6b0eb0143c4b43f16a11137b10
Provider           native
                              Node Information
Name               alice
Direct Link        https://polkadot.js.org/apps/?rpc=ws://127.0.0.1:57747#/explorer
Prometheus Link    http://127.0.0.1:57749/metrics
Log Cmd            tail -f
                   /var/folders/vp/df3l1_8n7fj2_0jj9dtr8xnw0000gp/T/zombie-c098af6b0eb0143c4b43f16a11137b10_-71336-Q…
                              Node Information
Name               bob
Direct Link        https://polkadot.js.org/apps/?rpc=ws://127.0.0.1:57751#/explorer
Prometheus Link    http://127.0.0.1:57753/metrics
Log Cmd            tail -f
                   /var/folders/vp/df3l1_8n7fj2_0jj9dtr8xnw0000gp/T/zombie-c098af6b0eb0143c4b43f16a11137b10_-71336-Q…
                              Node Information
Name               mandala-collator-1
Direct Link        https://polkadot.js.org/apps/?rpc=ws://127.0.0.1:57755#/explorer
Prometheus Link    http://127.0.0.1:57757/metrics
Log Cmd            tail -f
                   /var/folders/vp/df3l1_8n7fj2_0jj9dtr8xnw0000gp/T/zombie-c098af6b0eb0143c4b43f16a11137b10_-71336-Q…
Parachain ID       2000
ChainSpec Path     /var/folders/vp/df3l1_8n7fj2_0jj9dtr8xnw0000gp/T/zombie-c098af6b0eb0143c4b43f16a11137b10_-71336-Q…
```

4. Connect to Polkadot.JS Front-End You can view the network detailed information by connecting it to the Polkadot.JS Front-End. Follow the link given that showed up when running the network.



last block	target	total issuance	inactive issuance	epoch	last events	finalized	best
1.3 s	6 s	10.0000 MUnit	0.0000 Unit	1 min 36 s (40%)	4	32	35

recent blocks

35	0x0fe577fede7e29651e17ecf3abde73e3bffb5cab40e1df4...	BOB_STASH
34	0xe4c283745c42d647513fafb3e6eea702f348ef887a0db...	ALICE_STASH
33	0x46d101c98711a3c4df8aceb4078a7a9b3b8be601a365...	ALICE_STASH
32	0xaf67323eb1cd551c0ab1da63068f4f4bc595161d01c0b...	ALICE_STASH
31	0xf7903a2b4f4a28f58e2809233a0c5ee0795cfd592739...	BOB_STASH
30	0x5cc8bdd131263a2ac3420576fad9b1c087ad279332af...	BOB_STASH
29	0xa544bbc67870eea7c885f14aa82ddcdf7329b92b9a63...	ALICE_STASH
28	0x028bc4d1e52ffb1265ca9955355659483175cfaf770cb...	ALICE_STASH
27	0xd1c86d3977a71039a5d2b831d715caac2eba2ca1766e7...	BOB_STASH

recent events

grandpa.NewAuthorities New authority set has been applied	31-05
session.NewSession New session has happened	31-00

You are connected to the development instance of the UI. For a fully decentralized experience, you are encouraged to use the IPFS deployed version as the canonical URL: dotapps.io

TIP

- Interacting with the blockchain requires Polkadot.JS account. If you didn't have one yet, follow this article
- It's recommended to use Google Chrome for accessing the Polkadot.JS app

Congratulation! You've successfully run Mandala collator node. If you find any difficulties during the process, feel free to contact our support team in Discord or Telegram

Run a Full Node

Overview

Running a full node on a MandalaChain chain allows you to connect to the network, sync with a bootnode, obtain local access to RPC endpoints, author blocks on the parachain, and more.

Different from the archive node, a full node discards all finalized blocks older than the configured number of blocks (256 blocks by default). A full node occupies less storage space than an archive node because of pruning.

A full node may eventually be able to rebuild the entire chain with no additional information and become an archive node, but at the time of writing, this is not implemented. If you need to query historical blocks past that you have pruned, you need to purge your database and resync your node, starting in archive mode. Alternatively, you can use a backup or snapshot of a trusted source to avoid needing to sync from genesis, thus only requiring the blocks past that snapshot.

If your node needs to provide old historical blocks' data, please consider using the Archive node instead.

Requirements

Requirements for running any node are similar to what we recommend for the Archive node. Read more about this here. Note that the Full node requires less disk space. The hard disk requirement for the Archive node is not applied to Full nodes.

To set a full node, you need to specify the number of blocks to be pruned:

```
--pruning 1000 \
```

Run a Validator Node

Overview

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 tokens 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 accumulating rewards.

> DANGER
>
> It is highly recommended that you have significant system administration experience before attempting to run your own validator.
>
> You must be able to handle technical issues and anomalies with your node, which you must be able to tackle yourself. Being a validator involves more than just executing the Mandala Chain binary.

Since security is essential 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. As you progress as a validator, you will likely want to use this repository as a *starting point* for any modifications and customizations.

If you need help, please contact the ***Mandala Technical Support*** on Telegram. The team and other validators are there to help answer questions and provide tips from experience.

Role of validators in the Mandala Chain ecosystem

Validators maintain our ecosystem by collecting user transactions and producing state transition proofs. In other words, validators maintain the network by aggregating blockchain transactions into block candidates and producing state transition proofs based on those blocks.

The validators also secure the network. If a candidate block is invalid, it will get rejected. On the contrary, too many validators may slow down the network. The only nefarious power validators have transaction censorship. The blockchain must only ensure some neutral validators to prevent censorship - but not necessarily a majority. Theoretically, the censorship problem is solved by having just one honest validator.

The performance of the network depends directly on validators. To ensure optimal network performance, it runs on *Proof-of-Authority* (PoA). This means every validator has to be registered beforehand.

Aura PoA Consensus

The first phase of PoA implementation was via the Aura pallet. Aura PoA Phase - permissioned block authoring and collator session key setup for the Mandala Chain ecosystem.

Let’s break down the latest phase:

- **Validator staking**: validators can now start with securing the network. This will be with a minimum bond of a fixed amount of tokens.
- **Network inflation**: The Mandala Chain Niskala testnet has a 10% inflation. This 10% is based on a perfect block production every 6 seconds.
- **Rewards**: A fixed amount will be created at each block and divided between the

treasury and validators.

A validator is rewarded a fixed amount for each block produced.

Validator election mechanism

Election process

To join the election process, you must register for a validator and bond tokens. Please take a look at the Validator Requirements for details. When your node fits the parameters and checks all the boxes to become a validator, it will be added to the chain. **Note: If your validator doesn’t produce blocks during two sessions (2h), it will be kicked out.**

Validator reward distribution mechanism

At every block you produce as a validator, rewards will automatically be transferred to your account. The reward includes block reward + fees.

Slashing mechanism

A slashing mechanism will be implemented on The Mandala Chain networks - a validator that doesn't produce blocks during two sessions (2 hours) will be slashed 1% of its total stake and kicked out of the active collator set. This slashing ensures the best block rate and prevents malicious actors from harming the network without financial consequences.

