ignite chain

Whitepaper

Ignite Chain

Last Revised 2022.12.15

Authored and collectively contributed to by:

Lucas Bertrand
Head of Ignite Chain

@lubtd

Alex Johnson
Head of Ignite CLI

@aljo242

Scott Carter
Head of Design

@scottcarterco

Aidar Itkulov
Sr. Product Designer

@aidarItkulov

Table of Contents

Introduction

Ignite is a blockchain to help launch Cosmos SDK-based blockchains.

Using Cosmos SDK and Ignite CLI, developers can quickly create a crypto application that is decentralized, economical for usage, and scalable. The Cosmos SDK framework allows developers to create sovereign, application-specific blockchains that become part of the wider Cosmos ecosystem. Blockchains created with Cosmos SDK use a Proof-of-Stake (PoS) consensus protocol that requires validators to secure the chain.

Even though tools like Ignite CLI simplify the development of a Cosmos SDK blockchain, launching a new chain remains a highly complex process. One of the major challenges of developing and launching your own sovereign blockchain is ensuring the security of the underlying consensus. Since Cosmos SDK chains are based on the PoS consensus, each blockchain requires both initial coin allocations and validators before they can be launched. This presents developers with significant challenges, such as determining their chain's tokenomics and coordinating a robust validator set.

The initial coin allocations and validators are described in a JSON-formatted genesis file that is shared among all initial nodes in the network. This genesis file defines the initial state of the application. Based on PoS, secure chains require the initial allocation of coins to be well distributed so that no single validator holds more than 1/3 of all tokens or receives a disproportionate amount of voting power on the network.

Along with ensuring the security of the underlying consensus, another highly difficult task in launching a new blockchain is attracting a diverse set of validators for the genesis file. Many promising projects fail to attract a sufficient number of trustworthy validators to secure their chains due to a lack of resources or experience. Additionally, many projects struggle with their project’s fundraising efforts leading up to the launch of their chain.

The Ignite Chain has, therefore, been conceived to facilitate the launch of Cosmos SDK blockchains and fundraising efforts through a series of features that helps developers to navigate the complexities of launching a blockchain, coordinating the genesis of a new chain, and facilitating the fundraising efforts of a project. Ignite Chain's coordination features help blockchain builders connect with and assign validator roles for genesis thus speeding up the time to market of

their projects and furthering their chances of success. Ignite Chain's token issuance features help projects tokenize their future mainnet tokens to aid with their pre-launch fundraising efforts.

Commands to interact with Ignite Chain are integrated into Ignite CLI and allow launching chains from it. Integration with Ignite Chain allows the CLI to support the developer throughout the entire lifecycle of realizing a Cosmos project, from the development and experimentation of the blockchain to fundraising and launch of its mainnet.

What is Ignite Chain?

Ignite Chain is a secure blockchain-based platform that simplifies the launch of Cosmos SDK-based chains, lending vital resources and support during the coordination, preparation, and launch phases. Ignite Chain provides the tools that blockchain projects need to overcome the complexities of launching their chain, from validator coordination and token issuance to fundraising.

Ignite Chain facilitates the launch of new chains with an overall launch process during three main phases:

1. Coordination
2. Preparation
3. Launch

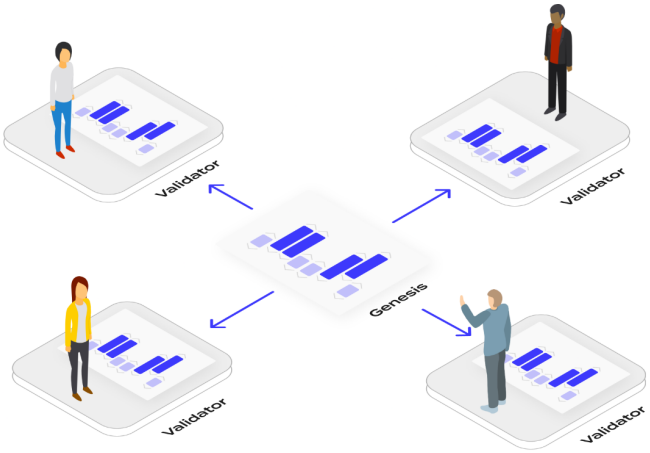
To reduce friction caused by the evolution of a chain during each phase, Ignite Chain provides an immutable and universal database to ensure ease of coordination with validators.

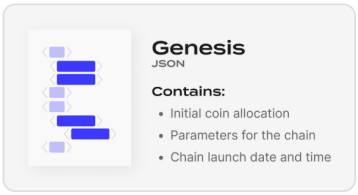
To support the entire project lifecycle from testnet to mainnet, Ignite Chain also offers:

1. The issuance of tokens (called vouchers) that represent a share allocation of a future mainnet blockchain.
2. A fundraising component for selling vouchers.
3. A permissionless framework to reward validator activities on launched testnet networks through the allocation of vouchers.

Validator coordination

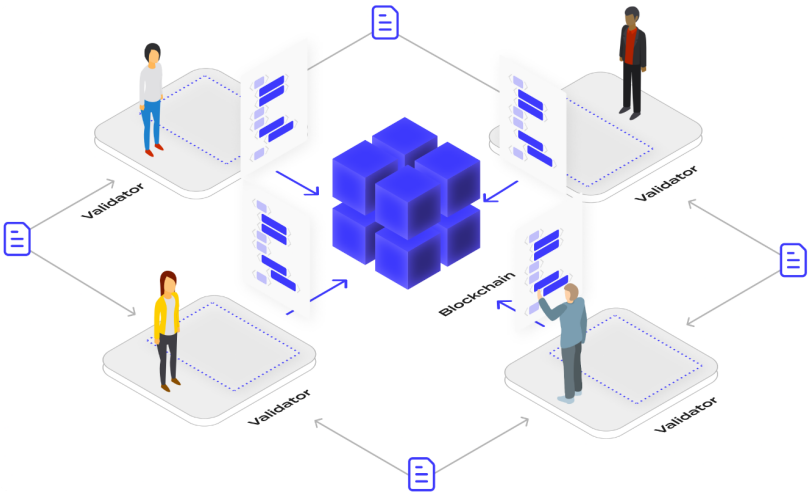
To launch a chain in the Cosmos ecosystem, the validators of a chain must start nodes that connect to each other to create the new blockchain network. A validator node must be started from a file called the genesis file. The genesis file must be identical on all validator nodes before the new chain can be started.

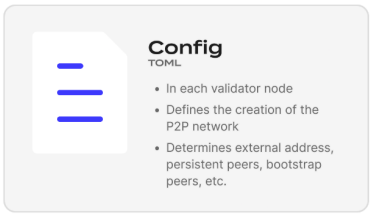




All Validators must possess the same Genesis file to launch.

The JSON-formatted genesis file contains information on the initial state of the chain, including coin allocations, the list of initial validators, various parameters for the chain like the maximum number of validators actively signing blocks, and the specific launch time for the blockchain. Because each validator has the exact same genesis file, the blockchain network starts automatically when validator nodes with token delegations collectively totaling at least ⅔ of all tokens for the network are online and once the genesis time is reached.





When the genesis time is reached, the blockchain network starts automatically.

