LiquidApps

Jan 14, 2020

Contents

CHAPTER 1

Developers

1.1 Getting Started

1.1.1 Overview Resources

1.1.2 Zeus SDK

- Getting Started
- Unit Testing
- Packages and Staking
- Zeus boxes

1.1.3 Dapp Client Library

- Getting Started

1.1.4 vRAM

- With Zeus

1.1.5 LiquidAccounts

- LiquidAccounts Getting Started

1.1.6 LiquidHarmony

- LiquidHarmony Getting Started

1.1.7 LiquidScheduler

- LiquidScheduler Getting Started

1.2 Overview

1.2.1 Articles

- Zeus IDE: Ready? Set. Code!
- LiquidApps DAPP Network Walkthrough #1: Zeus and vRAM
- LiquidApps Walkthrough #2: Staking to DAPP Service Providers and Deploying a vRAM dApp
- LiquidX Brings DSP Services To All EOSIO Chains and dApps
- Horizontally Scaling Blockchain Apps with vCPU
- EOS dApps and Their RAM Expenses
- vRAM guide for experts

1.2.2 Videos

- Multi-Chain dApp Scaling: Intro to LiquidApps & the DAPP Network (Blockchain Tools by Peter Keay)
- Setting up LiquidApps' Zeus SDK, including NVM + NPM (Blockchain Dev Tools by Peter Keay)
- Intro to Scalable, Decentralized Storage with DAPP Network vRAM (Blockchain Tools by Peter Keay)
- Developer Explains - Decentralized Dapp Scaling w/ IPFS! How LiquidApps Dapp Service Providers Work
- EOS Weekly - The LiquidApps Game-Changer
- EOS Weekly - Unlimited DSP Possibilities

1.2.3 Have questions?

- Join our Dev Telegram channel
- Join our Telegram channel
- Email us: support@liquidapps.io

1.2.4 Want more information?

- Read our whitepaper and subscribe to our Medium posts.

1.3 Zeus Getting Started

1.3.1 Overview

Zeus-cmd is an extensible command line tool. SDK extensions come packaged in "boxes" and are served through IPFS. Zeus is currently in alpha. *As a note, all Zeus commands must be run within the root directory of the package that was unboxed.*

- zeus-sdk
- overview of boxes

1.3.2 Features:

- Smart contract templating with a single command
- Install nodeos, keosd, cleos, and eosio.cdt with a single command
- Simulate a blockchain node with a single command
- Test, compile, and deploy smart contracts
- Easily migrate a contract to a different EOSIO chain such as the Kylin and Jungle testnets or the mainnet
- Design fluid dApp frontends
- Cross-platform (Windows, OS X, Linux)
- Easily install necessary libraries with a package manager
- Truffle-like interface
- Manage development lifecycle with version control
- Open source (BSD License)
- And more…

1.3.3 Gitpod Zeus-IDE

If you want to be up and running with Zeus quickly, you can use our cloud based Zeus-IDE, all you need is a Github account! Try it here!

1.3.4 dapp-client library

The dapp-client library makes it easier to interact with the DAPP Network's core smart contracts and services, read more here.

1.3.5 Hardware Requirements

- 16GB RAM
- 2 CPU Cores

1.3.6 Prerequisites

- *node version 10.16.3* is recommended (nvm recommended, install at bottom of doc)
- curl
- cmake
- make

Use node version manager to install node

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash
# use install instructions provided to set PATH
nvm install 10.16.3
nvm use 10.16.3
```

1.3.7 Recommended eosio.cdt and eosio versions

Automatically installed with `zeus unbox helloworld`

- eosio.cdt v1.7.0
- eosio v1.8.9

1.3.8 Install Zeus

```
npm install -g @liquidapps/zeus-cmd
```

1.3.9 Update

```
npm update -g @liquidapps/zeus-cmd
```

1.3.10 Test

```
zeus unbox helloworld
cd helloworld
zeus test
```

1.3.11 Create your own contract

This box supports all DAPP Services and unit tests and is built to integrate your own DAPP Network logic. When you run the command a sample unit test and smart contract will be created.

```
mkdir mydapp; cd mydapp
zeus unbox dapp --no-create-dir
zeus create contract mycontract
```

1.3.12 Try out LiquidApps's take on Elemental Battles:

http://elemental.liquidapps.io/ | code

The game incorporates:

- vRAM - light-weight caching solution for EOSIO based RAM
- LiquidAccounts - EOSIO accounts that live in vRAM instead of RAM
- LiquidDNS - DNS service on the blockchain | contract table
- Frontend stored on IPFS
- user data is stored in the vRAM `dapp::multi_index` table (vRAM) | code
- keys stored in `dapp::multi_index` table | code
- keys created using the account name and password as seed phrases | code
- eosjs-ecc's `seedPrivate` method is used to create the keypair | code
- logic to create LiquidAccount transactions | code

To launch locally:

```
zeus unbox cardgame
cd cardgame
zeus migrate
zeus run frontend main
```

1.3.13 Try out vCPU with our LiquidChess game:

https://chess.liquidapps.io/ | code

The game incorporates:

- vRAM - light-weight caching solution for EOSIO based RAM
- LiquidAccounts - EOSIO accounts that live in vRAM instead of RAM
- LiquidDNS - DNS service on the blockchain
- vCPU - a solution to scale blockchain processing power horizontally

To launch locally:

```
zeus unbox chess
cd chess
zeus migrate
zeus run frontend main
```

1.3.14 Try out LiquidPortfolio

LiquidPortfolio is a portfolio tracking tool for BTC, ETH (and tokens), and EOS (and tokens). The tool displays the total current value of the portfolio while also encrypting all user account info with the LiquidAccount's private key.

The game incorporates:

- vRAM - light-weight caching solution for EOSIO based RAM
- LiquidAccounts - EOSIO accounts that live in vRAM instead of RAM

- LiquidHarmony - oracle service for fetching prices
- LiquidDNS - DNS service on the blockchain
- Encryption/Decryption locally of account data using LiquidAccount private key

To launch locally:

```
zeus unbox portfolio
cd portfolio
zeus migrate
zeus run frontend main
```

1.3.15 Samples Boxes

```
zeus unbox <INSERT_BOX>
```

vRAM Boxes

- coldtoken - vRAM based eosio.token
- deepfreeze - vRAM based cold storage contract
- vgrab - vRAM based airgrab for eosio.token
- registry - vRAM based item registration

Zeus Extension Boxes

- contract-migrations-extensions - contract create/deployment command template, deploy contract and allocate DAPP tokens
- test-extensions - provides logic to test smart contract with unit tests
- eos-extensions - install eos/eosio.cdt, launch local nodeos, launch system contracts
- unbox-extensions - logic to unbox zeus boxes, list all boxes, and deploy a new box
- demux - install EOSIO's demux backend to capture events for contracts using the state-history plugin

DAPP Services Boxes

The DAPP Service boxes allow you to isolate the service that you wish to work with. If you instead would like to use all of the DAPP Services, you may unbox the `all-dapp-services` box.

- ipfs-dapp-service - utilize the dapp::multi_index table to store data in IPFS (vRAM) instead of RAM
- cron-dapp-service - schedule CRON tasks on-chain
- oracle-dapp-service - provide oracle services
- readfn-dapp-service - read a contract function without the need to submit a trx to the chain
- vaccounts-dapp-service - EOSIO accounts that live in vRAM instead of RAM
- vcpu-dapp-service - scale blockchain processing power horizontally

Miscellaneous Boxes

- microauctions - twin reverse dutch auctions used in DAPP's generation event
- eos-detective-reports - EOS Detective Reports - by EOSNation - https://eosdetective.io/
- helloworld - Hello World
- token - Standard eosio.token
- airhodl - First ever Air-HODL

1.3.16 Zeus Options

please note: zeus commands are directory sensitive, all commands should be performed in root of box

Zeus compile

Compile a smart contract. You can either compile all of the contracts within the contracts directory with `zeus compile` or a specific contract by name, such as `zeus compile dappservices`