How many KPGs do I need to become an active Validator?

You can have a rough estimate of that by using the methods listed here. To be elected into the set, you need a minimum stake behind your validator. This means that, at minimum, you will need enough KPG to set up stash and staking proxy accounts with the existential deposit, plus a little extra for transaction fees. The rest can come from nominators. To understand how validators are elected, refer to the ***PoA Election*** page.

Run an Archive Node

Overview

An **archive node** stores the history of past blocks. Most of the time, an archive node is used as an **RPC endpoint**. RPC plays a vital role in our network: it connects users and dApps to the blockchain through WebSocket and HTTP endpoints. For example, our public endpoints

```
node1.mandalachain.io
wss://node1.mandalachain.io
```

run archive nodes for anyone to connect to the Mandala Chain quickly.

DApp projects must run their own RPC node as an archive to retrieve necessary blockchain data and not rely on public infrastructure. Public endpoints respond slower because of the large amount of users connected and are rate-limited.

> CAUTION
>
> Be careful not to confuse with a **full node** that has a pruned database: a full node only stores the current state and most recent blocks (256 blocks by default) and uses much less storage space.

We maintain two different networks: Madya mainnet and Niskala testnet.

MandalaChain	Token	Ticker
Madya	Kepeng	KPG

MandalaChain	Token	Ticker
Niskala	Kepeng Testnet	KPGT

Requirements

Machine

Component	Requirement
System	Ubuntu 20.04
CPU	8 cores
Memory	16 GB
Hard Disk	500 GB SSD (NVMe preferable)

Component	Requirement
System	Ubuntu 20.04
CPU	4 cores
Memory	8 GB
Hard Disk	200 GB SSD (NVMe preferable)

Ports

The Mandala Chain node will listen at different ports by default for both the mainnet and the testnet.

Description	Testnet Port	Custom Port Flag
P2P	30333	`--port`
RPC	9944	`--rpc-port`
Prometheus	9615	`--prometheus-port`

For all types of nodes, ports `30333` and `30334` need to be opened for incoming traffic at the Firewall. **Validator nodes should not expose WS and RPC ports to the public.**

Installation

There are 2 different ways to run a Mandala Chain node:

Using Binary - run the node from a binary file and set it up as `systemd` service

Using Docker - run the node within a Docker container

Validator Requirement

How to become a validator

Permissioned validator

To become a permissionless collator on Mandala Chain networks, you need to meet the requirements below.

Collator staking requirements

- Bond: 3,200,000 KPG tokens (3.2M $KPG)
- Meet hardware requirements
- Bond: 3,200,000 KPGT tokens (3.2M $KPGT)
- Meet hardware requirements

If your node stops producing blocks for one session, your node will be kicked out of the active set, and 1% of the bonded funds will be slashed. Running a node with low performance can lead to skipping blocks, which can lead to being kicked out of the active set.

> TIP
>
> Set your collator with: **Extrinsics - CollatorSelection - Register as candidate** | Onboarding takes **n+1** session.

System requirements

A collator deploys its node on a remote server. You can choose your preferred provider for dedicated servers and operating systems. Generally speaking, we recommend you select a provider/server in your region. This will increase the decentralization of the network. You can choose your preferred operating system, though we highly recommend Linux.

Hardware requirements

Use the charts below to find the basic configuration, guaranteeing that all blocks can process in time. If the hardware doesn't meet these requirements, there is a high chance it will malfunction, and you risk being automatically **kicked out and slashed** from the active set.

> CAUTION
>
> Ensure your server is a **bare metal only dedicated to the collator node**; any unnecessary process running on it will significantly decrease the collator performance. **We strongly discourage using a VPS** to run a collator because of their low performance.
>
> Collators are the nodes which require the most powerful hardware because they have short time frames to assemble a block and collate it to the relay chain. To run a collator, it is necessary to use a **CPU of a minimum 4 Ghz per core** and an **NVMe SSD disk** (SATA SSDs are unsuitable for collators because they are too slow).

Component	Requirement
System	Ubuntu 20.04
CPU	8 cores
Memory	16 GB
Hard Disk	500 GB SSD (NVMe preferable)

Component	Requirement
System	Ubuntu 20.04
CPU	4 cores
Memory	8 GB
Hard Disk	200 GB SSD (NVMe preferable)

Ports

Niskala Testnet

Overview

The Niskala Testnet of Mandala Chain is a blockchain network designed to facilitate the development, testing, and deployment of decentralized applications (dApps) in a secure and scalable environment. As a testnet, it provides developers with a platform to experiment with new features and functionalities without the risks associated with mainnet deployments.

Key Features

✅ Scalability

Niskala is built to handle a high throughput of transactions, making it suitable for applications that require fast and efficient processing. Its architecture allows for horizontal scaling, enabling the network to accommodate increasing user demands.

✅ Interoperability

Niskala supports interoperability with other blockchain networks, allowing developers to create cross-chain applications. This feature enhances the flexibility and usability of dApps, enabling them to leverage resources and functionalities from multiple blockchains.

✅ Security

Security is a top priority for the Niskala. The network employs advanced cryptographic techniques and consensus mechanisms to ensure the integrity and confidentiality of transactions. Regular audits and community reviews further enhance the security posture of the testnet.

☑ Community-Driven Governance

Niskala operates under a community-driven governance model, allowing stakeholders to participate in decision-making processes. This approach ensures that the network evolves in a way that reflects the needs and desires of its users.

☑ Developer-Friendly Environment

Niskala provides a robust set of tools and resources for developers, including comprehensive documentation, and APIs. This support enables developers to quickly build, test, and deploy their applications with minimal friction.

Use Cases

- **Non-Fungible Tokens (NFTs)**: The testnet supports the creation and trading of NFTs, enabling artists and creators to experiment with digital ownership and collectibles.
- **Identity Management**: The Niskala can be use to build and test decentralized identity solutions, allowing user to control their personal data securely.
- **Decentralized Finance (DeFi)**: Developers can create and test DeFi applications, including lending platforms, decentralized exchanges, and yield farming protocols.
- **Medical Data Management:** Enhancing healthcare through secure, interoperable medical data management, ensuring patient privacy and seamless access for providers.
- **Insurance & Accreditation:** Working with multiple providers to enable insurance and accreditation processes with immutable records and automated verification, reducing fraud and improving trust.
- **Supply Chain Management:** Create and test infrastructure to optimize logistics and supply chains with transparent, tamper-proof tracking systems, ensuring efficiency and authenticity.

Getting Started

To begin using the Niskala Testnet of Mandala Chain, follow these steps:

1. **Set Up a Wallet**: Create a wallet compatible with the Niskala to manage your assets and interact with dApps.
2. **Access the Testnet**: Connect to the Niskala Testnet using the provided RPC endpoints and network configurations.
3. **Explore dApps**: Visit the testnet's ecosystem to discover and interact with various decentralized applications.