Ignite Chain as a coordination source of truth

Ignite Chain acts as a source of truth for new chains to coordinate a validator set and for validators to generate the genesis for a chain launch. Ignite Chain doesn't directly store the final genesis file in its own ledger but rather stores information that enables genesis file generation in a deterministic manner.

The information stored on Ignite Chain that supports deterministic generation of the genesis file for a specific chain launch is referred to as the *launch information*. When creating a new chain on Ignite Chain, the coordinator provides the initial launch information. Then, through on-chain coordination via Ignite Chain, this launch information is updated by interacting with Ignite Chain by sending messages. When the new chain is ready to be launched, the genesis file is generated by calling a genesis generation algorithm that uses the launch information stored on Ignite Chain.

`GenesisGenerate(LaunchInformation) ⇒ genesis.json`

The genesis generation algorithm is officially and formally specified. The official implementation of the genesis generation algorithm is developed in the Go programming language using Ignite CLI. However, any project is free to develop its own implementation of the algorithm as long as it complies with the specifications of the algorithm.

The genesis generation algorithm is not part of the on-chain protocol. In order to successfully launch a new chain, all validators must use the same algorithm to generate their genesis file using the launch information. The algorithm deterministically generates the genesis file from the launch information that is stored on the Ignite Chain.

If any element of the launch information is altered, for example, removing an account balance, then the launched chain reputation is negatively impacted and implies that the majority of validators did not use:

- The tamper-proof launch information
- The official genesis generation algorithm

Outside of genesis generation, the genesis generation algorithm specification gives guidance on how to set up the network configuration. For example, the launch information can contain the addresses of the persistent peers of the blockchain network.

Ignite Chain stores launch information for new chains

Each validator runs the same genesis generation from the information stored on Ignite Chain for the new chain

New chain successfully launched

Launch information

Launch information can be created or updated in three different ways:

1. Defined during chain creation (but updatable by the coordinator after creation)
2. Determined through coordination
3. Determined through specific on-chain logic not related to coordination

1 - Launch information defined during chain creation:

- ***GenesisChainID:*** The identifier for the network.
- ***SourceURL:*** The URL of the git repository of the source code for building the blockchain node binary.
- ***SourceHash:*** The specific hash that identifies the release of the source code.
- ***InitialGenesis:*** A multiformat structure that specifies the initial genesis for the chain launch before running the genesis generation algorithm.

2 - Launch information determined through coordination:

- ***GenesisAccounts:*** A list of genesis accounts for the chain, consisting of addresses with associated balances.
- ***VestingAccounts:*** A list of genesis accounts with vesting options.
- ***GenesisValidators:*** A list of the initial validators at chain launch.
- ***ParamChanges:*** A list of module parameter changes in the genesis state.

3 - Launch information determined through on-chain logic:

- ***GenesisTime:*** The timestamp for the network start time, also referred to as LaunchTime.

Initial genesis

The launch information contains the initial genesis structure. This structure provides the information for generating the initial genesis before running the genesis generation algorithm and finalizing the genesis file.

The initial genesis structure can be one of the following:

- ***DefaultGenesis:*** The default genesis file is generated by the chain binary init command.
- ***GenesisURL:*** The initial genesis for a chain launch is an existing genesis file that is fetched from a URL and then modified with the required algorithm.
    - This initial genesis type should be used when the initial genesis state is extensive, containing a lot of accounts for token distribution, and/or containing records for an airdrop.
- ***GenesisConfig:*** The initial genesis for a chain launch is generated from an Ignite CLI config that contains genesis accounts and module parameters.
    - This initial genesis type should be used when the coordinator doesn't have an extensive state for the initial genesis but some module parameters must be customized. For example, the staking bond denom for the staking token.

Coordination process

The coordination process starts immediately after the chain is created and ends when the coordinator triggers the launch of the chain.

Throughout the coordination process, any entity can send requests to the network. When an entity submits a request, they become known as the request creator, or requestor. A request is an object whose content specifies updates to the launch information. The launch information is updated throughout the coordination process and is directly affected by requests.

The chain coordinator may approve or reject any or all requests at any time. In doing so, the following outcomes occur:

- If a request is approved, the content is applied to the launch information.
- If the request is rejected, no change is made to the launch information.

The request creator can also directly reject or cancel the request. Approving or rejecting a request is irreversible, however, requests may be reverted via the submission of an opposing request for an opposite outcome or an original value (example: *AddAccount → RemoveAccount*).

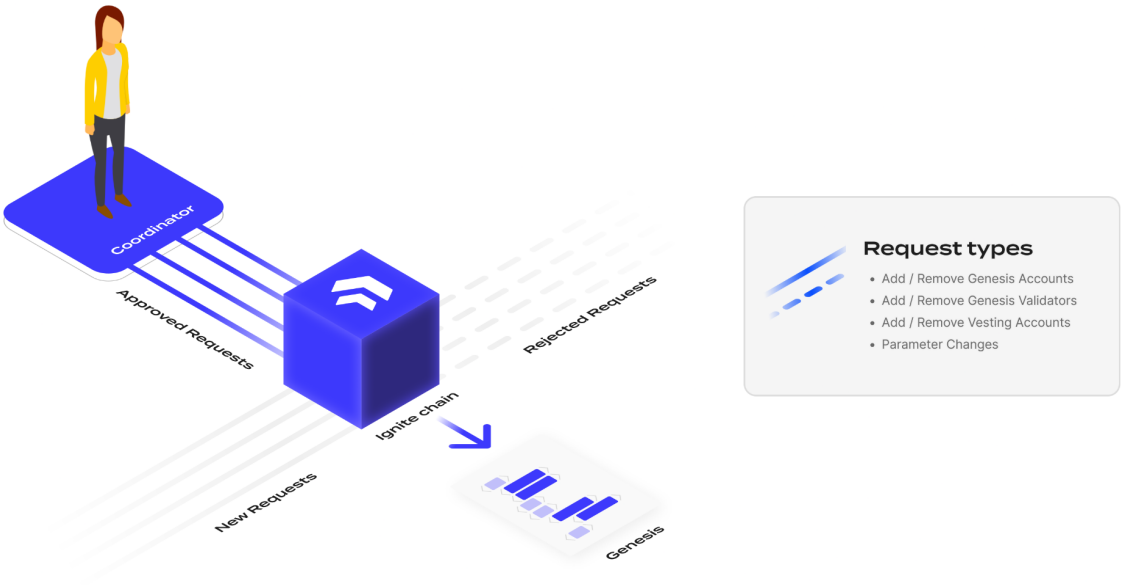
Each chain contains a request pool that is composed of all requests. Each request bears a status:

- **PENDING:** Waiting for the approval of the coordinator.
- **APPROVED:** Approved by the coordinator; its content has been applied to the launch information.
- **REJECTED:** Rejected by the coordinator or the request creator

The only possible status transitions are:

- **PENDING** to **APPROVED**
- **PENDING** to **REJECTED**

Because the coordinator is the sole approver for requests, each request created by the coordinator is immediately set to *APPROVED* and its content is applied to the launch information.



Available requests

Six types of requests can be sent to the Ignite Chain:

1. *AddGenesisAccount*
2. *AddVestingAccount*
3. *AddGenesisValidator*
4. *RemoveAccount*
5. *RemoveValidator*
6. *ChangeParam*

AddGenesisAccount requests a new account for the chain's genesis along with a coin balance. This request content is composed of two fields:

1. Account address (must be unique)
2. Account balance

The request automatically fails to be applied if a genesis account or a vesting account with an identical address is already specified in the launch information.