```
zeus compile <CONTRACT_NAME>

# optional flags:

--all # compile all contracts
# default: true
--chain # chain to work on
# default: eos
```

Zeus migrate

Compile and migrate a smart contract to another network such as the Kylin Testnet, Jungle Testnet, or Mainnet.

Be sure to run the following commands from inside the directory you unboxed, e.g., if you unboxed coldtoken, be in `/coldtoken`. Also be sure to set the network in the import and migrate commands so Zeus knows what chain the keys / contract is operating on (mainnet, kylin, or jungle).

```
# keys are stored in ~/.zeus/networks/<NETWORK>/accounts/
zeus key import <CONTRACT_ACCOUNT_NAME> --owner-private-key <KEY> --active-private-
↪key <KEY> --network=kylin
# contract deployment files are stored in ~/<BOX>/models/contract-deployments
zeus create contract-deployment <CONTRACT_FILE_NAME> <CONTRACT_ACCOUNT_NAME>
zeus migrate --network=kylin --creator=<CONTRACT_ACCOUNT_NAME> --creator-key=<ACTIVE_
↪PRIVATE_KEY>

# optional flags:

--compile-all # compile all contracts
# default: true
--wallet # keosd wallet to use
# default: zeus
--creator-key # contract creator private key
# default: (eosio test key) 5KQwrPbwdL6PhXujxW37FSSQZ1JiwsST4cqQzDeyXtP79zkvFD3
--creator # account to set contract to
```

(continues on next page)

(continued from previous page)

```
# default: eosio
--reset # reset testing environment
# default: true
--chain # chain to work on
# default: eos
--network # network to work on (other options, kylin, jungle, mainnet)
# default: development (local)
--verbose-rpc # verbose logs for blockchain communication
# default: false
--storage-path # path for persistent storage',
# default: path.join(require('os').homedir(), '.zeus')
--stake # account EOSIO staking amount
# default: '30.0000'
--no-compile # do not compile contracts
--no-reset # do not reset local testing environment
```

Zeus test

Compile and unit test a smart contract. You can either compile and test all of the contracts within the contracts & test directories with `zeus test` or a specific contract by name, with `zeus test dappservices`

```
zeus test <CONTRACT_NAME>

# optional flags:

--compile # compile contracts
# default: false
# alias: c
--wallet # keosd wallet to use
# default: zeus
# alias: w
--creator-key # contract creator key
# default: (eosio test key) 5KQwrPbwdL6PhXujxW37FSSQZ1JiwsST4cqQzDeyXtP79zkvFD3
--creator # account to set contract to
# default: eosio
# alias: a
--reset # reset testing environment
# default: true
--chain # chain to work on
# default: eos
--network # network to work on (other options, kylin, jungle, mainnet)
# default: development (local)
--verbose-rpc # verbose logs for blockchain communication
# default: false
--storage-path # path for persistent storage',
# default: path.join(require('os').homedir(), '.zeus')
--stake # account EOSIO staking amount
# default: '30.0000'
--no-reset # do not reset local testing environment
```

Zeus Import/Export Keys

Import and export keys to your Zeus wallet. **Please note by default keys are imported without encryption.**

```
zeus key import <ACCOUNT_NAME> --owner-private-key <KEY> --active-private-key <KEY>

# optional flags:

--encrypted # encrypt the account keys with a password
# default: false
--storage # path to the wallet which will store the key
# default: ${home}/.zeus/networks
--network # network to work on (other options, kylin, jungle, mainnet)
# development (local)
--password # password to encrypt the keys with
--vaccount # bool whether account is a LiquidAccount
# default: false

zeus key export <ACCOUNT_NAME>

# optional flags:

--encrypted # exports encrypted key
# default: false
--storage # path to where the key is stored
# default: ${home}/.zeus/networks
--network # network to work on (other options, kylin, jungle, mainnet)
# default: development (local)
--password # password to decrypt the keypair
--vaccount # bool whether account is a LiquidAccount
# default: false
```

Zeus Deploy

Deploy a custom Zeus box to your local working directory. Once deployed, if the `--update-mapping` flag is used, you may unbox this box like other packages. The `--type` method can be used to determine what medium to deploy the box to. The default `local` deploys with the syntax `file://${packagePath}/box.zip`. The option `ipfs` deploys to the IPFS network with the syntax `ipfs://${hash}`.

```
zeus deploy box

# optional flags:

--update-mapping # updates local mapping.js file with an IPFS URI where the package␣
↪may be accessed at
# default: true
--type # deploy destination (local, ipfs)
# default: local
```

Help

```
zeus --help
```

List Boxes

Lists all available zeus boxes that can be unboxed.

```
zeus list-boxes
```

Update Boxes

Updates zeus boxes for currently unboxed project

```
zeus update --boxes
```

Create a new box

Creates a new directory with an empty `zeus-config.json` file.

```
zeus box create <BOX_NAME>
```

Add a box to the mapping.js file

In order to add a new box to zeus, you either need to edit the `mapping.js` where Zeus was installed, or you can use the following command:

```
zeus box add <BOX_NAME> <URI>
# for example: zeus box add liquidx-jungle https://s3.us-east-2.amazonaws.com/
↪liquidapps.artifacts/boxes/
↪0a98835c75debf2f1d875be8be39591501b15352f7c017799d0ebf3342668d2c.zip
```

1.3.17 Project structure

Directory structure

```
    extensions/
    contracts/
    frontends/
    models/
    test/
    migrations/
    utils/
    services/
    zeus-box.json
    zeus-config.js
```

zeus-box.json

Add commands, NPM intalls, ignores, and command hooks

```
    {
      "ignore": [
        "README.md"
      ],
      "commands": {
        "Compile contracts": "zeus compile",
```

(continues on next page)

(continued from previous page)

```
      "Migrate contracts": "zeus migrate",
      "Test contracts": "zeus test"
    },
    "install":{
        "npm": {

        }
    },
    "hooks": {
      "post-unpack": "echo hello",
      "post-install": "git clone ..."
    }
  }
```

zeus-config.js

Configure zeus environments available to interact with. The `zeus-config.js` file is located in the root of an unboxed directory.