The Niskala Testnet of Mandala Chain is a powerful platform for developers and users looking to explore the potential of blockchain technology. With its focus on scalability, security, and community engagement, the Mandala Chain provides an ideal environment for innovation and experimentation in the decentralized space.

Build Environment

Niskala provides developers with the necessary tools and resources to create, test, and deploy decentralized applications (dApps) on the Niskala blockchain. This documentation outlines key components of the build environment, including network RPC endpoints, test tokens (faucets), and instructions for running a local network.

Network RPC Endpoint			network-rpc-endpoints.md
Test Token (Faucets)			test-tokens-faucets.md
Running Local Network			running-a-local-network.md

Running Local Network

This section guides users through setting up and running a local network, with clear steps and additional context to ensure a smooth setup experience.

Setting Up a Local Network with Niskala

This guide provides detailed instructions for setting up and running a local network using Niskala, built with Substrate. This setup is ideal for development, testing, and experimentation with Niskala.

Prerequisites

Before you begin, ensure you have the following:

- A compatible operating system (Linux, macOS, or Windows).
- Rust installed on your system.
- Familiarity with command-line operations.
- Zombienet installed for local testnet setup.
- Get the Latest Binary, visit the Release page of the Mandala Github repository.

Installation

Step 1: Install Dependencies

Ensure your system has the necessary dependencies. Follow the Substrate Installation Guide for your specific platform to install required packages.

Step 2: Clone the Repository

Clone the Mandala Node repository to your local machine:

```
git clone https://github.com/your-repo/mandala-node.git
cd mandala-node
```

Step 3: Build the Node

Compile the node with the desired runtime:

```
cargo build --release --features mandala-native
```

> **Note:** Use `niskala-native` if you prefer the Niskala runtime.

Running a Local Network

Step 1: Generate Genesis State and Runtime

Generate the genesis state and runtime for your node:

```
./target/release/mandala export-genesis-state --dev > <path>
```

```
./target/release/mandala export-genesis-wasm --dev > <path>
```

> Replace the path with folder you wish to store the state and runtime

Step 2: Set Up the Relay Chain

Navigate to your Zombienet directory and start the relay chain:

```
cd zombienet
./run.sh <zombienet-path>
```

> **Info:** This command initializes a 2-node relay chain with Bob and Alice as validators.

Step 3: Launch the Collator

Return to the Mandala Node directory and start the collator:

```
./target/release/mandala --dev --charlie --collator --rpc-port
9944 --port 30333 -- --chain ./zombienet/plain.json --discover-
local --port 30334
```

Step 4: Register the Parachain

After starting the Zombienet script, register your parachain:

1. Open a browser and navigate to the Polkadot/Substrate Portal.
2. In the developer tab, go to "sudo" and select `parasSudoWrapper`.
3. Choose `sudoScheduleParaInitialize(id, genesis)`.
4. Set the `id` parameter to `2000`.
5. Upload the `genesis-state` file for the `genesisHead` parameter.
6. Set `paraKind` to `true` and submit the transaction.

Wait for the next epoch to start, and your parachain should begin producing blocks.

Interacting with the Local Network

You can interact with your local network using the Polkadot/Substrate Portal. This interface allows you to explore blocks, transactions, and other network activities.

Additional Resources

- Substrate Documentation
- Polkadot-JS Apps

Network RPC endpoints

What is RPC Node?

Remote Procedure Call (RPC) endpoints are crucial components in blockchain networks, allowing users and applications to interact with the blockchain. This document provides an overview of RPC nodes in the Niskala network, detailing public endpoints and their intended use cases.

An RPC node is a server that allows clients to communicate with a blockchain network using the Remote Procedure Call protocol. It acts as an intermediary between the client (such as a decentralized application or dApp) and the blockchain, enabling users to send transactions, query data, and interact with smart contracts.

In the context of the Niskala network, RPC nodes facilitate seamless interaction with the blockchain, providing developers and users with the necessary tools to build and utilize decentralized applications.

Public Endpoints

Niskala offers several public RPC endpoints that are accessible to users and developers. These endpoints are designed to facilitate interaction with the blockchain for various purposes, including deploying and calling smart contracts, as well as querying blockchain data.

> INFO
>
> The free endpoints below are dedicated to end users, they can be used to interact with dApps or deploy/call smart contracts.

> They limit the rate of API calls, so they are not suitable for high demand, such as a dApp UI constantly scraping blockchain data or an indexer.

	Public endpoint Niskala
Network	Niskala
HTTPS	https://mlg1.mandalachain.io/
WebSocket	wss://mlg1.mandalachain.io/
chainID	6025
Symbol	KPGT

The RPC endpoint network for Niskala plays a vital role in enabling interaction with the blockchain. By understanding the capabilities and limitations of the public endpoints, developers can effectively build and deploy decentralized applications while ensuring optimal performance and user experience.

For more information and updates regarding the Niskala RPC endpoints, please reach out to us using our official support email: [email protected]

Test Tokens (Faucets)

A faucet is a dedicated platform where users can obtain test tokens to facilitate development and testing on the Niskala network. In this documentation, we will explore effective methods to acquire KPGT tokens.

How to Get KPGT

To receive testnet tokens, please follow these steps:

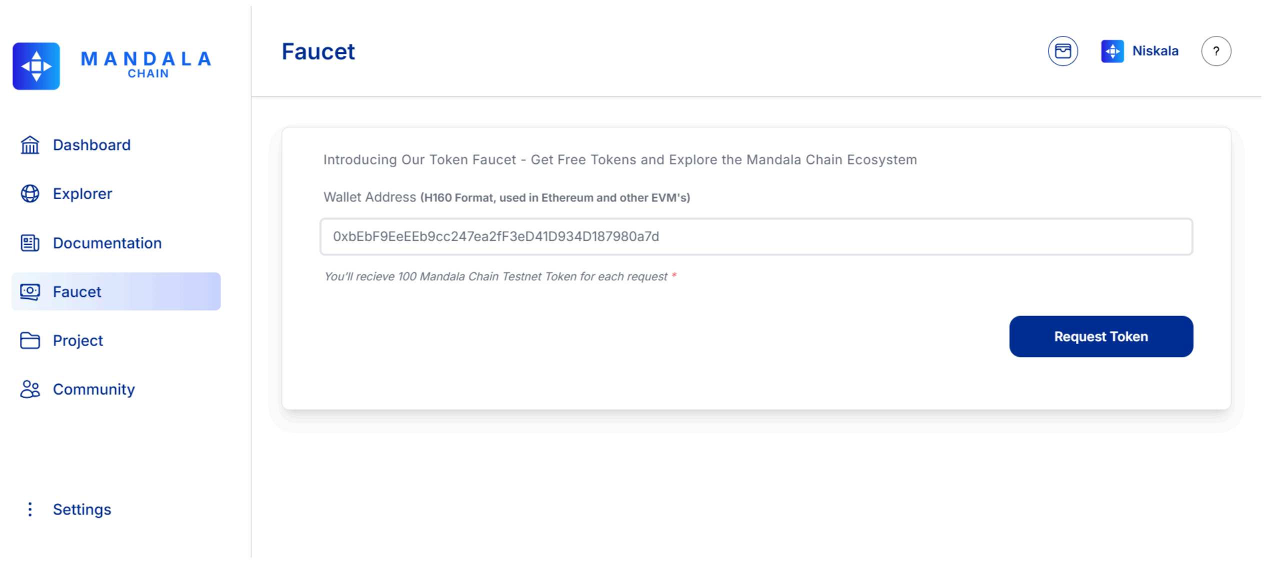
1. Visit the Mandala Chain Website

- Open your web browser
- Enter the URL: hub.mandalachain.io

2. Navigate to the Faucet Page

- Once on the site, look for the **Faucet** menu in the left sidebar
- Click on it to go to the faucet page

3. Enter Your Wallet Address



Faucet

Introducing Our Token Faucet - Get Free Tokens and Explore the Mandala Chain Ecosystem

Wallet Address **(H160 Format, used in Ethereum and other EVM's)**

0xbEbF9EeEEb9cc247ea2fF3eD41D934D187980a7d

*You'll recieve 100 Mandala Chain Testnet Token for each request **

- On the faucet page, you will find a field labeled **Wallet Address**
- Make sure your wallet address is in H160 format, which is used in Ethereum and other EVMs
- Enter your wallet address into the provided field

4. Request Tokens

- After entering your wallet address, click the **Request Token** button
- A pop-up will appear confirming your request:
    - Pop-up message: "Your request has been successful. You can check the result through the link below."