AddVestingAccount requests a new account for the chain genesis with a coin balance and vesting options. This request content is composed of two fields:

1. Address of the account
2. Vesting options of the account

Currently the only supported vesting option is delayed vesting in which the total balance of the account is specified and a number of tokens of the total balance of the account are vested only after an end time is reached.

The request automatically fails to be applied if a genesis account or a vesting account with an identical address is already specified in the launch information.

AddGenesisValidator requests a new genesis validator for the chain. A genesis validator in a Cosmos SDK blockchain represents an account with an existing balance in the genesis that self-delegates part of its balance during genesis initialization to become a bonded validator when the network starts. In most cases, the validator must first request an account with *AddGenesisAccount* before requesting to be a validator, unless they already have an account with a balance in the initial genesis of the chain.

Self-delegation during genesis initialization is performed with a Cosmos SDK module named genutils. In the genesis, the genutils module contains objects called *gentx* that represent transactions that were executed before network launch. To be a validator when the network starts, a future validator must provide a gentx that contains the transaction for the self-delegation from their account.

The request content is composed of five fields:

1. The gentx for the validator self-delegation
2. The address of the validator
3. The consensus public key of the validator node
4. The self-delegation amount
5. The peer information for the validator node

The request automatically fails to be applied if a validator with the same address already exists in the launch information.

RemoveAccount requests the removal of a genesis or vesting account from the launch information. The request content contains the address of the account to be removed. The request automatically fails to be applied if no genesis or vesting account with the specified address exists in the launch information.

RemoveValidator requests the removal of a genesis validator from the launch information. The request content contains the address of the validator to be removed. The request automatically fails to be applied if no validator account with the specified address exists in the launch information.

ChangeParam requests the modification of a module parameter in the genesis. Modules in a Cosmos SDK blockchain can have parameters that configure the logic of the blockchain. The parameters can be changed through governance once the blockchain network is live. During the launch process, the initial parameters of the chain are set in the genesis.

This request content is composed of three fields:

1. The name of the module
2. The name of the parameter
3. The value of the parameter (represented as generic data)

Request validity

Some checks are verified on-chain when applying a request. For example, a genesis account can't be added twice. However, some other validity properties can't be checked on-chain. For example, because a gentx is represented through a generic byte array in the blockchain, an on-chain check is not possible to verify that the gentx is correctly signed or that the provided consensus public key that is stored on-chain corresponds to the consensus public key in the gentx. This gentx verification is the responsibility of the client interacting with the blockchain to ensure the requests have a valid format and allow for the start of the chain. Some validity checks are also specified in the genesis generation algorithm.

Phases

The overall launch process of a chain through Ignite Chain is composed of three phases:

1. Coordination phase
2. Preparation phase
3. Launch phase

Coordination phase

After the coordinator creates the chain on Ignite Chain and provides the initial launch information, the launch process enters the coordination phase where users can send requests for the chain genesis. After the coordinator deems the chain as ready to be launched, they trigger the launch of the chain. During this operation, the coordinator provides the launch time, (also known as genesis time) for the chain.

Preparation phase

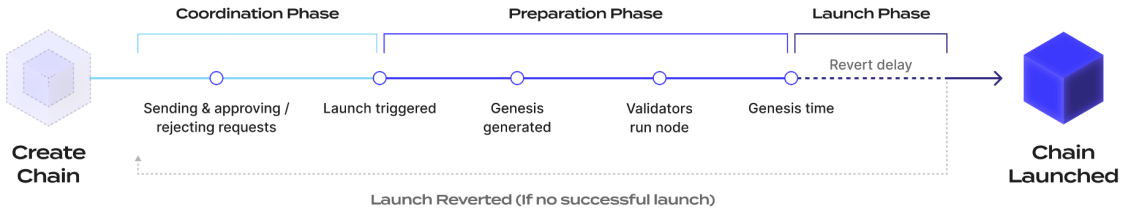
Once the launch has been triggered and before the launch time has been reached, the chain launch process enters the preparation phase. During the preparation phase, requests can no longer be sent and the launch information of the chain is finalized. The validators run the genesis generation algorithm to get the final genesis of the chain and prepare their node. The remaining time must provide enough time for the validators to prepare their nodes. This launch time is set by the coordinator, although a specific range for the remaining time is imposed.

Launch phase

Once the launch time is reached, the chain network is started and the chain launch process enters the launch phase. At this point, since the chain is live, no further action is required from the coordinator. However, under some circumstances, the chain might have failed to start. For example, a chain will not start if every validator in the genesis file does not start their node by the launch time.

The coordinator also has the ability to revert the chain launch. Reverting the chain launch sets the launch process back to the coordination phase where requests can be sent again to allow for addressing issues related to the launch failure. Reverting the launch has an effect only on Ignite

Chain. If the new chain is effectively launched, reverting the launch on Ignite Chain has no effect on the chain liveness. Reverting the launch of the chain can be performed only by the coordinator after the launch time plus a delay called *revert delay*.



Genesis generation

To ensure determinism, genesis generation rules must be rigorously specified depending on the launch information of the chain.

The general steps for the genesis generation are:

1. Building the blockchain node binary from source
2. Generating the initial genesis
3. Setting the chain ID
4. Setting the genesis time
5. Adding genesis accounts
6. Adding genesis accounts with vesting options
7. Adding gentxs for genesis validators
8. Changing module params from param changes

Projects and vouchers

In the previous section, the process of coordination between validators is described. In this model, the coordinator keeps full control over the launch information. The coordinator can deny new accounts and can remove or reduce the balance of an existing account. Adding a new account with a balance also increases the total supply for the chain and therefore dilutes the balance of every other account.

In the launch of a mainnet chain, the tokens for the chain can have a real monetary value. Therefore, in a context where some tokens for the mainnet could have been acquired from fundraising or other similar methods, having a model where the coordinator can censor accounts or dilute the supply can raise some concerns.

The project framework proposes a solution for launching chains that handles these issues.

Projects

A *project* represents the intention to launch a specific chain referred to as a mainnet on Ignite Chain. Compared to chain launches described in the previous sections, a mainnet has some differences in the launch information, and the genesis generation algorithm will follow different logic.

The difference between a mainnet and other chains launched with Ignite Chain is that with a mainnet, genesis accounts in the launch information don't directly have an explicit token balance but they contain a share balance.

Shares represent proportions of the total supply of the mainnet. Once an account has been allocated shares, this account owns a portion of the total supply. The coordinator can't remove the already allocated shares for an account and the balance of this account can't be diluted. If the total supply for the chain increases, the tokens reflected by a share of the total supply also increase.

The project is the object that manages the shares and the total supply. This object tracks two values:

1. ***TotalSupply:*** The total supply of the mainnet when it is launched.
2. ***AllocatedShares:*** The number of shares that have already been allocated for the mainnet.

The total supply is represented by explicit tokens. It can contain any number of different token denominations if the chain has several native tokens.

For example, the total supply can be:

1billionFOO, 2billionBAR

The coordinator of the project designates the total supply. They can add a new denomination (also called *denom*) with a supply or change the supply of existing denoms at any time before the mainnet is initialized. However, the coordinator can't remove existing denom from the total supply since this action would render existing allocated shares worthless.

