Token Vesting Strategies

If you plan to launch a token on Sui, then you might consider implementing a vesting strategy to strengthen the long-term outlook of your token. A vesting strategy typically releases your tokens to team members, investors, or other early stakeholders over time, rather than releasing them all at once.

Implementing and publishing the details of your vesting strategy helps to:

There are different vesting strategies available for your token launch. The best option for your project depends on a number of factors unique to your project and its goals.

The following sections highlight some available options to consider when launching a token on the Sui network.

Cliff vesting refers to a situation where the entire amount of tokens or assets becomes available after a specific period (the "cliff"). Until the cliff period is met, no tokens are released.

Each of the ten employees of a project are granted 1,000 tokens with a one-year cliff. After one year, they receive the full 1,000 tokens. Before the year is up, they have no access to the tokens.

The following smart contract implements a cliff vesting schedule for token releases. The module includes a new_wallet function that you pass the total sum of coins to vest and the cliff date as a timestamp. You can then call the claim function to retrieve the tokens from the wallet if the cliff date is in the past.

Considering the example scenario, you would call new_wallet ten times so that a separate wallet existed for each employee. You would include 1,000 tokens in each call to load the wallet with the necessary funds. Subsequent calls to claim using the relevant wallet would compare the cliff time (cliff_time in the Wallet object) against the current time, returning tokens from the wallet if the cliff time is later than the current time.

cliff.move

Graded vesting allows tokens to be gradually released over time, often in equal portions, during the vesting period.

An employee receives 1,200 tokens, with 300 tokens vesting every year over four years. At the end of each year, 300 tokens become available.

The following Hybrid vesting section includes a smart contract that demonstrates how to perform graded vesting.

Hybrid vesting combines different vesting models, such as cliff and graded vesting. This allows flexibility in how tokens are released over time.

50% of tokens are released after a one-year cliff, and the rest are distributed linearly over the next three years.

The following smart contract creates a hybrid vesting model. Like the cliff vesting smart contract, the hybrid model defines a Wallet struct to hold all the tokens for each stakeholder. This wallet, however, actually contains two different wallets that each follow a different set of vesting rules. When you call the new_wallet method for this contract, you provide the cliff cutoff timestamp, the timestamp for when the linear schedule begins, and a timestamp for when the linear vesting should end. Calls to claim then return the sum of tokens that fall within those parameters.

hybrid.move

Backloaded vesting distributes the majority of tokens near the end of the vesting period, rather than evenly over time. This approach can help an ecosystem become more mature before large amounts of tokens become unlocked. Team members and stakeholders can be rewarded early but save the biggest rewards for those that remain with the project for a greater length of time.

An employee's tokens release under the following schedule:

The smart contract for backloaded vesting creates two Wallet objects inside a parent wallet, which contains all the tokens to be vested. Each of the child wallets is responsible for its own vesting schedule. You call new_wallet with the coins to vest and start_front , start_back , duration , and back_percentage values. Based on the values you provide, the contract determines how many tokens to return when the wallet owner calls the claim function.

For the example scenario, you could pass the start timestamp for the frontload and the start timestamp for the backload (three years after the frontload start). You would also pass the duration of four years (126230400000) and 90 for the back_percentage value.

backloaded.move

With performance-based vesting, achieving specific goals or metrics trigger vest events, such as hitting revenue targets or progressing through project stages.

A team's tokens vest in relation to the number of monthly active users (MAUs). All tokens become vested after the platform reaches its goal of 10 million MAUs.

Similarly, milestone-based vesting creates vest events when specific project or personal milestones are achieved, rather than being tied to time-based conditions.

Tokens unlock when the mainnet of a blockchain project launches.

Like the other examples, the following smart contract creates a wallet to hold the coins to be distributed. Unlike the others, however, this Wallet object includes a milestone_controller field that you set to the address of the account that has the authority to update the milestone progress. The call to the new_wallet function aborts with an error if the wallet has the same address as the entity with milestone update privileges as an integrity check.

The milestone update authority can call update_milestone_percentage to update the percentage-to-complete value. The owner of the vested token wallet can call claim to retrieve the tokens that are unlocked based on the current percentage-to-complete value. Considering the first example scenario, you could update the milestone value by ten percent for every million MAUs the project achieves. You could use the same contract for the second scenario, updating the percentage one time to 100 only after mainnet launches.

milestone.move

With linear vesting, tokens are released gradually over a set time period.

An employee is granted 1,000 tokens to be gradually released over a one-year period.

The linear vesting smart contract creates a Wallet object with start and duration fields. The contract uses those values along with the current time to determine the number of tokens that are vested. The current time, in this case, is the time at which the wallet owner calls the claim function.

For the example scenario, you create the wallet (call new_wallet) with 1,000 tokens, the timestamp for the employee start date, and one year (31557600000) as the duration.

linear.move

All tokens vest immediately, meaning they are fully available as soon as they are allocated.

An early investor receives their full allocation of tokens at the time of purchase.

With immediate investing, you could always just transfer tokens to an address. Opting for a smart contract approach provides several advantages over a manual transfer, however.