    - You will see options to **Check** or **Close**

5. Confirm Receipt

- Check your wallet to ensure that the tokens have been received

- You will receive 100 Niskala Testnet Tokens for each request (*this process may take a few minutes*)

Additional Tips

- Make sure your wallet is connected to **Niskala** to receive the tokens
- If you do not receive tokens after some time, try again or check your wallet settings

Important Security Notice: Our team will **NEVER** DM you first, and we **NEVER** ask for your **money** in exchange for faucets!

If someone approaches you pretending to be part of the team, do NOT engage with them. Instead, please use the official Discord faucet channel mentioned above.

Use

Overview on account format and how to interact with Mandala Chain via wallet.

📄 Account

Mandala Chain Account

📄 Wallets

Step 1

Account

Mandala Chain Account

Address Format

The address format used in Substrate-based chains like Mandala Chain is SS58. SS58 is a modification of Base-58-check from Bitcoin with some minor modifications. Notably, the format contains an address type prefix that identifies an address as belonging to a specific network. Mandala Chain supports EVM as well as WASM smart contract. With the use of two different virtual machines come two different kinds of addresses.

- A Mandala Chain Native address or SS58 address
- A Mandala Chain EVM address or H160 address which starts with 0x

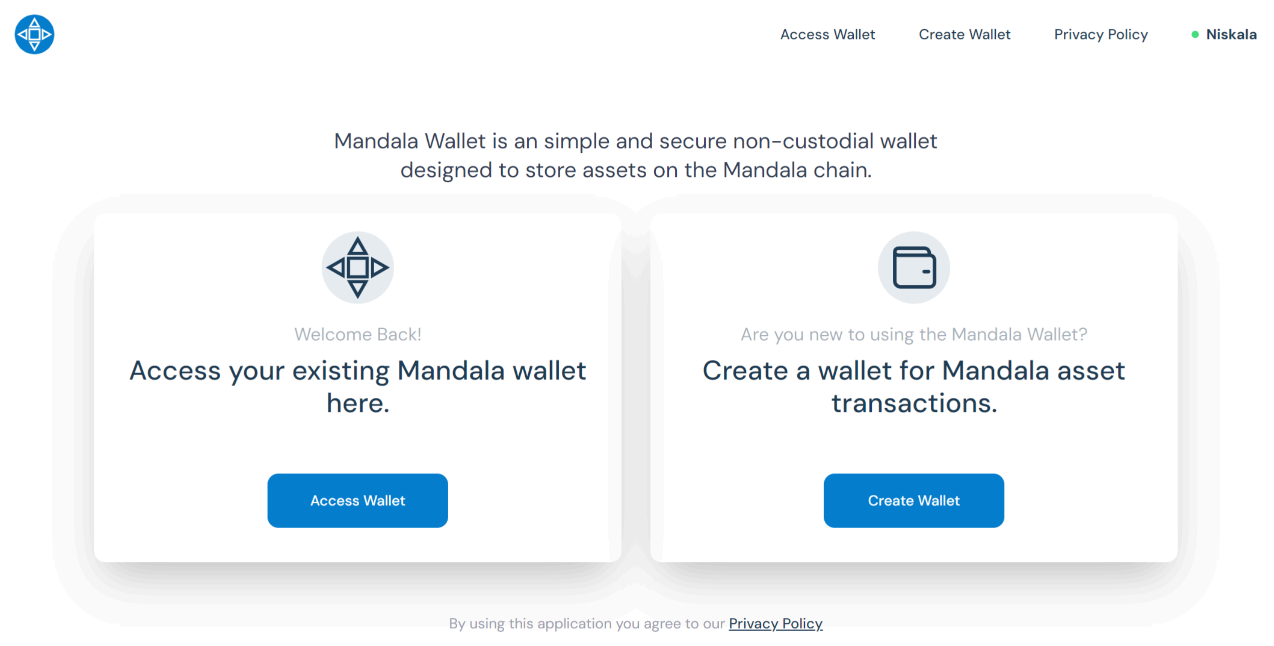
You will interact with our Mandala Chain native address when using WASM dApps. Using this address requires extensions. We recommend using the Mandala Wallet or Polkadot JS extension. We will support EVM-based wallets in the future.

Wallets

Mandala Wallet

Step 1

Head over to our Mandala Wallet website and click on the **Create Wallet** button.



Mandala Wallet is an simple and secure non-custodial wallet designed to store assets on the Mandala chain.

Welcome Back!

Access your existing Mandala wallet here.