The coordinator can change the total supply from the example above to:

10billionFOO, 1billionBAR, 3billionBAZ

The coordinator is not allowed to change the total supply from the example above to remove BAR:

1billionFOO

The *allocated shares* for the project track the shares that have already been allocated for the total supply. Every account with a share allocation will be tracked by the *allocated shares* of the project. The total number of shares that can be allocated is limited to a defined constant *TotalSharesNumber*. This constant is the same for all projects and all denoms of the total supply. This constant allows for determining the final balance of every account.

For example, let's assume:

TotalSharesNumber is set to 100,000

A share of the denom of the total supply has the following format: s/[token_denom]

For example, the allocated shares can be: 50000s/FOO, 70000s/BAR. In this example:

- 50,000/100,000 shares of FOO or 50% of the FOO total supply have been allocated.
- 70,000/100,000 shares of BAR or 70% of the BAR total supply have been allocated.

Mainnet genesis accounts

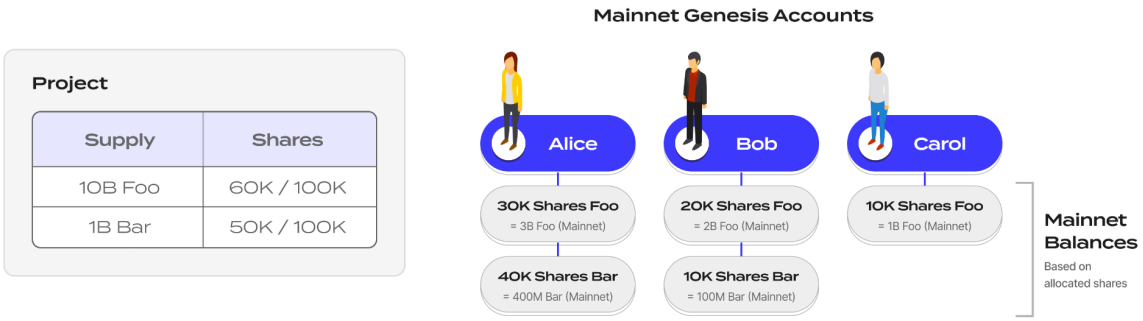
Accounts with allocated shares are called *mainnet genesis accounts* and contain two values:

1. Address of the account
2. Shares allocated to the account

The balance of a token *A* of a mainnet genesis account is dynamically calculated from the *total supply* with the following formula:

$$Balance(A) = Shares(A)/TotalSharesNumber * TotalSupply(A)$$

If the total supply is *10000000000FOO (10 Billion Foo)*, *1000000000BAR (1 Billion Bar)*, and the account's allocated shares are *1000s/FOO, 2000s/BAR*. The account owns 1% of the FOO supply and 2% of the BAR supply (implying *TotalSharesNumber=100,000*). Therefore, the balance of this account is *10,000FOO*, *20,000BAR*.



Special allocations

To avoid dilution, all genesis tokens from the total supply of a mainnet must be tracked through the allocated shares. Not all token allocations are defined through mainnet genesis accounts, the coordinator might want to set direct token or airdrop distributions directly in the initial genesis of the mainnet for eligible addresses. The coordinator can specify special allocations to the project to track these values through allocated shares.

Special allocations represent allocations that are not mainnet genesis accounts but are still taken into consideration when calculating the allocated shares of the project.

The coordinator can currently set two types of special allocations:

1. **Genesis distribution:** A share of the total supply allocated for the distribution of tokens to accounts in the initial genesis.
2. **Claimable airdrop:** A share of the total supply allocated for an airdrop that can be claimed in a *claim* module.

Special allocations can be set and updated by the coordinator until the launch of the mainnet is triggered. The validity of the special allocations is verified during the genesis generation.

The validity of the special allocations is performed off-chain and is part of the genesis generation algorithm. This check depends on the special allocation type and consists of comparing a value in the initial genesis with the number of tokens reflected by the special allocation.

For genesis distribution, the number of tokens of the special allocation must be equal to the *Supply* value in the *bank* module of the initial genesis state.

For claimable airdrop, the number of tokens of the special allocation must be equal to the *AirdropSupply* value in the *claim* module of the initial genesis state.

The *claim* module is a module that defines the logic of distributing airdrops upon the completion of specific actions. Ignite provides a generic *claim* module that can be reused by any project and that is compatible with the claimable airdrop special allocation. However, any module named *claim* and that contains an *AirdropSupply* value in its genesis state can be used for the special allocation.

Vouchers

The allocated shares of a project have a token representation called *vouchers*. Vouchers are tokenized shares. Vouchers are issued by the coordinator and can be redeemed by anyone to obtain allocated shares on the mainnet.

This framework allows the ownership of the shares for the total supply to be liquid and offers the coordinator a wide range of options for distributing the mainnet tokens. Coordinators can directly send the vouchers, sell the vouchers, use these vouchers in DeFi protocols, or any other number of use cases. Additionally, voucher holders can trade the tokens, thus creating a market for the total supply of the future mainnet tokens and with price discovery of the mainnet token before it is launched.

A voucher is similar to any regular token on Ignite Chain and it has a specific denom. The denom of a voucher is unique depending on the project and the underlying share denomination. The format of the denom is the following:

`v/{project_id}/{underlying_denom}`

50000v/1/FOO represents 50,000 vouchers for the token FOO for project 1. This voucher balance represents 50% ownership of the FOO total supply for project 1 (implying *TotalSharesNumber=100,000*).

Vouchers issuance and redeeming

Four native operations are available for issuing and manipulating vouchers:

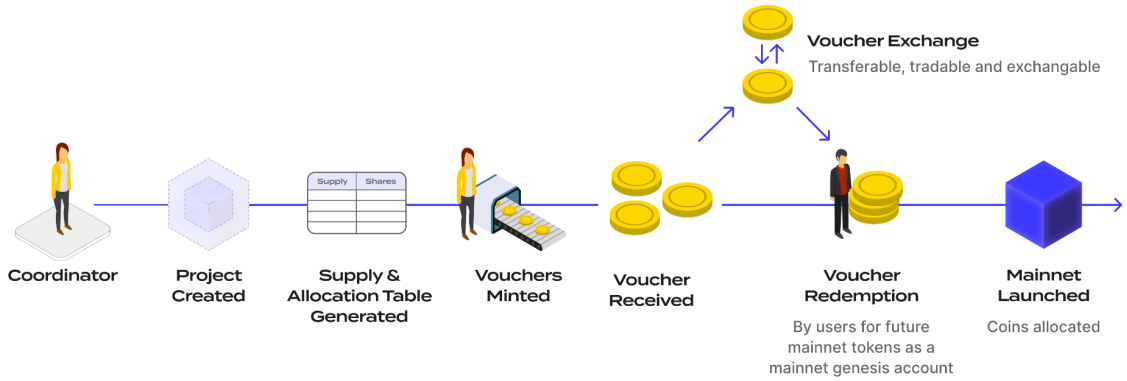
1. *MintVouchers*
2. *BurnVouchers*
3. *RedeemVouchers*
4. *UnredeemVouchers*

MintVouchers operation can only be performed by the coordinator of the project and is the main entry point for issuing new vouchers. Minting vouchers will increase the allocated shares of the project for the underlying token and will deposit the newly minted vouchers in the coordinator's balance. The coordinator has the ability to mint any number of vouchers at any time as long as the maximum number of allocated shares has not been reached.