```
  module.exports = {
      defaultArgs:{
        chain:"eos",
        network:"development"
      },
      chains:{
          eos:{
              networks: {
                  development: {
                      host: "localhost",
                      port: 7545,
                      network_id: "*", // Match any network id
                      secured: false
                  },
                  jungle: {
                      host: "jungle2.cryptolions.io",
                      port: 80,
                      network_id: "*", // Match any network id
                      secured: false
                  },
                  kylin: {
                      host: "api.kylin.eosbeijing.one",
                      port: 80,
                      network_id: "*",
                      secured: false
                  },
                  mainnet:{
                      host: "bp.cryptolions.io",
                      port: 8888,
                      network_id: "*", // Match any network id
                      secured: false
                  }
              }
          }
      }
  };
```

1.4 Dapp Client Library

The dapp-client library makes using DAPP Network services much easier. It also allows you to easily tap into the `dappservices` (core DAPP Network Smart contract) and `dappairhodl1` (Air-HODL smart contract) RAM tables with precision utilizing secondary and tertiary indices without the hassle.

To setup the library, first install it:

```
npm i -g @liquidapps/dapp-client
```

From there include the library's client creator script:

```
import { createClient } from '@liquidapps/dapp-client';
const { createClient } = require('@liquidapps/dapp-client');
```

Then pass your desired arguments to the creator function

```
/*
    * network: specify your network of choice: mainnet, kylin, jungle
    * httpEndpoint: you may also specify an endpoint to use instead of our defaults
    * fetch: pass a fetch reference as needed
*/

export const getClient = async() => {
  return await createClient({ network: "kylin", httpEndpoint: endpoint, fetch: window.
↪fetch.bind(window) });
};
```

Finally, setup the service you would like to interact along with your smart contract name:

```
(async () => {
    const service = await (await createClient()).service('ipfs','cardgame1112');
    const response = await service.get_vram_row( "cardgame1112", "cardgame1112",
↪"users", "nattests" );
    console.log(response);
    // { username: 'nattests',
    //     win_count: 0,
    //     lost_count: 0,
    //     game_data:
    // { life_player: 5,
    //     life_ai: 5,
    //     deck_player:
    //     [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17 ],
    //     deck_ai:
    //     [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17 ],
    //     hand_player: [ 0, 0, 0, 0 ],
    //     hand_ai: [ 0, 0, 0, 0 ],
    //     selected_card_player: 0,
    //     selected_card_ai: 0,
    //     life_lost_player: 0,
    //     life_lost_ai: 0,
    //     status: 0 } }
})().catch((e) => { console.log(e); });
```

Here is a full list of service options. There are DAPP Network service extensions and `dappservices` / `dappairhodl1` RAM row calls.

1.4.1 DAPP Network service extensions

vRAM - IPFS

get_vram_row - get vRAM row from DSP's endpoint

```
/*
    getClient()
    * service name: ipfs
    * contract name

    service.get_vram_row - read vRAM table row from DSP's endpoint
    * code - smart contract account
    * scope - scope of table
    * table - name of table
    * primary key

*/

const service = await (await getClient()).service('ipfs','cardgame1112');
const response = await service.get_vram_row( "cardgame1112", "cardgame1112", "users",
↪"nattests" );
```

LiquidAccounts

push_liquid_account_transaction - register new account

```
/*
    getClient()
    * service name: vaccounts
    * contract name

    service.push_liquid_account_transaction
    * account name of contract with LiquidAcount code deployed
    * private key of LiquidAccount
    * action name: regaccount to register new account
    * payload
        * vaccount - name of LiquidAccount

*/

const service = await (await getClient()).service('vaccounts', "vacctstst123");
const response = await service.push_liquid_account_transaction(
    "vacctstst123",
    "5JMUyaQ4qw6Zt816B1kWJjgRA5cdEE6PhCb2BW45rU8GBEDa1RC",
    "regaccount",
    {
        vaccount: 'testing126'
    }
);
```

push_liquid_account_transaction - push transaction after creating account

```
/*
    getClient()
    * service name: vaccounts
    * contract name

    service.push_liquid_account_transaction
    * account name of contract with LiquidAcount code deployed
    * private key of LiquidAccount
    * action name: hello
    * payload
        * any - payload must match struct outline in the smart contract

*/

const service = await (await getClient()).service('vaccounts', "vacctstst123");
const response = await service.push_liquid_account_transaction(
    "vacctstst123",
    "5JMUyaQ4qw6Zt816B1kWJjgRA5cdEE6PhCb2BW45rU8GBEDa1RC",
    "hello",
    {
        vaccount: 'testing126'
    }
);
```

1.4.2 dappservices

get_package_info - returns package info

```
/*

    dappNetwork.get_package_info
    * account name
    * service name - service names are listed under the services section of the docs␣
↪as the Contract name

*/

const response = await (await getClient()).dappNetwork.get_package_info( "cardgame1112
↪", "accountless1" );
console.log(response);
// {
//     api: 'https://kylin-dsp-2.liquidapps.io',
//     package_json_uri: 'https://kylin-dsp-2.liquidapps.io/liquidaccts2.dsp-package.
↪json',
//     package_id: 'liquidaccts2',
//     service: 'accountless1',
//     provider: 'heliosselene',
//     quota: '10.0000 QUOTA',
//     package_period: 60,
//     min_stake_quantity: '10.0000 DAPP',
//     min_unstake_period: 3600,
//     enabled: 0
// }
```

get_table_accountext - returns entire accountext table

```
/*

    dappNetwork.get_table_accountext

*/

const response = await (await getClient()).dappNetwork.get_table_accountext();
for (const row of response.rows) {
    console.log(row);
    // {
    //     id: 144,
    //     account: 'mailcontract',
    //     service: 'ipfsservice1',
    //     provider: 'heliosselene',
    //     quota: '9.9907 QUOTA',
    //     balance: '10.0000 DAPP',
    //     last_usage: '1564112241500',
    //     last_reward: '1564112241500',
    //     package: 'ipfs1',
    //     pending_package: 'ipfs1',
    //     package_started: '1564112241500',
    //     package_end: '1564112301500'
    // }
}
```

get_table_accountext_by_account_service - returns entire accountext by account and service specified

```
/*

    dappNetwork.get_table_accountext_by_account_service
    * account name
    * service name - service names are listed under the services section of the docs␣
↪as the Contract name

*/

const response = await (await getClient()).dappNetwork.get_table_accountext_by_␣
↪account_service('cardgame1112', 'ipfsservice1');
for (const row of response.rows) {
    console.log(row);
    // {
    //     id: 144,
    //     account: 'mailcontract',
    //     service: 'ipfsservice1',
    //     provider: 'heliosselene',
    //     quota: '9.9907 QUOTA',
    //     balance: '10.0000 DAPP',
    //     last_usage: '1564112241500',
    //     last_reward: '1564112241500',
    //     package: 'ipfs1',
    //     pending_package: 'ipfs1',
    //     package_started: '1564112241500',
    //     package_end: '1564112301500'
```

(continues on next page)

(continued from previous page)

```
    // }
}
```

get_table_accountext_by_account_service_provider - returns entire accountext by account, service, and provider specified

```
/*

    dappNetwork.get_table_accountext_by_account_service_provider
    * account name
    * service name - service names are listed under the services section of the docs as the Contract name
    * provider name - DSP name