The following test leverages the linear vesting smart contract example to demonstrate how to use one of the other vesting strategy smart contracts to support immediate vesting. The test uses the Wallet object and new_wallet function from vesting::linear to perform (or test) an immediate vest scenario. The test accomplishes this by setting the duration value to 0.

immediate_tests.move

Vesting options

There are different vesting strategies available for your token launch. The best option for your project depends on a number of factors unique to your project and its goals.

The following sections highlight some available options to consider when launching a token on the Sui network.

Cliff vesting refers to a situation where the entire amount of tokens or assets becomes available after a specific period (the “cliff”). Until the cliff period is met, no tokens are released.

Each of the ten employees of a project are granted 1,000 tokens with a one-year cliff. After one year, they receive the full 1,000 tokens. Before the year is up, they have no access to the tokens.

The following smart contract implements a cliff vesting schedule for token releases. The module includes a new_wallet function that you pass the total sum of coins to vest and the cliff date as a timestamp. You can then call the claim function to retrieve the tokens from the wallet if the cliff date is in the past.

Considering the example scenario, you would call new_wallet ten times so that a separate wallet existed for each employee. You would include 1,000 tokens in each call to load the wallet with the necessary funds. Subsequent calls to claim using the relevant wallet would compare the cliff time (cliff_time in the Wallet object) against the current time, returning tokens from the wallet if the cliff time is later than the current time.

cliff.move

Graded vesting allows tokens to be gradually released over time, often in equal portions, during the vesting period.

An employee receives 1,200 tokens, with 300 tokens vesting every year over four years. At the end of each year, 300 tokens become available.

The following Hybrid vesting section includes a smart contract that demonstrates how to perform graded vesting.

Hybrid vesting combines different vesting models, such as cliff and graded vesting. This allows flexibility in how tokens are released over time.

50% of tokens are released after a one-year cliff, and the rest are distributed linearly over the next three years.

The following smart contract creates a hybrid vesting model. Like the cliff vesting smart contract, the hybrid model defines a Wallet struct to hold all the tokens for each stakeholder. This wallet, however, actually contains two different wallets that each follow a different set of vesting rules. When you call the new_wallet method for this contract, you provide the cliff cutoff timestamp, the timestamp for when the linear schedule begins, and a timestamp for when the linear vesting should end. Calls to claim then return the sum of tokens that fall within those parameters.

hybrid.move

Backloaded vesting distributes the majority of tokens near the end of the vesting period, rather than evenly over time. This approach can help an ecosystem become more mature before large amounts of tokens become unlocked. Team members and stakeholders can be rewarded early but save the biggest rewards for those that remain with the project for a greater length of time.

An employee's tokens release under the following schedule:

The smart contract for backloaded vesting creates two Wallet objects inside a parent wallet, which contains all the tokens to be vested. Each of the child wallets is responsible for its own vesting schedule. You call new_wallet with the coins to vest and start_front , start_back , duration , and back_percentage values. Based on the values you provide, the contract determines how many tokens to return when the wallet owner calls the claim function.

For the example scenario, you could pass the start timestamp for the frontload and the start timestamp for the backload (three years after the frontload start). You would also pass the duration of four years (126230400000) and 90 for the back_percentage value.

backloaded.move

With performance-based vesting, achieving specific goals or metrics trigger vest events, such as hitting revenue targets or progressing through project stages.

A team's tokens vest in relation to the number of monthly active users (MAUs). All tokens become vested after the platform reaches its goal of 10 million MAUs.

Similarly, milestone-based vesting creates vest events when specific project or personal milestones are achieved, rather than being tied to time-based conditions.

Tokens unlock when the mainnet of a blockchain project launches.

Like the other examples, the following smart contract creates a wallet to hold the coins to be distributed. Unlike the others, however, this Wallet object includes a milestone_controller field that you set to the address of the account that has the authority to update the milestone progress. The call to the new_wallet function aborts with an error if the wallet has the same address as the entity with milestone update privileges as an integrity check.

The milestone update authority can call update_milestone_percentage to update the percentage-to-complete value. The owner of the vested token wallet can call claim to retrieve the tokens that are unlocked based on the current percentage-to-complete value.

Considering the first example scenario, you could update the milestone value by ten percent for every million MAUs the project achieves. You could use the same contract for the second scenario, updating the percentage one time to 100 only after mainnet launches.

milestone.move

With linear vesting, tokens are released gradually over a set time period.

An employee is granted 1,000 tokens to be gradually released over a one-year period.

The linear vesting smart contract creates a Wallet object with start and duration fields. The contract uses those values along with the current time to determine the number of tokens that are vested. The current time, in this case, is the time at which the wallet owner calls the claim function.

For the example scenario, you create the wallet (call new_wallet) with 1,000 tokens, the timestamp for the employee start date, and one year (31557600000) as the duration.

linear.move

All tokens vest immediately, meaning they are fully available as soon as they are allocated.

An early investor receives their full allocation of tokens at the time of purchase.

With immediate investing, you could always just transfer tokens to an address. Opting for a smart contract approach provides several advantages over a manual transfer, however.

The following test leverages the linear vesting smart contract example to demonstrate how to use one of the other vesting strategy smart contracts to support immediate vesting. The test uses the Wallet object and new_wallet function from vesting::linear to perform (or test) an immediate vest scenario. The test accomplishes this by setting the duration value to 0.

immediate_tests.move