BurnVouchers operation can be performed by any voucher holder. This operation burns the vouchers from the holder's balance and decreases the allocated shares of the project.

RedeemVouchers can be performed by any voucher holder. This operation burns the vouchers from the holder's balance while increasing the allocated shares of a mainnet genesis account specified by the holder. The allocated shares of the project are not modified.

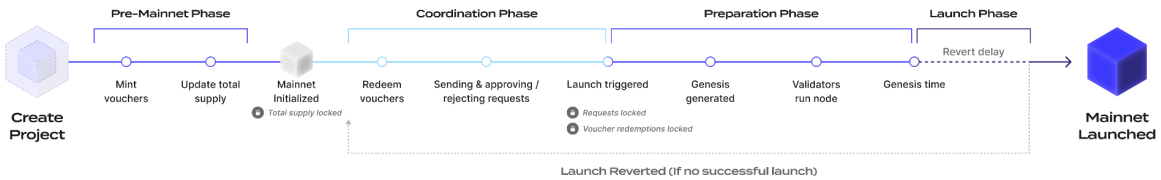
UnredeemVouchers operation can be performed by any entity owning a mainnet genesis account. This operation decreases allocated shares of the mainnet genesis account and mints, into the balance of the user performing the action, the vouchers corresponding to the decreased shares. The allocated shares of the project are not modified.



Mainnet launch process

When a project is created, the coordinator can update the total supply of the mainnet and issue vouchers. The coordinator also has the ability to initialize the mainnet of the project. When the mainnet is initialized, the total supply can no longer be updated and the mainnet chain launch is created for validators' coordination. The mainnet enters the coordination phase and requests can be submitted to update the launch information of the mainnet. After the mainnet is initialized, the coordinator still has the ability to issue vouchers, and vouchers can still be redeemed. Initializing the mainnet is an irreversible action.

Once the mainnet is initialized, the mainnet launch follows the same process as regular chain launches as described in a previous section, *Launch process*. Although, a notable difference compared to normal chains is that *AddGenesisAccount* and *AddVestingAccount* requests are unavailable for the mainnet since the genesis accounts are specified when the mainnet genesis accounts are defined.



Mainnet genesis generation

To launch a mainnet, the validators will run the *genesis generation algorithm* from the launch information like a regular chain. However, this algorithm has different logic for mainnets.

All the account balances for a mainnet genesis must be tracked by the *allocated shares* of the project. Therefore, a mainnet launch can't contain regular genesis accounts or vesting accounts. As mentioned, no *AddGenesisAccount* or *AddVestingAccount* requests can be sent or approved during the coordination phase.

During genesis generation, mainnet genesis accounts and mainnet vesting accounts are read in order to determine the balance of all accounts in the mainnet genesis.

The balance of an account in the final genesis is calculated by the following equation:

$$Balance = (MainnetGenesisAccount.Shares / TotalSharesNumber) * TotalSupply$$

Fundraising

One of the benefits of tokenizing the future total supply of the mainnet into vouchers is for the coordinator to use them for fundraising efforts. Ignite Chain includes a fundraising module that allows the creation of auctions for selling vouchers.

Two types of auctions can be created:

1. Fixed Price Auction
2. Batch Auction

Fixed Price Auction

A *fixed price auction* is an auction in which a given amount of tokens (defined as *SellingCoin*) are sold on a first-come, first-served basis. In other words, once a bidder places a bid with the predetermined bid price (defined as *StartPrice*), then the bidder receives the selling token. When all the selling tokens are sold out, the auction immediately ends.

For example, suppose that the amount of *SellingCoin* is 200 and its *denom* is CoinS. From Bidder 1 to Bidder 6, the bidders place bids with 10, 40, 30, 20, 70, and 30 CoinS, respectively. Then, when Bidder 6's bid is confirmed, all the *SellingCoin* are sold out and the auction ends.

Batch Auction

A *batch auction* allows each bidder to participate in the auction by placing limit orders with a bid price chosen freely at any time within the auction period. An order book is created to record the bids with various bid prices.

When the end time of the auction is reached, the calculation of the *matched price* is performed. According to the *matched price*, the matched bids (and also the *matched bidders*) are determined. Regardless of the bid price, all the matched bidders can get the *SellingCoin* at the same price, which is the matched price.

Requirements for participation

To enable a pool of high quality investors, requirements must be met in order to participate in auctions. Investors must delegate IGNT, the native token of Ignite Chain, to validators in order for a chance to participate in fundraisers. Depending on the amount that the investor delegated and the current fundraisers they are participating in, they will have the ability to purchase (or bid upon) a certain maximum amount of vouchers for sale in the fundraisers.

The power measurement for participating in auctions is represented through a value called *allocation*. The more allocations the user gets, the more they can participate in auctions with a higher maximum amount for bidding.

Currently, the only way for a user to receive allocation is to stake IGNT. A user receives 1 allocation for each set number of IGNT that has been staked. This number can be adjusted through governance.

In the Ignite blockchain, a structure also gives a list of tiers for auction participation. A *tier* is a number of required allocations with benefits associated with them. The benefits describe the actions allowed in the auctions. The current single value defined in benefits is *MaxBidAmount* which represents the maximum amount the user can bid in an auction. In the future, benefits can describe other actions like a number of lottery tickets received to increase the chance of participation in an auction and so on..

When a user wants to participate in an auction, they must pre-register to participate. They will use some of their available allocations to register for an auction they choose to participate in with the chosen tier of benefits.

Auction process

When an auctioneer creates a new auction, they must specify a start and end date and time for the auction. Optionally, they may specify a vesting schedule for the release of funds at the conclusion of an auction.

Registration Period

A specific amount of time before the auction starts, the auction enters a registration period. During this period, a user that wants to participate in the auction must register by using some of their

available allocations. Once registered, the participant is added to the participation list for the auction. After registering, a participant can't withdraw their participation.

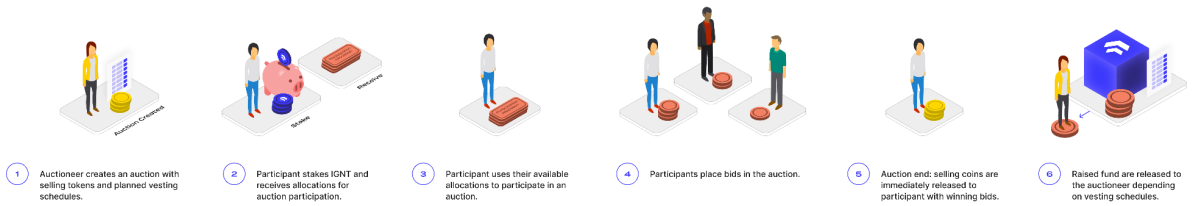
Auction Start

Once the start time of the auction is reached, the auction enters the *pending phase*. During the pending phase, the auction is in progress and users can longer register for the auction. During this phase participants can place bids.

Auction End

At the end time of the auction, the winning bids are executed. The participants with winning bids receive the purchased tokens and the raised funds are released to the auctioneer depending on the vesting periods outlined for the auction.

Independently from the end time, during a period of time after the start time of an auction called the *withdrawal delay*, the participant can withdraw their used allocation for the auction and reuse them for another auction.