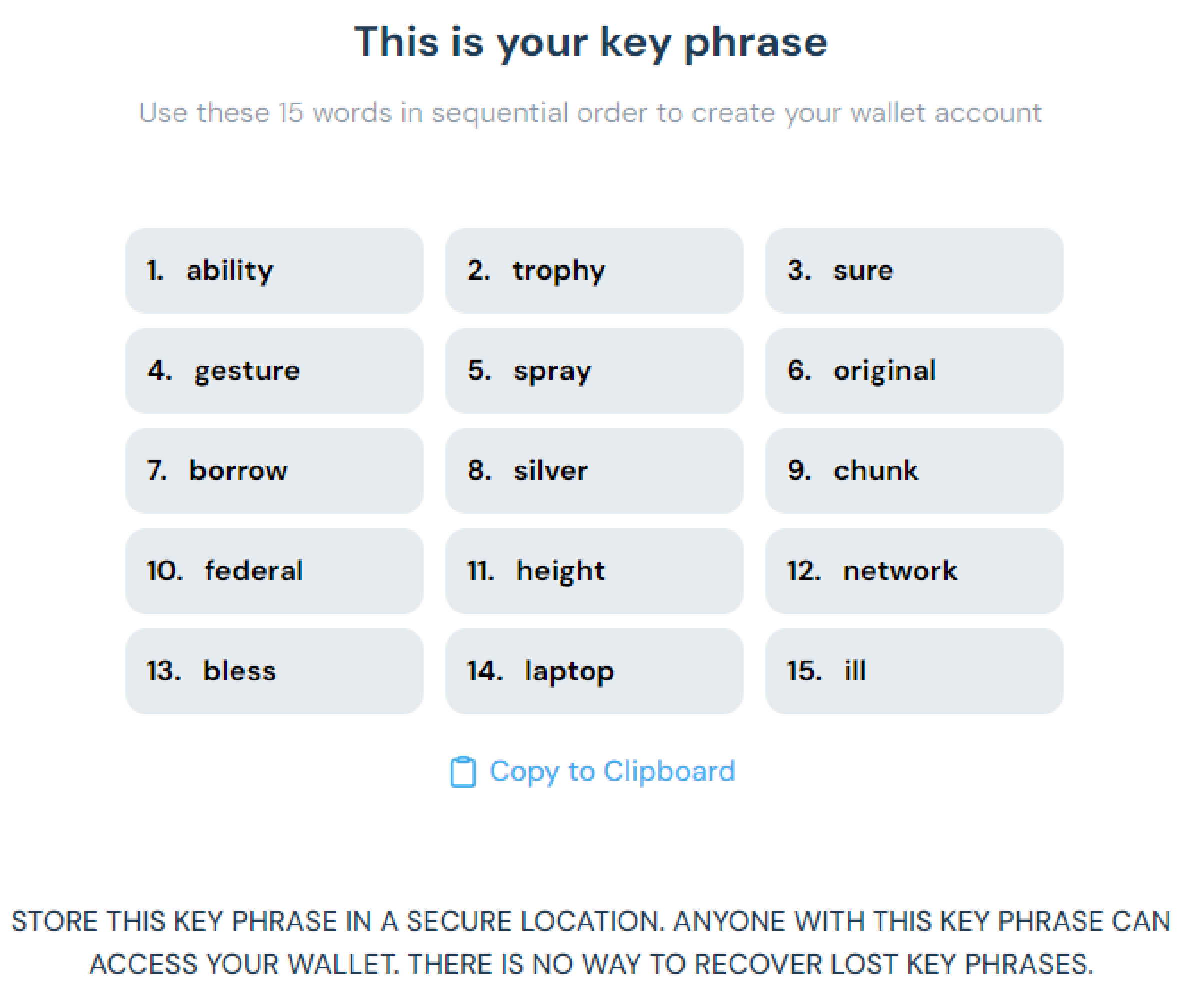
Are you new to using the Mandala Wallet?

Create a wallet for Mandala asset transactions.

By using this application you agree to our Privacy Policy

Step 2

Get your **Key Phrase** stored in a ***secure*** location. You may copy them into Clipboard and paste them somewhere else temporarily, or (preferred) write them down on paper and keep them accessible by yourself only.



This is your key phrase

Use these 15 words in sequential order to create your wallet account

1. ability	2. trophy	3. sure
4. gesture	5. spray	6. original
7. borrow	8. silver	9. chunk
10. federal	11. height	12. network
13. bless	14. laptop	15. ill

Copy to Clipboard

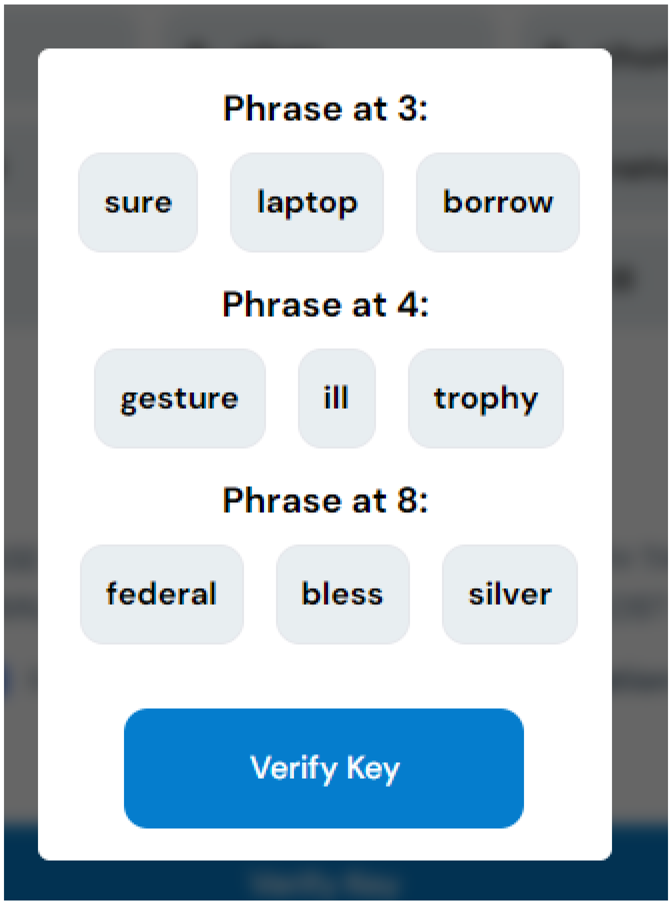
STORE THIS KEY PHRASE IN A SECURE LOCATION. ANYONE WITH THIS KEY PHRASE CAN ACCESS YOUR WALLET. THERE IS NO WAY TO RECOVER LOST KEY PHRASES.

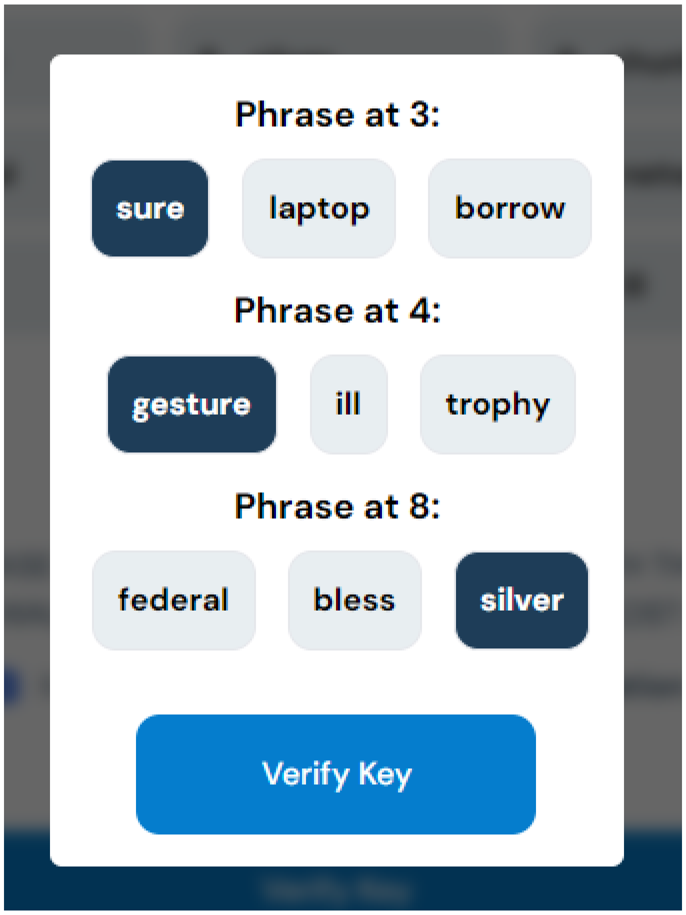
⚠ **WARNING**

Losing your **Key Phrase** means you will lose all your assets or if someone can access your **Key Phrase** inadvertently, that also means they will gain control of your assets. Be sure to store them in a secure location and if you really want to share them, please be sure you do it with full awareness.