*/

const response = await (await getClient()).dappNetwork.get_table_accountext_by_account_service_provider('cardgame1112', 'ipfsservice1','heliosselene');
for (const row of response.rows) {
    console.log(row);
    // {
    //     id: 144,
    //     account: 'mailcontract',
    //     service: 'ipfsservice1',
    //     provider: 'heliosselene',
    //     quota: '9.9907 QUOTA',
    //     balance: '10.0000 DAPP',
    //     last_usage: '1564112241500',
    //     last_reward: '1564112241500',
    //     package: 'ipfs1',
    //     pending_package: 'ipfs1',
    //     package_started: '1564112241500',
    //     package_end: '1564112301500'
    // }
}
```

get_table_package - returns entire package table

```
/*

    dappNetwork.get_table_package
    * [limit] - optional limit for how many packages to return

*/

const response = await (await getClient()).dappNetwork.get_table_package({ limit: 500});
for (const row of response.rows) {
    console.log(row);
    // {
    //     id: 9,
    //     api_endpoint: 'https://kylin-dsp-2.liquidapps.io',
    //     package_json_uri: 'https://kylin-dsp-2.liquidapps.io/package1.dsp-package.json',
```

(continues on next page)

(continued from previous page)

```
    //     package_id: 'package1',
    //     service: 'ipfsservice1',
    //     provider: 'heliosselene',
    //     quota: '1.0000 QUOTA',
    //     package_period: 86400,
    //     min_stake_quantity: '10.0000 DAPP',
    //     min_unstake_period: 3600,
    //     enabled: 1
    // }
}
```

get_table_package_by_package_service_provider - returns packages by package, service, and DSP name

```
/*

    dappNetwork.get_table_package_by_package_service_provider
    * package name
    * service name - service names are listed under the services section of the docs␣
↪as the Contract name
    * DSP name
    * [limit] - optional limit for how many packages to return

*/

const response = await (await getClient()).dappNetwork.get_table_package_by_package_
↪service_provider('package1', 'ipfsservice1', 'heliosselene', { limit: 500 });
for (const row of response.rows) {
    console.log(row);
    // {
    //     id: 9,
    //     api_endpoint: 'https://kylin-dsp-2.liquidapps.io',
    //     package_json_uri: 'https://kylin-dsp-2.liquidapps.io/package1.dsp-package.
↪json',
    //     package_id: 'package1',
    //     service: 'ipfsservice1',
    //     provider: 'heliosselene',
    //     quota: '1.0000 QUOTA',
    //     package_period: 86400,
    //     min_stake_quantity: '10.0000 DAPP',
    //     min_unstake_period: 3600,
    //     enabled: 1
    // }
}
```

get_table_refunds - returns refund table details for account name specified

```
/*

    dappNetwork.get_table_refunds
    * account name

*/
```

(continues on next page)

(continued from previous page)

```
const response = await (await getClient()).dappNetwork.get_table_refunds('heliosselene
↪');
for (const row of response.rows) {
    console.log(row);
    // {
    //     id: 0,
    //     account: 'heliosselene',
    //     amount: '10.0000 DAPP',
    //     unstake_time: 12345678
    //     provider: 'heliosselene',
    //     service: 'ipfsservice1'
    // }
}
```

get_table_staking - returns staking table details for account name specified

```
/*

    dappNetwork.get_table_staking
    * account name

*/

const response = await (await getClient()).dappNetwork.get_table_staking('cardgame1112
↪');
for (const row of response.rows) {
    console.log(row);
    // {
    //     id: 0,
    //     account: 'cardgame1112',
    //     balance: '10.0000 DAPP',
    //     provider: 'uuddlrlrbass',
    //     service: 'accountless1'
    // }
}
```

1.4.3 dappairhodl1

get_dapphdl_accounts - get an account's DAPPHDL stats

```
/*

    airhodl.get_dapphdl_accounts
    * account name

*/

const response = await (await getClient()).airhodl.get_dapphdl_accounts('natdeveloper
↪');
for (const row of response.rows) {
    console.log(row);
```

(continues on next page)

(continued from previous page)

```
    // {
    //     balance: '0.0033 DAPPHDL',
    //     allocation: '0.0199 DAPPHDL',
    //     staked: '0.0000 DAPPHDL',
    //     claimed: 1
    // }
}
```

1.5 Kylin Testnet Account

The CryptoKylin testnet is one of the EOS Testnets. Feel free to join their Telegram, or checkout their Github repo.

1.5.1 Account

```
# Create a new available account name (replace 'yourtestaccount' with your account␣
↪name):
export ACCOUNT=yourtestaccount

# Configure endpoint
export DSP_ENDPOINT=https://kylin-dsp-1.liquidapps.io

# Create wallet
cleos wallet create --file wallet_password.pwd

# Create account and import key
curl http://faucet-kylin.blockzone.net/create/$ACCOUNT > keys.json
export ACTIVE_PRIVATE_KEY=`cat keys.json | jq -e '.keys.active_key.private'`
export ACTIVE_PUBLIC_KEY=`cat keys.json | jq -e '.keys.active_key.public'`
cleos wallet import --private-key $ACTIVE_PRIVATE_KEY
# if this does not work, import key directly