Auction Cancellation

An auction may be canceled at any time prior to the start date and time. Once the start date and time have been reached, an auction cannot be canceled. When an auction is canceled, the registrant's used allocation for the auction is released to be reused for another auction.

Withdrawal Delay

The initial value for the registration period length and the withdrawal delay are respectively one and two weeks (respectively ⅓ and ⅔ of the unbonding period). The reason for these specific delays is to have the longest period of times possible that would not incentivize users to directly unbond tokens after using allocations to delegate and use allocations with another account before the used allocations of the former account are withdrawn.

Vesting Schedules

The vesting periods for the release of the raised funds are defined by the auctioneer. The auctioneer can also decide to not have vesting periods and directly release the raised fund to their address. They are however incentivized to define fair vesting schedules to bring trust for the investors.

Incentivized testnets

A testnet chain must offer a real-world environment to make the network battle-tested and therefore it must be composed of a sufficient number of validators. Since the testnet tokens, and therefore rewards for validators, have no monetary value, it can be difficult for the chain coordinator to find validators for their testnet.

Ignite Chain provides a permissionless framework to reward validators for testnet participation. The validators sign blocks on the testnet and receive reward tokens on Ignite Chain. These testnets are called *incentivized testnets*.

Incentivized testnets overview

When the coordinator wants to launch an incentivized testnet, they will first publish a normal chain on Ignite Chain as described in the *Validator coordination* section. The coordinator then creates a reward pool on Ignite Chain that is associated with their chain and sends tokens to this pool. The reward pool contains the number of coins to be distributed for the testnet participation as rewards and contains information on how the rewards must be distributed, including the maximum block height on the testnet at which all rewards are distributed and the chain is no longer incentivized. Validators can query the reward pool of a chain to see the amount of rewards that can be obtained by participating in the testnet.

In order to be incentivized, the testnet chain must import a Cosmos SDK module named *monitoring-provider*. This module monitors validator activities on the testnet and sends the activities through the IBC protocol to Ignite Chain. Ignite Chain imports a module named *monitoring-consumer* to receive the IBC packet for monitoring. The monitoring provider module will automatically write the validator activities packet when the specified height for the testnet incentivization is reached.

Reward pools

A reward pool contains the following values:

- ***LaunchID:*** The launch ID of the chain associated with a reward pool.
- ***Coins:*** The coins of the reward pool to be distributed as rewards.
- ***LastRewardHeight:*** The last height on the incentivized test that will be considered before distributing all rewards.

The *Coins* and the *LastRewardHeight* of a reward pool can be updated by the coordinator before the chain is launched. The reward pool may no longer be updated once the launch of the chain is triggered.

Verified IBC connection for monitoring

IBC is an interoperability protocol for communicating arbitrary data between arbitrary state machines.

The IBC application layer can be used to build a wide range of cross-chain applications, including but not limited to token transfers, interchain accounts (delegate calls between two chains), non-fungible token (NFT) transfers and oracle data feeds.

IBC is used by Ignite Chain in order to securely relay the information about validator activities between the incentivized testnet and Ignite Chain.

In most IBC modules, the established IBC connection is permissionless. For example, with the transfer module that enables token transfers between chains, any chain can establish an IBC connection with another chain without restriction. For the monitoring modules, the connection must be verified and is not permissionless. The *monitoring provider* module on the incentivized testnet must connect to the *monitoring consumer* module on Ignite Chain. On Ignite Chain, the connecting chain must be identified with the *launch ID* used by the incentivized testnet.

There is a single monitoring IBC channel between Ignite Chain and an incentivized testnet.

Classical IBC connection

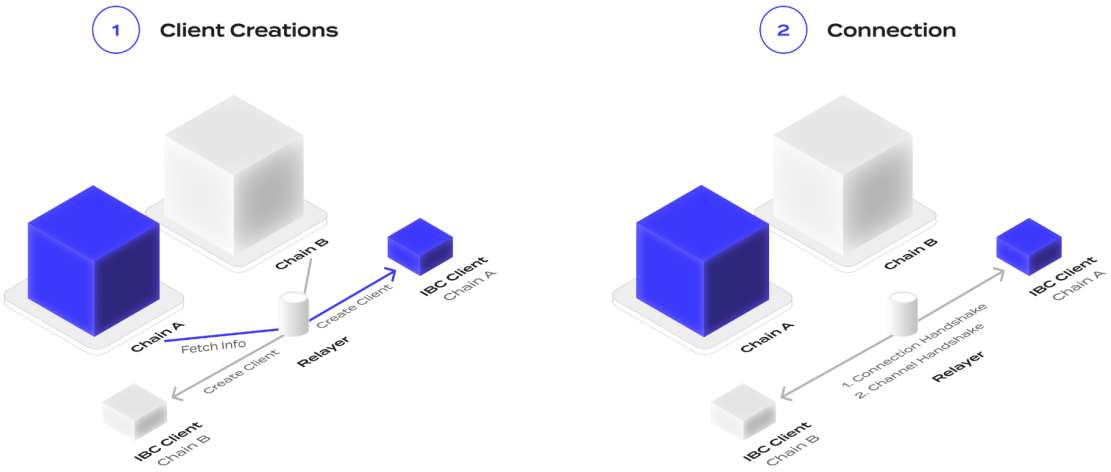
Establishing an IBC connection between two modules between two chains is performed through three steps:

1. Create an IBC client on both chains
2. Establish IBC connection between both chains
3. Establish IBC channel between both modules

With IBC, when chain A wants to connect to chain B, chain A must first create, on its ledger, an IBC client representing chain B, and vice-versa. This IBC client contains a consensus state and a client state. As its name suggests, the consensus state represents the data related to the consensus of the chain to connect to. In particular, it contains the hash of the validator set at a specific height. The client state is the state of the created client. It contains the *chain ID*, the *revision height*, and the *unbonding period* of the chain.

The consensus state and the client state in the created IBC client are verified when establishing the IBC connection with a secure handshake protocol. The provided consensus state must obligatorily be an existing consensus state in the chain to connect to. If the chain can't prove the existence of the consensus state, the handshake can't be completed.

The actor performing the IBC connection is called the *relayer*. The first step for the relayer is to fetch information from both chains in order to create IBC clients with the correct consensus state and client state. Once IBC clients are created, the relayer executes the IBC operations to create a connection and a channel.



Verified IBC connection

In order to establish a verified IBC connection, Ignite Chain uses verified IBC clients. In permissionless IBC, the relayer can create any IBC clients on any chain even with invalid consensus states, the validity of the IBC client is only checked when the IBC connection is performed.

With Ignite Chain's monitoring module, the relayer can't freely create an IBC client and use it for an IBC connection, the client must be created through specific on-chain logic.

During the IBC channel handshake, if one of the clients is not verified, the channel creation fails and modules fail to connect.

