# Cheshire¶

Cheshire enables fast CryptoKitties dApp development by providing local implementations of the CryptoKitties web API and smart contracts. It features:

1. AnEthereum testnet
2. running the CryptoKitties smart contracts
3. An HTTP server running aminimal implementation of the CryptoKitties web API
4. :
5. /kitties
6. /kitties/:id
7. /user/:address
8. A simpleNode.js framework
9. for seeding the development environment with realistic data and bootstraping your dApp

Cheshire has simplified and accelerated development atEndless Nameless considerably. We're excited to share it.

## Installation¶

You can install Cheshire with git or as aTruffle Box .

### Git¶

git clone http://github.com/endless-nameless-inc/cheshirecd

cheshire yarn install

### Truffle box¶

truffle unbox endless-nameless-inc/cheshire

## Usage¶

Cheshire is meant to be used with theTruffle Framework , but can function as a standalone service, depending on your workflow.

### Start Cheshire¶

To start Cheshire, run:

yarn start

This does the following:

1. Starts an Ethereum testnet (ganache-cli
2. )
3. Deploys CryptoKitties'sKittyCore
4. ,SaleClockAuction
5. , andSiringClockAuction
6. contracts to the testnet
7. Starts a local CryptoKitties API server
8. Executes/scripts/setup.js

The output should look something like this:

> Starting database... Starting testnet... Compiling contracts... Deploying CryptoKitties contracts to testnet...
> Starting local CryptoKitties API server... Running setup script...

Cheshire is live 😸 Here's what's inside:

# Available Accounts

(0) 0x182fc09c33fdd6c2f5b2562f3ca721fa954689c8 ... (9) 0xcdf40e926a778d93429b72c341b4a9e0ee8624c4

# Private Keys

(0) 0x76a67ae288fd67ea8d4f7fb94f50c36b606d9448db579584af90d52105f9d8cf ... (9) 0x6e77cfded732de6d423abcaccc45ee8c4bdc2eb3c0c47938acb386ac17c496b8

# Testnet Contracts

KittyCore: 0xa751b62893867d0608a2ada5d17d0c43e3433040 SaleClockAuction: 0x1ab49d53d0bff0202ec4b330349b427155bba7ac SiringClockAuction: 0x671843106e07f9d835d7299381cd14863af18593

# Services

Ethereum testnet listening on port 8546 CryptoKitties API listening on port 4000 Cheshire dashboard available at http://localhost:4000

View the above at any time by running`yarn run help` Eureka! When Cheshire's running, you have your very own local copy of CryptoKitties, enabling you to build your dApp with the speed and convenience of testnet. Let's try it out.

### Interacting with your local CryptoKitties API¶

Cheshire automatically imports the Genesis kitty. To fetch the Genesis kitty from your local CryptoKitties API, run:

curl http://localhost:4000/kitties/1 The response should look exactly like the response returned by CryptoKitties'sproduction API .

See thescripts section below to learn how to seed your environment with more data.

### Interacting with the testnet contracts¶

To interact with the testnet contracts, start by opening a Truffle console:

truffle console --network cheshire Then, taking note of the KittyCore testnet address displayed when you started Cheshire, create an instance of KittyCore, and use thegetKitty function to fetch the Genesis kitty's genes:

truffle( cheshire)

```
// Be sure to replace the KittyCore address below truffle( cheshire)

kittyCore
```

=

KittyCore.at( '0xa751b62893867d0608a2ada5d17d0c43e3433040' ) truffle( cheshire)

```
kittyCore.getKitty( 1 ) The response should be pretty similar to the one you get from themainnet contract .
```

## Suggested Conventions¶

You'll get the most out of Cheshire by adopting these conventions:

- Store your contracts in the/contracts
- directory
- Design the web application layers of your stack to reference Cheshire's environment variables
- (hat tip to thetwelve-factor
- methodology)
- Update yoursetup script
- to deploy your contracts to testnet
- Update yoursetup script
- to start your dApp's web application

## Scripts¶

Cheshire provides a simple scripting framework designed to help seed the development environment with realistic data, primarily by importing kitties from mainnet.

A Cheshire script is just a Node.js module that runs in the context of the Cheshire environment.

Here's an example of a script that imports aBug Cat from mainnet to your testnet.