# Get some tokens, stake CPU/NET, buy RAM for contract
curl http://faucet-kylin.blockzone.net/get_token/$ACCOUNT
curl http://faucet-kylin.blockzone.net/get_token/$ACCOUNT
cleos -u $DSP_ENDPOINT system buyram $ACCOUNT $ACCOUNT "100.0000 EOS" -p
↪$ACCOUNT@active
cleos -u $DSP_ENDPOINT system delegatebw $ACCOUNT $ACCOUNT "20.0000 EOS" "80.0000 EOS
↪" -p $ACCOUNT@active
```

Save wallet_password.pwd and keys.json somewhere safe!

1.5.2 Kylin DAPP Tokens

DAPP Faucet

1.6 Zeus IDE

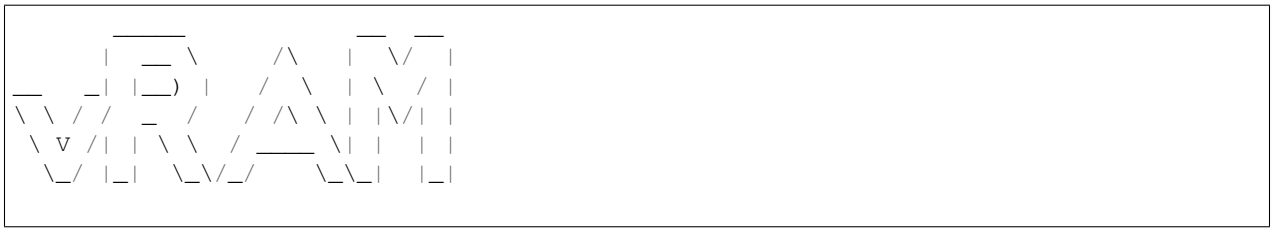
The Zeus IDE allows new developers to get up and running in mere minutes! It is based on Gitpod, a cloud-based tool for creating IDEs from git repos backed by Docker containers. All a you need to do is log in to Gitpod with your GitHub account and establish a new workspace based on the LiquidApps Zeus IDE GitHub repo.

Behind the scenes, Gitpod executes the following:

- Automatically installs the Docker image, which already contains EOSIO, Zeus and many other tools
- Starts an EOSIO development node
- Generates a basic starting point for a project (using the Zeus SDK)
- Starts a LiquidApps development DSP

Click here to try it!

1.7 vRAM Getting Started



vRAM is a caching solution that enables DAPP Service providers (specialized EOS nodes) to load data to and from RAM <> vRAM on demand. Data is evicted from RAM and stored in vRAM after the transaction has been run. This works similar to the way data is passed to and from regular computer RAM and a hard drive. As with EOS, RAM is used in a computer sometimes because it is a faster storage mechanism, but it is scarce in supply as well. For more information on the technical details of the transaction lifecycle, please read the vRAM Guide For Experts article and/or the whitepaper.

vRAM requires a certain amount of data to be stored in RAM permanently in order for the vRAM system to be trustless. This data is stored in a regular `eosio::multi_index` table with the same name as the `dapp::multi_index` vRam table defined by the smart contract. Each row in the regular `eosio::multi_index` table represents the merkle root of a partition of the sharded data with the root hash being `vector<char> shard_uri` and the partition id being `uint64_t shard`. Note that this is equivalent to having a single merkle root with the second layer of the tree being written to RAM for faster access. The default amount of shards (which is proportional to the maximum amount of permanent RAM required) is 1024 meaning that, the total amount of RAM that a `dapp::multi_index` table will need to permanently use is `1024 * (sizeof(vector<char> shard_uri) + sizeof(uint64_t id))`.

In order to access/modify vRam entries certain data may need to be loaded into RAM in order to prove (via the merkle root) that an entry exists in the table. This temporary data (the "cache") is stored in the `ipfsentry` table. The DAPP Services Provider is responsible for removing this data after the transaction's lifecycle. If the DSP does not perform this action, the `ipfsentry` table will continue to grow until the account's RAM supply has been exhausted or the DSP resumes its services.

1.7.1 Prerequisites

- Zeus - Zeus installs eos and the eosio.cdt if not already installed
- Kylin Account

1.7.2 Unbox sample template

This box supports all DAPP Services and unit tests and is built to integrate your own vRAM logic.

```
mkdir mydapp; cd mydapp
zeus unbox dapp --no-create-dir
zeus create contract mycontract
```

1.7.3 Or use one of our template contracts

```
# unbox coldtoken contract and all dependencies
zeus unbox coldtoken
cd coldtoken
# unit test coldtoken contract locally
zeus test
```

1.7.4 Advanced features

To use advanced multi index features include `#define USE_ADVANCED_IPFS` at the top of the contract file while following the steps below. If you have already deployed a contract that does not use advanced features, do not add this line, as it is not backwards compatible.

1.7.5 Add your contract logic

in contract/eos/mycontract/mycontract.cpp

```
#pragma once

#include "../dappservices/ipfs.hpp"
#include "../dappservices/multi_index.hpp"

#define DAPPSERVICES_ACTIONS() \
  XSIGNAL_DAPPSERVICE_ACTION \
  IPFS_DAPPSERVICE_ACTIONS

/*** IPFS: (xcommit)(xcleanup)(xwarmup) ***/
#define DAPPSERVICE_ACTIONS_COMMANDS() \
  IPFS_SVC_COMMANDS()

/*** UPDATE CONTRACT NAME ***/
#define CONTRACT_NAME() mycontract

using std::string;

CONTRACT_START()
  public:

  /*** YOUR LOGIC ***/

  private:
    struct [[eosio::table]] vramaccounts {
      asset    balance;
      uint64_t primary_key()const { return balance.symbol.code().raw(); }
    };

    /*** VRAM MULTI_INDEX TABLE ***/
```

(continues on next page)

(continued from previous page)

```
    typedef dapp::multi_index<"vaccounts"_n, vramaccounts> cold_accounts_t;

    /*** FOR CLIENT SIDE QUERY SUPPORT ***/
    typedef eosio::multi_index<".vaccounts"_n, vramaccounts> cold_accounts_t_v_abi;
    TABLE shardbucket {
      std::vector<char> shard_uri;
      uint64_t shard;
      uint64_t primary_key() const { return shard; }
    };
    typedef eosio::multi_index<"vaccounts"_n, shardbucket> cold_accounts_t_abi;

/*** ADD ACTIONS ***/
CONTRACT_END((your)(actions)(here))
```

1.7.6 Compile

See the unit testing section for details on adding unit tests.

```
zeus compile
# compile and test with
zeus test
```

1.7.7 Deploy Contract

```
export DSP_ENDPOINT=https://kylin-dsp-1.liquidapps.io
export KYLIN_TEST_ACCOUNT=<ACCOUNT_NAME>
export KYLIN_TEST_PUBLIC_KEY=<ACTIVE_PUBLIC_KEY>
# Buy RAM:
cleos -u $DSP_ENDPOINT system buyram $KYLIN_TEST_ACCOUNT $KYLIN_TEST_ACCOUNT "200.
↪0000 EOS" -p $KYLIN_TEST_ACCOUNT@active
# Set contract code and abi
cleos -u $DSP_ENDPOINT set contract $KYLIN_TEST_ACCOUNT ../contract -p $KYLIN_TEST_
↪ACCOUNT@active

# Set contract permissions
cleos -u $DSP_ENDPOINT set account permission $KYLIN_TEST_ACCOUNT active "{\
↪"threshold\":1,\"keys\":[{\"weight\":1,\"key\":\"$KYLIN_TEST_PUBLIC_KEY\"}],\
↪"accounts\":[{\"permission\":{\"actor\":\"$KYLIN_TEST_ACCOUNT\",\"permission\":\
↪"eosio.code\"},\"weight\":1}]}" owner -p $KYLIN_TEST_ACCOUNT@active
```

1.7.8 Select and stake DAPP for DSP package

- Use the faucet to get some DAPP tokens on Kylin
- Information on: DSP Packages and staking DAPP/DAPPHDL (AirHODL token)

```
export PROVIDER=uuddlrlrbass
export PACKAGE_ID=package1

# select your package:
export SERVICE=ipfsservice1
```

(continues on next page)

(continued from previous page)

```
cleos -u $DSP_ENDPOINT push action dappservices selectpkg "[\"$KYLIN_TEST_ACCOUNT\",\"
↪$PROVIDER\",\"$SERVICE\",\"$PACKAGE_ID\"]" -p $KYLIN_TEST_ACCOUNT@active

# Stake your DAPP to the DSP that you selected the service package for:
cleos -u $DSP_ENDPOINT push action dappservices stake "[\"$KYLIN_TEST_ACCOUNT\",\"
↪$PROVIDER\",\"$SERVICE\",\"10.0000 DAPP\"]" -p $KYLIN_TEST_ACCOUNT@active
```

1.7.9 Test

Finally you can now test your vRAM implementation by sending an action through your DSP's API endpoint