The creation of the verified IBC client is defined as follows depending on the module (provider or consumer):

- **Provider:** a unique verified IBC client is created at genesis
- **Consumer:** a specific message allows to create a verified IBC client for a launched chain

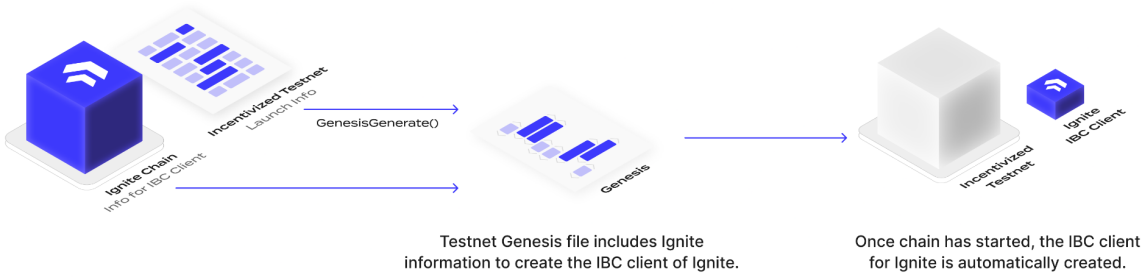
Provider – consumer client creation

The verified IBC client is created at genesis, the consensus state and information for the client state are provided in the genesis. Therefore, setting the necessary information in the genesis to create the IBC client is part of the genesis generation algorithm for incentivized testnets.

The information to be provided in the genesis for the monitoring-provider module are:

- **ConsumerChainID:** The chain ID of Ignite Chain.
- **ConsensusState:** The consensus state of Ignite Chain at a specific height.
- **UnbondingPeriod:** The unbonding period of Ignite Chain.
- **RevisionHeight:** The height used to fetch information on Ignite Chain.

The *chain ID*, *unbonding period*, and *revision height* allow the creation of the client state. If the IBC client fails to be created, the testnet network fails to start.



Testnet Genesis file includes Ignite information to create the IBC client of Ignite.

Once chain has started, the IBC client for Ignite is automatically created.

Consumer – provider client creation

A verified IBC client is created for a testnet by sending a message to Ignite Chain to create the client. The message includes the following values:

- ***LaunchID:*** The launch ID of the testnet on Ignite Chain.
- ***ConsensusState:*** The consensus state of the client to create.
- ***ValidatorSet:*** The validator set of the testnet.
- ***UnbondingPeriod:*** The unbonding period of the testnet.
- ***RevisionHeight:*** height used for the consensus state and validator set

In order for the IBC client to be valid, the information provided to create the client is verified with the information that has been previously determined during the coordination phase to launch the testnet: validators of the chain.

The consensus state used to create the IBC client contains in its data the hash of the next validator set. This hash is compared with the hash of the provided validator set dynamically calculated. When both hashes are equal, the provided consensus state and validator set association is valid. Then the validator set is compared with the validator information defined during the coordination phase for the chain. The validator set contains the consensus public key of the validators. During the coordination phase, the consensus public key of validators is stored on Ignite Chain with their associated self-delegation for the *gentx*.

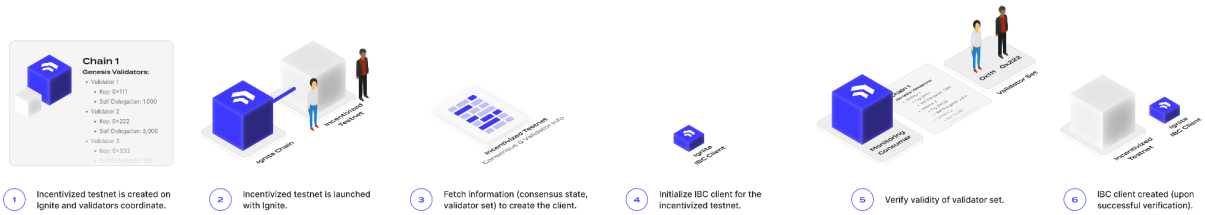
The validator set is valid if the following two conditions are met:

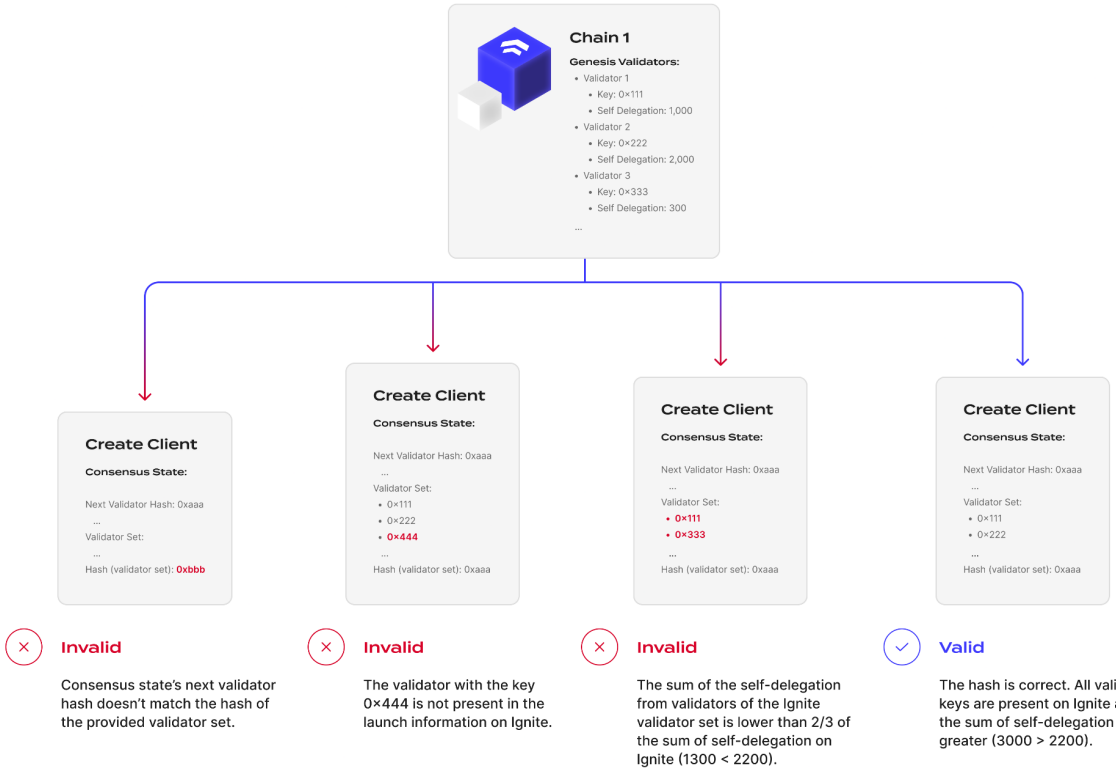
1. Each consensus public key in the validator set is present in the chain launch information on Ignite Chain.

2. The sum of the self-delegation of the validators from the validator set is greater than ⅔ of the sum of the self-delegation of the validators defined on Ignite Chain.

If the validator set is valid, the IBC client is created with the associated consensus state.

Finally, the unbonding period and revision height can be freely chosen when creating the client. These values will be used to create the client state for the IBC client. The validity of the client state is checked once the IBC connection is performed.

