Yield vault harvesting

ETHA Lend

ETHA is using Gelato Automate for automated yield harvesting.

This is the function which Gelato will be calling.harvestVault claims yield generated from a pool and re-deposits them back into the pool.

```
Copy functionharvestVault(IVaultvault)publiconlyAfterDelay(vault) { // Amount to Harvest uint256afterFee=vault.harvest(); require(afterFee>0,"!Yield");

IERC20 from=vault.rewards(); IERC20 to=vault.target();

addressconnector=getBestConnector( address(from), address(to), afterFee );

// Quickswap path address[]memorypath;

if(connector==address(0)) { path=newaddress; path[0]=address(from); path[1]=address(to); }else{ path=newaddress; path[0]=address(from); path[1]=connector; path[2]=address(to); }

// Swap underlying to target from.approve(address(ROUTER),afterFee); uint256received=ROUTER.swapExactTokensForTokens( afterFee, 1, path, address(this), block.timestamp+1 )[path.length-1];

// Send profits to vault to.approve(address(vault),received); vault.distribute(received);

emitHarvested(address(vault),msg.sender); }
```

ETHA uses this resolver below to check for ready to be harvested pools.

```
Copy functionchecker() external view returns(boolcanExec,bytesmemoryexecPayload) { uint256delay=harvester.delay();

for(uint256i=0; i<vaults.length(); i++) { IVault vault=IVault(getVault(i));

canExec=block.timestamp>=vault.lastDistribution().add(delay);

execPayload=abi.encodeWithSelector( IHarvester.harvestVault.selector, address(vault) );

if(canExec)break; } }
```

The resolver loops through an array of pools. And for each vault, if a defineddelay has elapsed since the previous harvest time,canExec will returntrue , prompting Gelato to execute the task.execPayload will be the data to the function callharvestVault(address vault) and its argument is the address of the vault to be harvested.