Step 3

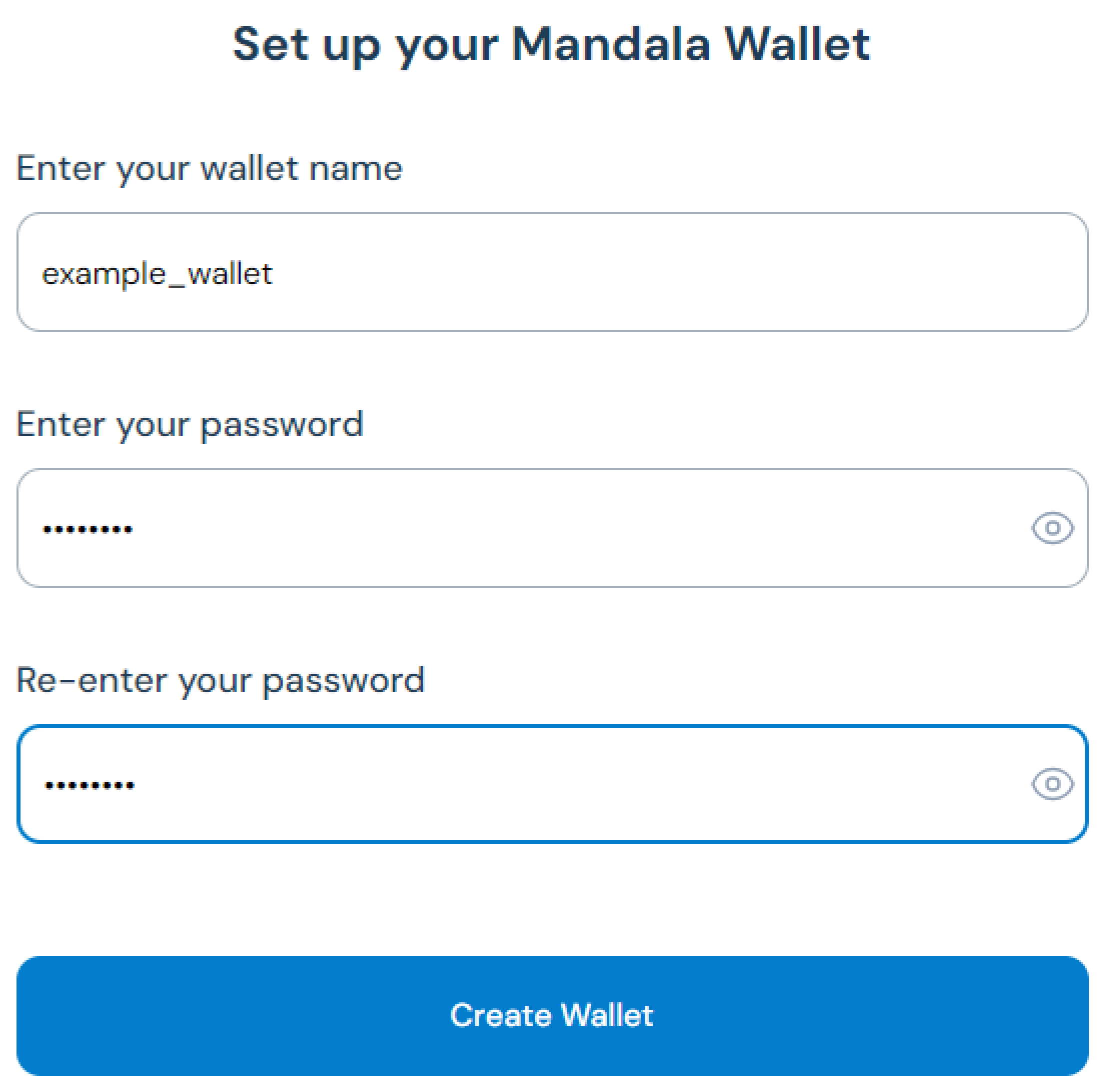
Confirm that you've already stored your Key Phrase by confirming any key required. Verify those keys by selecting all the correct key phrases.


Phrase at 3:
sure
laptop
borrow
Phrase at 4:
gesture
ill
trophy
Phrase at 8:
federal
bless
silver
Verify Key



Step 6

Give your wallet a **name** and **password**.



Set up your Mandala Wallet

Enter your wallet name

example_wallet

Enter your password

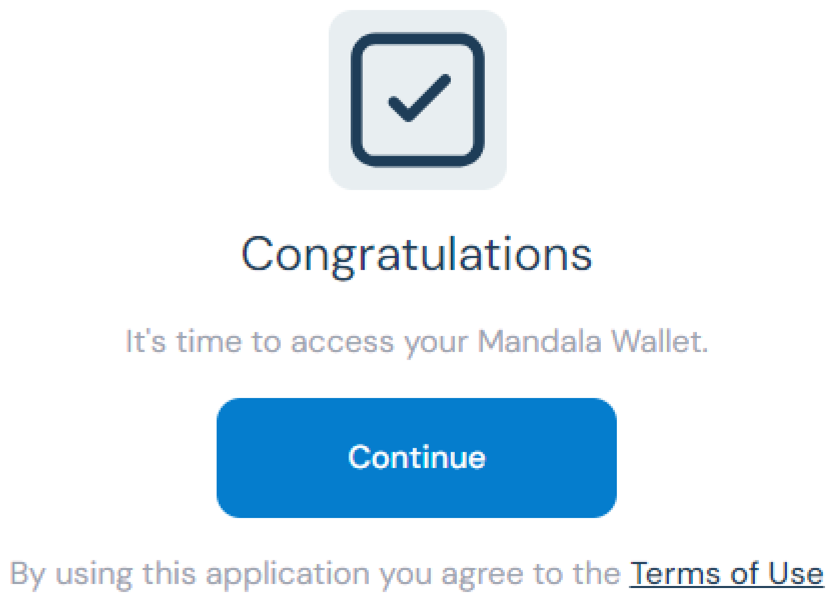
•••••••••

Re-enter your password

•••••••••

Step 7

Your wallet should be set up, and you can click on the Continue button to start using it.