```
cleos -u $DSP_ENDPOINT push action $KYLIN_TEST_ACCOUNT youraction1 "[\"param1\",\
↪"param2\"]" -p $KYLIN_TEST_ACCOUNT@active

# coldtoken (issue / transfer use vRAM):
cleos -u $DSP_ENDPOINT push action $KYLIN_TEST_ACCOUNT create "[\"$KYLIN_TEST_ACCOUNT\
↪",\"1000000000 TEST\"]" -p $KYLIN_TEST_ACCOUNT
cleos -u $DSP_ENDPOINT push action $KYLIN_TEST_ACCOUNT issue "[\"$KYLIN_TEST_ACCOUNT\
↪",\"1000 TEST\",\"yay vRAM\"]" -p $KYLIN_TEST_ACCOUNT
cleos -u $DSP_ENDPOINT push action $KYLIN_TEST_ACCOUNT transfer "[\"$KYLIN_TEST_
↪ACCOUNT\",\"natdeveloper\",\"1000 TEST\",\"yay vRAM\"]" -p $KYLIN_TEST_ACCOUNT
```

The result should look like:

```
executed transaction:␣
↪865a3779b3623eab94aa2e2672b36dfec9627c2983c379717f5225e43ac2b74a  104 bytes  67049␣
↪us
#  yourcontract <= yourcontract::youraction1         {"param1":"param1","param2":
↪"param2"}
>> {"version":"1.0","etype":"service_request","payer":"yourcontract","service":
↪"ipfsservice1","action":"commit","provider":"","data":"DH......"}
```

1.7.10 Get table row

```
# zeus:
zeus get-table-row "CONTRACT_ACCOUNT" "TABLE_NAME" "SCOPE" "TABLE_PRIMARY_KEY" --
↪endpoint $DSP_ENDPOINT | python -m json.tool
# curl:
curl http://$DSP_ENDPOINT/v1/dsp/ipfsservice1/get_table_row -d '{"contract":"CONTRACT_
↪ACCOUNT","scope":"SCOPE","table":"TABLE_NAME","key":"TABLE_PRIMARY_KEY"}' | python -
↪m json.tool

# coldtoken:
zeus get-table-row $KYLIN_TEST_ACCOUNT "accounts" $KYLIN_TEST_ACCOUNT "TEST" --
↪endpoint $DSP_ENDPOINT | python -m json.tool
curl http://$DSP_ENDPOINT/v1/dsp/ipfsservice1/get_table_row -d '{"contract":"CONTRACT_
↪ACCOUNT","scope":"CONTRACT_ACCOUNT","table":"accounts","key":"TEST"}' | python -m␣
↪json.tool
```

1.7.11 Get table - `get-table.js`

Reads all vRAM tables of a smart contract and stores them with the naming syntax: `${contract_name}-${table_name}-table.json`. The script is located in the `utils/ipfs-service/get-table.js` of an unboxed zeus box.

Mandatory env variables:

```
# account name vRAM table exists on
export CONTRACT_NAME=
# run script
node utils/ipfs-service/get-table
```

Optional env variables (if using non-local nodeos / IPFS instance):

```
# defaults to all vRam tables in the abi, can be used to target a specific table
export TABLE_NAME=
# defaults to localhost:8888, can be used to specify external nodeos instance
export NODEOS_ENDPOINT=
# defaults to localhost, can be used to specify external IPFS instance
export IPFS_HOST=
# defaults to 5001
export IPFS_PORT=
# defaults to http
export IPFS_PROTOCOL=
# defaults to 1024
export SHARD_LIMIT=
# defaults to false
# produces a ${contractName}-${tableName}-roots.json file which is the table's␣
↪current entries
# also produces an ipfs-data.json which can be used to recreate the current state of␣
↪the IPFS table
export VERBOSE=
```

Steps to produce `/ipfs-dapp-service/test1-test-table.json` file below:

```
npm i -g @liquidapps/zeus-cmd
zeus unbox ipfs-dapp-service
cd ipfs-dapp-service
zeus test
export CONTRACT_NAME=test1
node utils/ipfs-service/get-table
```

Expected output `/ipfs-dapp-service/test1-test-table.json`:

```
[
  {
    "scope": "test1",
    "key": "2b02000000000000",
    "data": {
      "id": "555",
      "sometestnumber": "0"
    }
  },
  {
    "scope": "test1",
    "key": "0200000000000000",
    "data": {
```

(continues on next page)

(continued from previous page)

```
      "id": "2",
      "sometestnumber": "0"
    }
  }
  ...
]
```

If `VERBOSE=true`, you will also get `test1-test-roots.json` and `ipfs-data.json`:

`test1-test-roots.json` - equivalent of `cleos get table test1 test1 test`

```
[
  {
    "shard_uri":
↪"01551220d0c889cbd658f2683c78a09a8161ad406dd828dadab383fdcc0659aa6dfed8dc",
    "shard": 3
  },
  {
    "shard_uri":
↪"01551220435f234b3af595737af50ac0b4e44053f0b31d31d94e1ffe917fd3dfbc6a9d88",
    "shard": 156
  },
  ...
]
```

`ipfs-data.json` - produces all data necessary to recreate current state of the table, can be used for populating a DSP's IPFS cluster

```
{
  "015512204cbbd8ca5215b8d161aec181a74b694f4e24b001d5b081dc0030ed797a8973e0":
↪"01000000000000000000000000000000",
  "01551220b422e3b9180b32ba0ec0d538c7af1cf7ccf764bfb89f4cd5bc282175391e02bb":
↪"77cc0000000000007f00000000000000",
  ...
}
```

1.7.12 Get ordered keys - `get-ordered-keys.js`

Prints ordered vRAM table keys in ascending order account/table/scope. This can be used to iterate over the entire table client side. The script is located in the `utils/ipfs-service/get-ordered-keys.js` of an unboxed zeus box.

Mandatory env variables:

```
export CONTRACT_NAME=
export SCOPE=
export TABLE_NAME=
node utils/ipfs-service/get-ordered-keys
```

Optional env variables (if using non-local nodeos / IPFS instance):

```
# defaults to localhost:8888, can be used to specify external nodeos instance
export NODEOS_ENDPOINT=
# defaults to localhost, can be used to specify external IPFS instance
export IPFS_HOST=
```

(continues on next page)

(continued from previous page)

```
# defaults to 5001
export IPFS_PORT=
# defaults to http
export IPFS_PROTOCOL=
# defaults to 1024
export SHARD_LIMIT=
```

Steps to produce console logged output below:

```
npm i -g @liquidapps/zeus-cmd
zeus unbox ipfs-dapp-service
cd ipfs-dapp-service
zeus test
export CONTRACT_NAME=test1
export SCOPE=test1
export TABLE_NAME=test
node utils/ipfs-service/get-ordered-keys
```

Expected output:

```
[ '0', '1', '2', '20', '555', '12345', '52343' ]
```

Querying table rows with Zeus or the `dapp-client`'s `get_vram_row` call:

```
# zeus get-table-row <contract> <table> <scope> <key> <keytype>
zeus get-table-row test1 test test1 52343 number
# output:
{"row":{"id":"0","sometestnumber":"0"}}
```

1.8 LiquidAccounts Getting Started