// /scripts/import-bug-cat.js module . exports

=

async

function

importBugCat ( cheshire )

{

const

bugCatIdMainnet

=

101

const

ownerTestnet

=

cheshire . accounts [ 0 ]. address

const

kittyIdTestnet

=

await

cheshire . importKitty ( bugCatIdMainnet ,

ownerTestnet )

console . log (Kitty # { kittyIdTestnet } => { ownerTestnet } ) } To run this script, you would execute the following command:

yarn run script ./scripts/import-bug-cat.js The output would look something like:

Kitty #2 => 0x182fc09c33fdd6c2f5b2562f3ca721fa954689c8

## Setup Script¶

Cheshire executes/scripts/setup.js when started. You should update thesetup.js shipped with Cheshire to:

1. Deploy your dApp's contracts to testnet. For example:

const kittyRace = await cheshire.deployContract('KittyRace', process.env.ADDRESS_KITTY_CORE) log('KittyRace deployed at:', kittyRace.address) 1. Start your dApp's web application, so it inherits the variousenvironment variables 2. set by Cheshire.

We recommend adopting the convention in thesetup.js shipped with Cheshire which simply expects theAPP_START environment variable to contain a command that starts your dApp's web application.

For example:

APP_START="cd ~/Projects/kittyrace-web; bundle exec rails server" yarn start You can run any script in place ofsetup.js by passing its path toyarn start . This is handy for setting up specific scenarios, such as a KittyRace with 9 registered racers:

yarn start ./scripts/setup-registered-racers.js 9

## Cheshire API Reference¶

Cheshire scripts receive an instance of the Cheshire class with these methods:

**accounts()**

¶

Returns array of available Ethereum accounts (the same accounts defined in config.json)

**contractAddress(contractName)**

¶

Returns address ofcontractName

**contractInstance(contractName)**

¶

Returns an instance ofcontractName as aweb3.eth.contract object

**createKitty(matronId, sireId, generation, genes, owner, apiObject)**

¶

Create a kitty with the given parameters.

Returns the kitty's ID.

**async deployContract(contractName, ...constructorArgs)**

¶

DeploycontractName to testnet.

Cheshire compiles all contracts in/contracts at start time. Expects/contracts/ContractName.sol to exist.

Returns an instance ofcontractName as aweb3.eth.contract object

**async importKitty(kittyIdMainnet, ownerTestnet)**

¶

Import a kitty from mainnet, and assign it toownerTestnet

Returns the testnet kitty's ID.

**async importUser(addressMainnet, addressTestnet)**

¶

Import user's profile and kitties from mainnet, and assign toaddressTestnet .

Returns address of testnet user.

### Cheshire Environment Variables¶

Cheshire sets several environment variables before running any script:

- ADDRESS_KITTY_CORE
- ADDRESS_SALE_CLOCK_AUCTION
- ADDRESS_SIRING_CLOCK_AUCTION
- URL_CRYPTO_KITTIES_API

In addition to these, the address for any contract deployed with a Cheshire script will be stored in an environment variable named with the convention,ADDRESS_ .

## Configuration¶

Theconfig.json file defines the following:

- accounts
-
    - list of Ethereum accounts to load into testnet

- ethNodeMainnet
  - URL for the node used to access the Ethereum mainnet
- addressKittyCoreMainnet
  - address of the mainnet KittyCore contract
- portTestnet
  - port bound by Ethereum testnet
- portApi
  - port bound by local CryptoKitties API

# Utilities¶

## Mine¶

To mine some number of blocks on your testnet:

yarn run mine

## Help¶

Print information about the environment including available Ethereum accounts, contract addresses, etc.

yarn run help

## Cheshire Dashboard¶

Cheshire ships with a simple dashboard you can access athttp://localhost:4000

# Developer notes¶

## KittyCore¶

The smart contracts bundled with Cheshire are identical to those in production except for KittyCore, to which we've added anexternal createKitty function that lets us push kitties into the local testnet contract.

function

createKitty (

uint256

_matronId ,

uint256

_sireId ,

uint256

_generation ,

uint256

_genes ,

address

_owner )

external

returns

( uint ) {

return

_createKitty( _matronId,

_sireId,

_generation,

_genes,

_owner); }

### Contributions¶

Cheshire works pretty well for us atEndless Nameless , but there's probably a whole lot more it could do!

If you're interested in contributing, we humbly request the following:

1. Adhere to Airbnb'sJavaScript style guide
2. (yarn eslint
3. makes it easy)
4. Include tests. We're happy whenyarn test
5. is happy, andyarn test
6. is only happy when coverage is 100%

## Acknowledgements¶

We're grateful for the contributions of the many open source projects on which Cheshire depends, none more so than the excellentTruffle Suite .

Cheshire is byEndless Nameless . It is based on tools and processes we developed while buildingKittyRace , a CryptoKitties dApp. We hope Cheshire makes it easier to #buidl

Your name here - we will gladly review PRs.