Congratulations

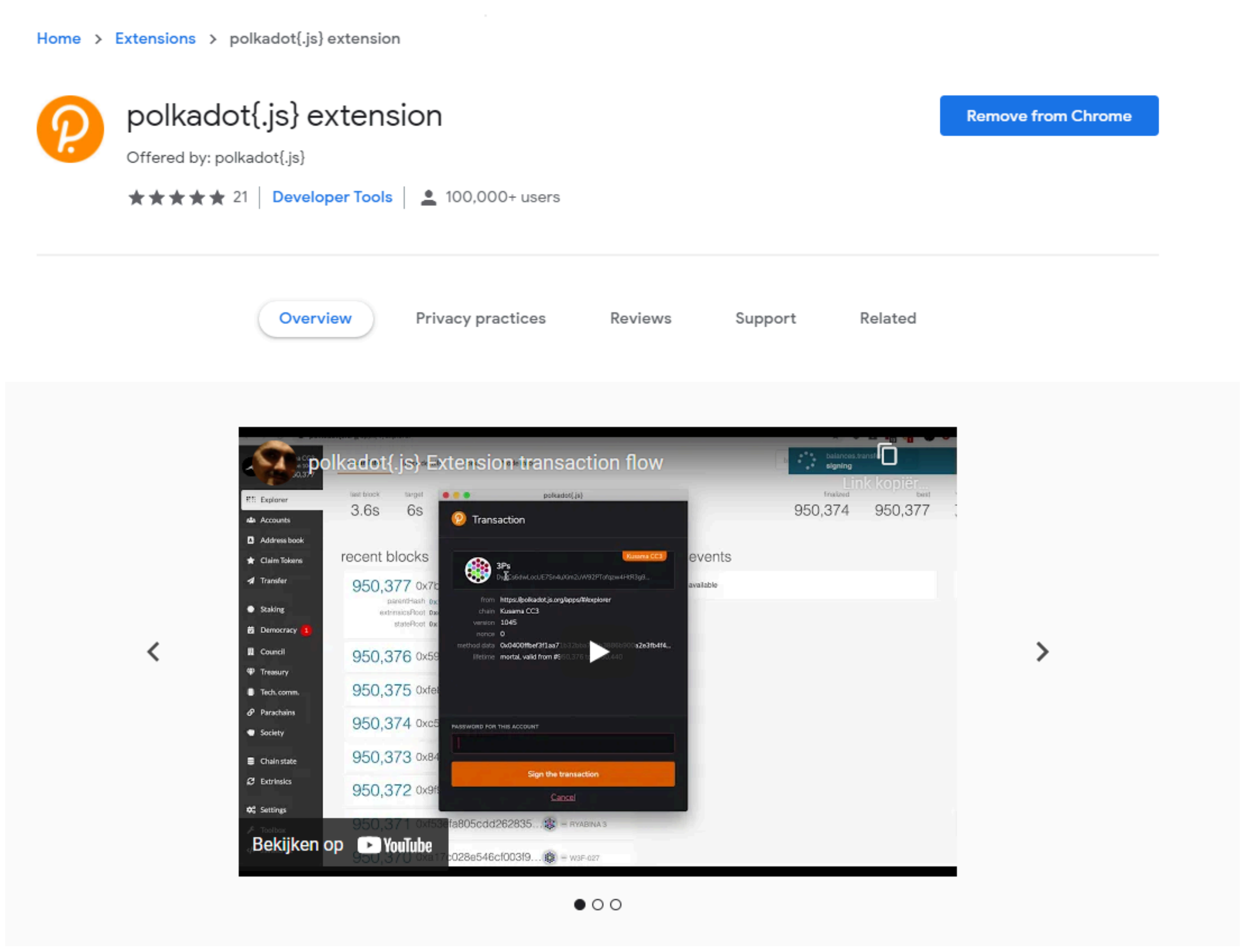
It's time to access your Mandala Wallet.

By using this application you agree to the Terms of Use

Polkadot JS App

Polkadot.js Browser Plugin

Polkadot.js Browser Plugin The browser plugin is available for both Google Chrome (and Chromium-based browsers like Brave) and Firefox. After installing the plugin, you should see the orange and white Polkadot.js logo in your browser menu bar.


Home > Extensions > polkadot{.js} extension

polkadot{.js} extension

Offered by: polkadot{.js}

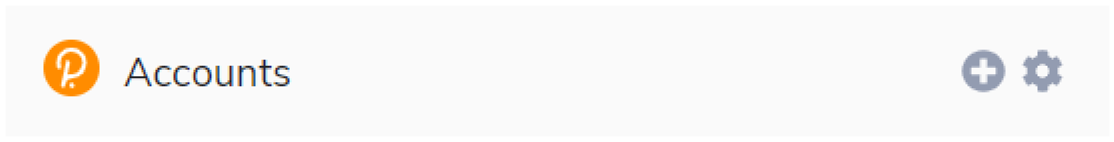
★★★★★ 21 | Developer Tools | 100,000+ users

Remove from Chrome

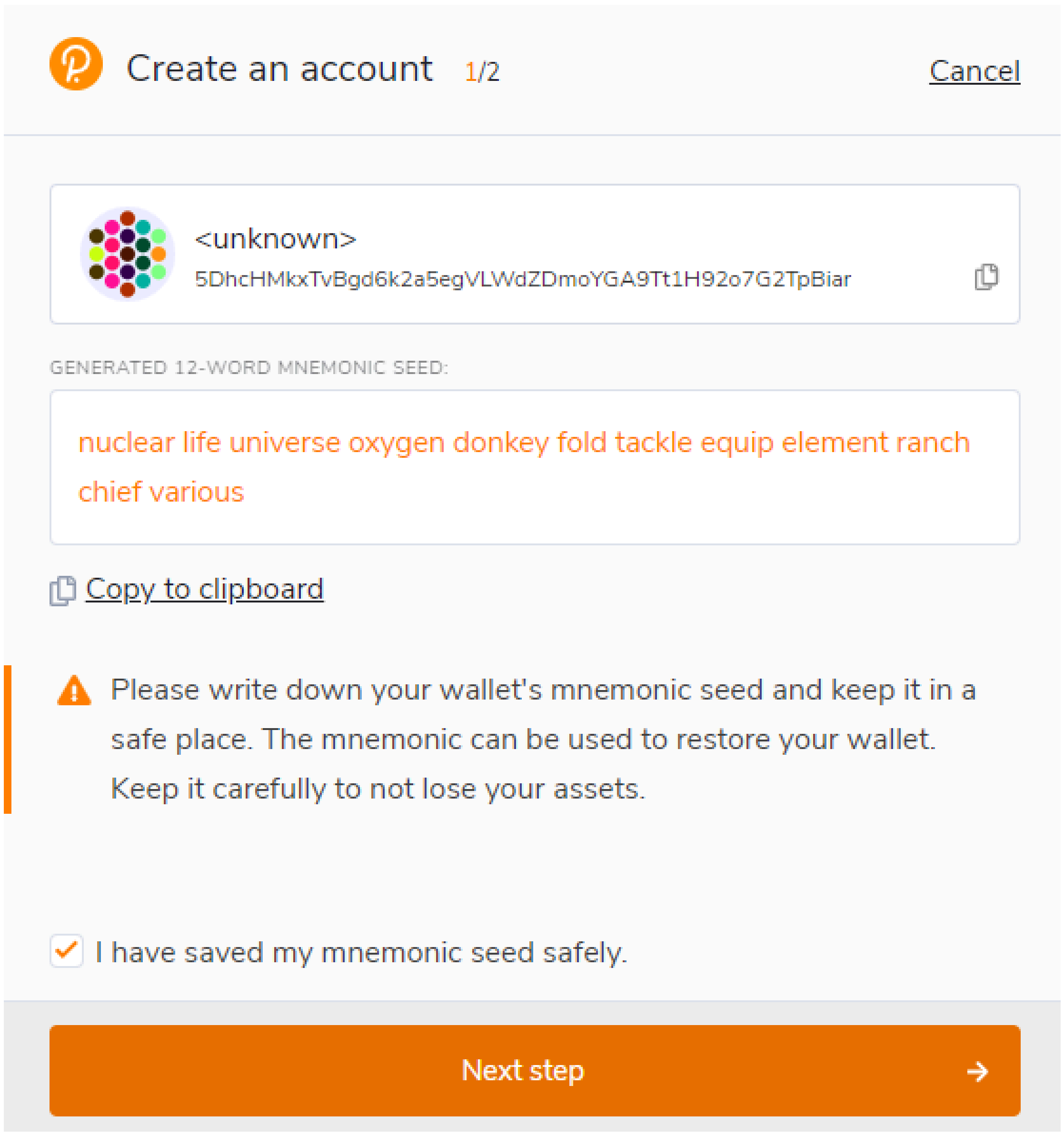
Overview Privacy practices Reviews Support Related

Create Account

Open the Polkadot.js browser extension by clicking the logo on the top bar of your browser.