```
  _       _                    _       _      _                                                  _
 | |     (_)   __ _   _   _  (_)   __| |    / \      ___     ___     ___     _   _   _ __    | |_   ___
 | |     | |  / _` | | | | | | |  / _` |   / _ \    / __|   / __|   / _ \   | | | | | '_ \   | __| / __|
 | |___  | | | (_| | | |_| | | | | (_| |  / ___ \   | (__  | (__  | (_) | | |_| | | | | | | |_  \__ \
 |_____| |_|  \__, |  \__,_| |_|  \__,_| /_/   \_\  \___|  \___|  \___/   \__,_| |_| |_|  \__| |___/
                 |_|
```

LiquidAccounts are EOS accounts that are stored in vRAM instead of RAM. This drastically reduces the cost of creating accounts on EOS. Another great place to understand the service is in the unit tests.

1.8.1 Prerequisites

- Zeus - Zeus installs eos and the eosio.cdt if not already installed
- Kylin Account

1.8.2 Unbox LiquidAccounts DAPP Service box

This box contains the LiquidAccounts smart contract libraries, DSP node logic, unit tests, and everything else needed to get started integrating / testing the DAPP Network LiquidAccounts in your smart contract.

```
# npm install -g @liquidapps/zeus-cmd
zeus unbox vaccounts-dapp-service
cd vaccounts-dapp-service
zeus test
```

1.8.3 LiquidAccount Consumer Example Contract used in unit tests

in `contract/eos/vaccountsconsumer/vaccountsconsumer.cpp` The consumer contract is a great starting point for playing around with the LiquidAccount syntax.

```
/* DELAY REMOVAL OF USER DATA INTO VRAM */
/* ALLOWS FOR QUICKER ACCESS TO USER DATA WITHOUT THE NEED TO WARM DATA UP */
#define VACCOUNTS_DELAYED_CLEANUP 120

/* ADD NECESSARY LIQUIDACCOUNT / VRAM INCLUDES */
#include "../dappservices/vaccounts.hpp"
#include "../dappservices/ipfs.hpp"
#include "../dappservices/multi_index.hpp"

/* ADD LIQUIDACCOUNT / VRAM RELATED ACTIONS */
#define DAPPSERVICES_ACTIONS() \
  XSIGNAL_DAPPSERVICE_ACTION \
  IPFS_DAPPSERVICE_ACTIONS \
  VACCOUNTS_DAPPSERVICE_ACTIONS

#define DAPPSERVICE_ACTIONS_COMMANDS() \
  IPFS_SVC_COMMANDS()VACCOUNTS_SVC_COMMANDS()

#define CONTRACT_NAME() vaccountsconsumer


CONTRACT_START()

  /* THE FOLLOWING STRUCT DEFINES THE PARAMS THAT MUST BE PASSED */
  struct dummy_action_hello {
      name vaccount;
      uint64_t b;
      uint64_t c;

      EOSLIB_SERIALIZE( dummy_action_hello, (vaccount)(b)(c) )
  };

  /* DATA IS PASSED AS PAYLOADS INSTEAD OF INDIVIDUAL PARAMS */
  [[eosio::action]] void hello(dummy_action_hello payload) {
    /* require_vaccount is the equivalent of require_auth for EOS */
    require_vaccount(payload.vaccount);

    print("hello from ");
    print(payload.vaccount);
    print(" ");
    print(payload.b + payload.c);
```

(continues on next page)

(continued from previous page)

```
    print("\n");
  }

  [[eosio::action]] void hello2(dummy_action_hello payload) {
    print("hello2(default action) from ");
    print(payload.vaccount);
    print(" ");
    print(payload.b + payload.c);
    print("\n");
  }

  [[eosio::action]] void init(dummy_action_hello payload) {
  }

  /* EACH ACTION MUST HAVE A STRUCT THAT DEFINES THE PAYLOAD SYNTAX TO BE PASSED */
  VACCOUNTS_APPLY(((dummy_action_hello)(hello))((dummy_action_hello)(hello2)))

CONTRACT_END((init)(hello)(hello2)(regaccount)(xdcommit)(xvinit))
```

1.8.4 Compile

See the unit testing section for details on adding unit tests.

```
zeus compile
# test without compiling
zeus test
# compile and test with
zeus test -c
```

1.8.5 Deploy Contract

```
export DSP_ENDPOINT=https://kylin-dsp-2.liquidapps.io
export KYLIN_TEST_ACCOUNT=<ACCOUNT_NAME>
export KYLIN_TEST_PUBLIC_KEY=<ACTIVE_PUBLIC_KEY>
# Buy RAM:
cleos -u $DSP_ENDPOINT system buyram $KYLIN_TEST_ACCOUNT $KYLIN_TEST_ACCOUNT "200.
↪0000 EOS" -p $KYLIN_TEST_ACCOUNT@active
# Set contract code and abi
cleos -u $DSP_ENDPOINT set contract $KYLIN_TEST_ACCOUNT vaccountsconsumer -p $KYLIN_
↪TEST_ACCOUNT@active

# Set contract permissions
cleos -u $DSP_ENDPOINT set account permission $KYLIN_TEST_ACCOUNT active "{\
↪"threshold\":1,\"keys\":[{\"weight\":1,\"key\":\"$KYLIN_TEST_PUBLIC_KEY\"}],\
↪"accounts\":[{\"permission\":{\"actor\":\"$KYLIN_TEST_ACCOUNT\",\"permission\":\
↪"eosio.code\"},\"weight\":1}]}" owner -p $KYLIN_TEST_ACCOUNT@active
```

1.8.6 Select and stake DAPP for DSP package | DSP Portal Link

- Use the faucet to get some DAPP tokens on Kylin
- Information on: DSP Packages and staking DAPP/DAPPHDL (AirHODL token)

```
export PROVIDER=heliosselene
export PACKAGE_ID=accountless1

# select your package:
export SERVICE=accountless1
cleos -u $DSP_ENDPOINT push action dappservices selectpkg "[\"$KYLIN_TEST_ACCOUNT\",\"
↪$PROVIDER\",\"$SERVICE\",\"$PACKAGE_ID\"]" -p $KYLIN_TEST_ACCOUNT@active

# Stake your DAPP to the DSP that you selected the service package for:
cleos -u $DSP_ENDPOINT push action dappservices stake "[\"$KYLIN_TEST_ACCOUNT\",\"
↪$PROVIDER\",\"$SERVICE\",\"10.0000 DAPP\"]" -p $KYLIN_TEST_ACCOUNT@active
```

1.8.7 Test

First you'll need to initialize the LiquidAccounts implementation with the `chain_id` of the platform you're operating on.