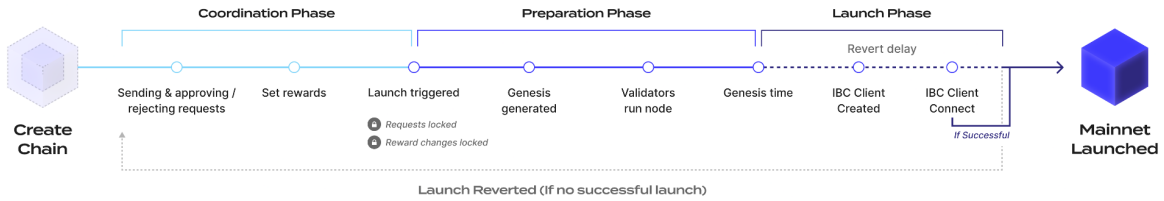




Incentivized testnet launch process

An incentivized testnet is launched through the same process as a normal chain as described in the *Validator coordination* section. The reward pool is defined during the coordination phase and can no longer be updated after the launch is triggered.

During the launch phase, when the IBC connection is established for monitoring, the launch can no longer be reverted, even after the revert delay has passed.



Incentivized testnet genesis generation

Two additional pieces of information must be set in the genesis during the genesis generation for incentivized testnets:

1. The last block height for monitoring
2. Ignite Chain information to create the IBC client

The last block height for monitoring is directly fetched from the reward pool's last reward height.

The Ignite Chain information is queried by validators from Ignite Chain during the genesis generation. Since the genesis generation must be deterministic, the validators must query the consensus state of Ignite Chain at the same height. In order to have a common height used by all validators to query the consensus state, a height value, *ConsumerRevisionHeight*, is set when the launch of the testnet is triggered on Ignite Chain. This height is queried by the validators and used to set an identical consensus state in the testnet genesis.

Monitoring

The *Monitoring Provider* module reports the validator signatures in the *Begin Blocker*, a method executed at the beginning of each block and that provides the last commit information, including the validator votes. The routine, executed each block, reporting signatures is called *ReportBlockSignatures*. This routine updates the monitoring info with the new signatures and is executed until the block height reaches the last reward height value.

Once the last reward height is reached, the monitoring packet is generated from the monitoring information and is transmitted to Ignite Chain.

The module starts reporting validator activities once the network has started and until the last reward height is reached, independently of the monitoring connection. If the monitoring connection is still not enabled when the last reward height is reached, the monitoring information is saved and the monitoring packet is only written once the connection is established.

Monitoring packet

The monitoring packet is the IBC packet transmitted to Ignite Chain that contains all the information that allows distributing the rewards to the validators for an incentivized testnet.

It contains particularly the number of blocks taken into consideration for counting signatures and a value for each validator called *RelativeSignatures*. *RelativeSignatures* is the sum of all signatures of a validator relative to the validator set size at each block. If at a specific height, the validator set size is 2 and the validator signs the block. The validator's *RelativeSignatures* value will increase by 0.5, if the validator set is 10, it would increase by 0.1 because they signed a "tenth" of a block. This structure allows the monitoring packet to be self-contained and to distribute the rewards without any other context.

Reward distribution

When the Ignite Chain receives a monitoring packet, it automatically distributes the rewards to the validators from the reported activity. Ignite Chain fetches the reward pool associated with the monitoring packet then executes the *DistributeRewards* method.

The following values are calculated for the distribution:

- ***BlockRatio*:** The ratio between the number of blocks reported in the monitoring packet and the total remaining number of blocks of the testnet to be incentivized. In practice, this number will be 1 since the monitoring packet is only relaying once the planned last reward height is reached. The algorithm is currently designed in a way to make it future-proof once periodic relaying of the monitoring packet is allowed. Eventually, the governance of the incentivized testnet could also vote to relay the monitoring packet sooner.
- For all validators in the monitoring packet:
    - ***SignatureRatio*:** The percentage of rewards a validator gets from its number of signatures.

$$SignatureRatio = RelativeSignatures / BlockCount$$

- ***RefundRatio*:** The percentage of rewards refunded to the provider, this value increases when some validators didn't sign for some of the blocks.

$$RefundRatio = (BlockCount - sum(RelativeSignatures))/ BlockCount$$

The rewards for each validator are calculated as follow:

$$rewardPool.RemainingCoins * BlockRatio * SignatureRatio$$

The refunds for the coordinator are calculated as follow:

$$rewardPool.RemainingCoins * BlockRatio * RefundRatio$$

Terminology

Launch coordination

- ***Coordinator***: An entity that initiates and manages chain launches and projects.
- ***Chain:*** In Ignite Chain terminology, a chain refers to a new chain network launched (or launching) on Ignite Chain.
- ***Launch ID***: Identifier of a chain on Ignite Chain.
- ***Chain Information***: Data about a blockchain that validators require to perform a decentralized launch of a chain network.
- ***Launch Information***: The set of information relied on by a chain to deterministically launch the network of the chain. Launch information includes the required information to generate the genesis file and the list of persistent peers for the network.
- ***Request***: A request for a change in the launch information of the chain. A request must be approved by a coordinator in order to be effective.
- ***Request Pool***: The list of requests that are waiting to be approved or rejected by the coordinator for a chain.
- ***Validator***: An entity that participates in a chain launch or a project with an intent to become a validator ([doc](https://docs.tendermint.com/master/nodes/validators.html)).

Projects

- ***Project***: A project represents intent to launch a mainnet chain network. A project can be a sequence of testnets that precede and result in a mainnet launch.
- ***Total Supply***: The total amount of coins allocated for a project's mainnet.
- ***Share***: A share of the total supply. Each coin defined in the total supply has its associated share. If an entity holds 100 of the 1000 ATOM shares, the entity will receive 10% of the supply of ATOMs.
- ***Voucher***: A tokenized share that can be transferred.

Fundraising

- ***Allocations***: The unit of measure to determine the benefits an auction participant can obtain when in auction participation. Allocations depend on the number of staking tokens the entity has bonded.

- ***Registration Period***: The period before an auction starts where an entity can apply to be a participant in this auction.
- ***Withdrawal Delay***: The delay after an auction starts before a participant is authorized to withdraw the allocations used for the auction.

Incentivized testnets

- ***Incentivized Testnet***: A chain launched on Ignite Chain whose the validator activity is rewarded on Ignite Chain in the form of tokens.
- ***Reward***: Tokens that already exist on Ignite Chain which are sent into an escrow for direct payment to validators to incentivize and reward them to participate in a blockchain network.
- ***Reward Pool***: A pool associated with a testnet that contains rewards to be distributed. The pool determines how the contained rewards should be distributed.
- ***Incentivized Block***: A block that is eligible for a reward for the validator that signs it.
- ***Last Reward Height***: The block height of the last incentivized block in a testnet for a reward pool.
- ***Current Reward Height***: The current tracked block height for a chain for reward distribution. This value is increased when a monitoring packet is received.
- ***Distribution Round***: A round of reward distribution for a chain when a monitoring packet is received on Ignite Chain.
- ***Signature Counts***: A structure that contains the number of signatures with their associated validator address in a monitoring packet to determine the rewards for the validator.
- ***Monitoring Period***: A number of blocks specific to the testnet to monitor validator activities.
- ***Monitoring Packet***: An IBC packet that is sent from a testnet to Ignite Chain that contains data from the monitoring period.
- ***Last Block Height***: The height of the last block that was monitored in a monitoring round.

References

Ignite CLI: https://docs.ignite.com/

Cosmos SDK: https://docs.cosmos.network/main/

IBC protocol: https://ibcprotocol.org/

Fundraising module: https://github.com/tendermint/fundraising/tree/main/docs

Ignite claim module: https://github.com/ignite/modules/tree/main/x/claim/spec