Click the big plus button - "Create new account." The Polkadot.js plugin will then use system randomness to make a new seed for you and display it to you in the form of twelve words.


Create an account 1/2

Cancel

<unknown>

5DhcHMkxTvBgd6k2a5egVLWdZDmoYGA9Tt1H92o7G2TpBiar

GENERATED 12-WORD MNEMONIC SEED:

nuclear life universe oxygen donkey fold tackle equip element ranch chief various

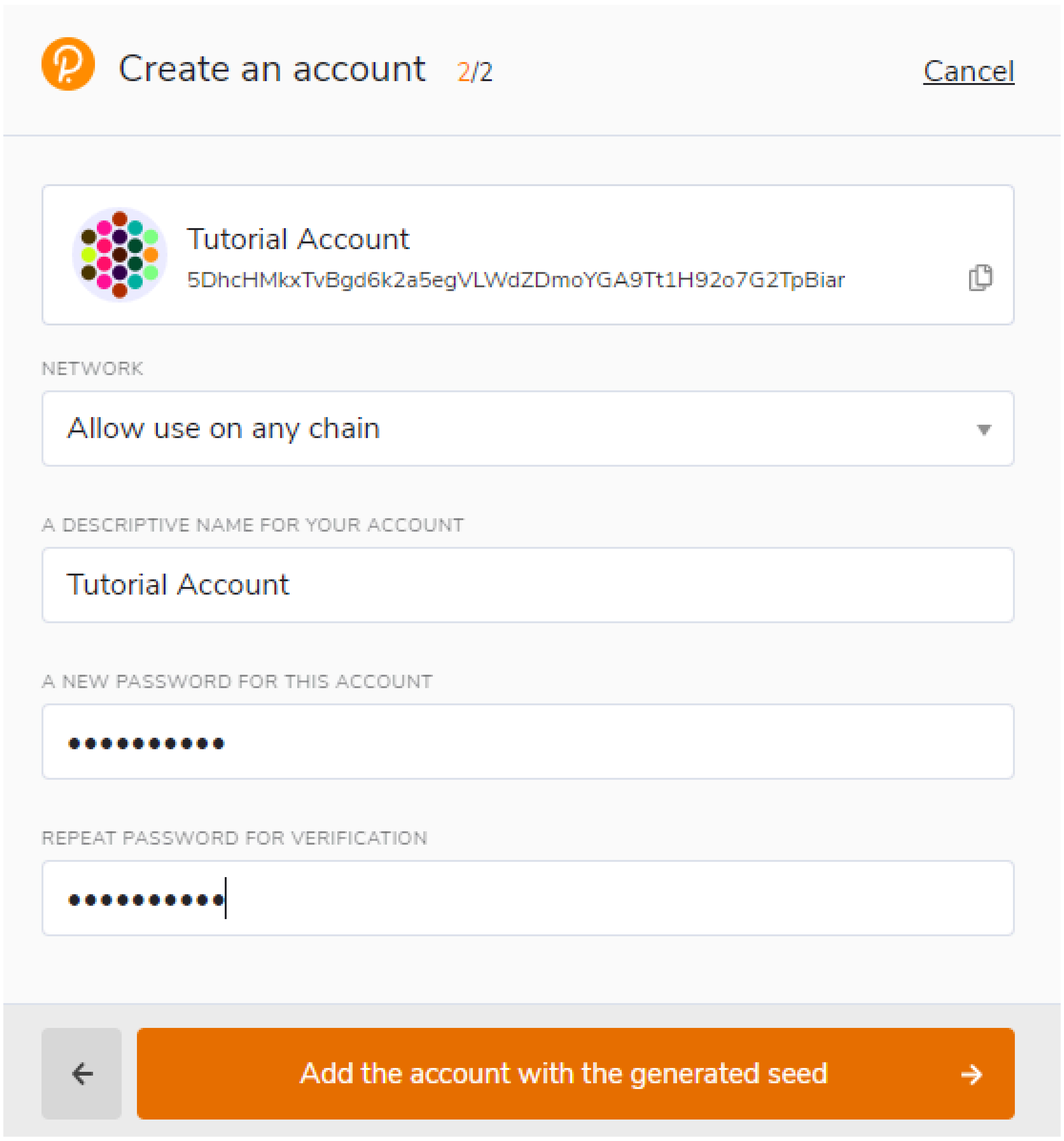
Copy to clipboard

Please write down your wallet's mnemonic seed and keep it in a safe place. The mnemonic can be used to restore your wallet. Keep it carefully to not lose your assets.

✓ I have saved my mnemonic seed safely.

Next step

You need to back up these words securely!



Tutorial Account
5DhcHMkxTvBgd6k2a5egVLWdZDmoYGA9Tt1H92o7G2TpBiar

NETWORK
Allow use on any chain

A DESCRIPTIVE NAME FOR YOUR ACCOUNT
Tutorial Account

A NEW PASSWORD FOR THIS ACCOUNT
••••••••••

REPEAT PASSWORD FOR VERIFICATION
••••••••••

DANGER

Note that this password does **NOT** protect your ***seed phrase***. If someone knows the twelve words in your mnemonic seed, they still have control over your account even if they do not know the password.