Introduction

These docs are in active development by the Keep3r community.

Keep3r Network is a decentralized keeper network for projects that need external devops and for external teams to find keeper jobs

Keepers

A Keeper is the term used to refer to an external person and/or team that executes a job. This can be as simplistic as calling a transaction, or as complex as requiring extensive off-chain logic. The scope of Keep3r network is not to manage these jobs themselves, but to allow contracts to register as jobs for keepers, and keepers to register themselves as available to perform jobs. It is up to the individual keeper to set up their devops and infrastructure and create their own rules based on what transactions they deem profitable.

Jobs

A Job is the term used to refer to a smart contract that wishes an external entity to perform an action. They would like the action to be performed in "good will" and not have a malicious result. For this reason they register as a job, and keepers can then execute on their contract.

Becoming a Keeper

To join as a Keeper you call `bond(uint)` on the Keep3r contract. You do not need to have KPR tokens to join as a Keeper, so you can join with `bond(0)`. There is a 3 day bonding delay before you can activate as a Keeper. Once the 3 days have passed, you can call `activate()`. Once activated you `lastJob` timestamp will be set to the current block timestamp.

Registering a Job

A job can be any system that requires external execution, the scope of Keep3r is not to define or restrict the action taken, but to create an incentive mechanism for all parties involved. There are two cores ways to create a Job;

Registering a Job via Governance

If you prefer, you can register as a job by simply submitting a proposal via Governance, to include the contract as a job. If governance approves, no further steps are required.

Registering a Job via Contract Interface

You can register as a job by calling `addLiquidityToJob(address,uint)` on the Keep3r contract. You must not have any current active jobs associated with this account. Calling `addLiquidityToJob(address,uint)` will create a pending Governance vote for the job specified by address in the function arguments. You are limited to submit a new job request via this address every `14 days` .

Job Interface

Some contracts require external event execution, an example for this is the `harvest()` function in the yearn ecosystem, or the `update(address,address)` function in the uniquote ecosystem. These normally require a restricted access control list, however these can be difficult for fully decentralized projects to manage, as they lack devops infrastructure.

These interfaces can be broken down into two types, no risk delta (something like `update(address,address)` in uniquote, which needs to be executed, but not risk to execution), and `harvest()` in yearn, which can be exploited by malicious actors by front-running deposits.

For no, or low risk executions, you can simply call `Keep3r.isKeeper(msg.sender)` which will let you know if the given actor is a keeper in the network.

For high, sensitive, or critical risk executions, you can specify a minimum bond, minimum jobs completed, and minimum Keeper age required to execute this function. Based on these 3 limits you can define your own trust ratio on these keepers.

So a function definition would look as follows;

```
function execute() external {
  require(Keep3r.isKeeper(msg.sender), "Keep3r not allowed");
}
```

At the end of the call, you simply need to call `workReceipt(address,uint)` to finalize the execution for the keeper network. In the call you specify the keeper being rewarded, and the amount of KPR you would like to award them with. This is variable based on what you deem is a fair reward for the work executed.

Example Keep3rJob

```
interface UniOracleFactory {
    function update(address tokenA, address tokenB) external;
}

interface Keep3r {
    function isKeeper(address) external view returns (bool);
    function workReceipt(address keeper, uint amount) external;
}

contract Keep3rJob {
    UniOracleFactory constant JOB = UniOracleFactory(0x61da8b0808CEA5281A91
    Keep3r constant KPR = Keep3r(0x1cEB5cB57C4D4E2b2433641b95Dd330A33185A44

    function update(address tokenA, address tokenB) external {
        require(KPR.isKeeper(msg.sender), "Keep3rJob::update: not a valid k
        JOB.update(tokenA, tokenB);
        KPR.workReceipt(msg.sender, 1e18);
    }
}
```

Job Credits

As mentioned in Job Interface, a job has a set amount of `credits` that they can award keepers with. To receive `credits` you do not need to purchase KPR tokens, instead you need to provide KPR-WETH liquidity in Uniswap. This will give you an amount of credits equal to the amount of KPR tokens in the liquidity you provide.

You can remove your liquidity at any time, so you do not have to keep buying new credits. Your liquidity provided is never reduced and as such you can remove it whenever you no longer would like a job to be executed.

To add credits, you simply need to have KPR-WETH LP tokens, you then call `addLiquidityToJob(address,uint)` specifying the job in the address and the amount in the uint. This will then transfer your LP tokens to the contract and keep them in escrow. You can remove your liquidity at any time by calling `unbondLiquidityFromJob()`, this will allow you to remove the liquidity after 14 days by calling `removeLiquidityFromJob()`

Github

Keep3r