```
# kylin
export CHAIN_ID=5fff1dae8dc8e2fc4d5b23b2c7665c97f9e9d8edf2b6485a86ba311c25639191
cleos -u $DSP_ENDPOINT push action $KYLIN_TEST_ACCOUNT xvinit "[\"$CHAIN_ID\"]" -p
↪$KYLIN_TEST_ACCOUNT
```

Then you can begin registering accounts. You will need to do this either in a nodejs environment using the `dapp-client-lib`, or you can use the `zeus vaccounts push-action`. Here is an example of using the lib to register an account..

All payloads must include a key value pair with `"vaccount":"vaccountname"` or the transaction will fail. This is so the `dapp-client` can fetch the nonce associated with the LiquidAccount.

dapp-client

```
npm install -g @liquidapps/dapp-client
```

This example takes:

- the contract name the LiquidAccount project is deployed to
- the active private key of that account
- the `regaccount` as the action name
- the payload with the `vaccount` name

After registering an account, you may also use the library to push LiquidAccount transactions. In the linked example, you can see that the action name has changed to `hello` and the payload has changed to include the required parameters.

Zeus vaccounts push-action

Push LiquidAccount actions easily with zeus's wrapper of the `dapp-client` library. You can pass a `--dsp-url` for your DAPP Service Provider's API endpoint. Then pass the name of the contract that the LiquidAccount code is deployed to, the action name (`regaccount` for example), and the payload.

You also have the ability to store your private keys with or without encryption. If you choose to encrypt, you can pass the `--encrypted` flag when creating a new account to store the keys. You can provide a password by command

line, or with the flag `--password`. If you use the account again, zeus will look for the key based on what network you are operating on. If it finds it, it will use that key to sign and prompt for a password if needed.

```
zeus vaccounts push-action <CONTRACT> <ACTION> <PAYLOAD> --dsp-url https://kylin-dsp-
↪2.liquidapps.io

# optional flags:

--dsp-url # url to DAPP Service Provider's API endpoint
# default: https://kylin-dsp-2.liquidapps.io
--private-key # LiquidAccount private key, can be provided or auto generated
# will be auto generated and stored in the storage path if not provided
--encrypted # Encrypt the LiquidAccount keys with a password
# default: false
--password # password to encrypt the keys with
--network # network LiquidAccount contract deployed on (other options: kylin, jungle,␣
↪mainnet)
# development (local)
--storage-path # path to the wallet which will store the LiquidAccount key
# default: path.join(require('os').homedir(), '.zeus')

# Example:
zeus vaccounts push-action test1v regaccount '{"vaccount":"vaccount1"}'
zeus vaccounts push-action vacctstst123 regaccount '{"vaccount":"vaccount2"}' --
↪private-key 5KJL... -u https://kylin-dsp-2.liquidapps.io
zeus vaccounts push-action vacctstst123 regaccount '{"vaccount":"vaccount3"}' -u␣
↪http://kylin-dsp-2.liquidapps.io/ --encrypted --network=kylin --password=password
```

1.9 LiquidHarmony Getting Started

```
 _       _                _      _ _     _
| |     (_)              (_)    | | |   | |
| |      _  __ _ _   _ _  __| | |__| | __ _ _ __ _ __ ___   ___  _ __  _   _
| |     | |/ _` | | | | |/ _` |  __  |/ _` | '__| '_ ` _ \ / _ \| '_ \| | | |
| |_____| | (_| | |_| | | (_| | |  | | (_| | |  | | | | | | (_) | | | | |_| |
|_______|_|\__, |\__,_|_|\__,_|_|  |_|\__,_|_|  |_| |_| |_|\___/|_| |_|\__, |
              | |                                                         __/ |
              |_|                                                        |___/
```

LiquidHarmony includes all oracle related logic from the DAPP Network. This includes things like http fetches, IBC and XIBC fetches, vCPU and more. The full list of options can be explored in the oracles directory of the repo: https://github.com/liquidapps-io/zeus-sdk/tree/master/boxes/groups/oracles

The DAPP Network offers the ability to fetch information using DAPP Service Providers (DSPs). A developer may choose as many or as few oracles to use in fetching data points from the internet. If a developer wishes to prevent a scenario where all oracles have an incentive to return false information, the developer may run a DSP themselves and set the threshold of acceptance for the information to all parties. Another great place to understand the service is in the unit tests within each sub directory.

1.9.1 Prerequisites

- Zeus - Zeus installs eos and the eosio.cdt if not already installed
- Kylin Account

The DAPP Network currently supports the following oracle requests:

- HTTP(S) Get
- HTTP(S) Post
- HTTP(S)+JSON Get
- HTTP(S)+JSON Post
- Nodeos History Get
- IBC Block Fetch - (Mainnet, BOS, Telos, Kylin, Worbli, Jungle, Meetone)
- Oracle XIBC - read for foreign chain (Ethereum, Tron, Cardano, Ripple, Bitcoin, Litecoin, Bitcoin Cash)
- Wolfram Alpha
- Random Number
- Stockfish - chess engine AI
- SQL

1.9.2 Unbox Oracle DAPP Service box

This box contains the oracle smart contract libraries, DSP node logic, unit tests, and everything else needed to get started integrating / testing the DAPP Network oracles in your smart contract.

```
# npm install -g @liquidapps/zeus-cmd
zeus unbox oracle-dapp-service
cd oracle-dapp-service
zeus test
```

1.9.3 Creating an Oracle Request URI

Each of the following oracle request types comes equipped with its own syntax that gets encoded with `Buffer.from("<URI HERE>", 'utf8')`. The following guide will explain the syntax to generate your URI. Each URI should be passed through the buffer as plaintext is not accepted.

- HTTP(S) Get & Post: `https://ipfs.io/ipfs/Qmaisz6NMhDB51cCvNWa1GMS7LU1pAxdF4Ld6Ft9kZEP2a` - simply add the full URL path
- HTTP(S)+JSON Get: `https+json://name/api.github.com/users/tmuskal` - prepend your uri with `https+json`, then specify the key mapping path of your desired data point, in the example, the `name` key is used as the requested data point. To request nested values beneath the first layer of keys, simple separate the desired data point with a `.`, e.g., `name.value`. Then add the path to your desired data point: `api.github.com/users/tmuskal`. Note you may use `http+json` or `https+json`.
- HTTP(S)+JSON Post: `https+post+json://timestamp/${body}/nodes.get-scatter.com:443/v1/chain/get_block` - where body is `const body = Buffer.from('{"block_num_or_id":"36568000"}').toString('base64').` In this example you specify the type of request: `https+post+json` then the key mapping `timestamp` then the body of the POST request, encoded in base64, then the URL path `nodes.get-scatter.com:443/v1/chain/get_block`.
- Nodeos History Get: `self_history://${code}/0/0/0/action_trace.act.data.account` - where code is `const code = 'test1';`

- IBC Block Fetch: `sister_chain_block://bos/10000000/transaction_mroot` - the `sister_chain_block` specifies the type of oracle request, followed by the chain of choice `bos` then the requested data point.
- Oracle XIBC: `foreign_chain://ethereum/history/0x100/result.transactionsRoot` - here the `foreign_chain` oracle type is used followed by the foreign chain of choice: `ethereum`, the type of data point (`block_number`, `history`, `balance`, `storage`). To see other blockchain data point options, see this file. Then the required data parameter is passed `0x100` followed by the object key mapping `result.transactionsRoot`.
  - You may also see more examples in the unit test
- Wolfram Alpha: `wolfram_alpha://What is the average air speed velocity of a laden swallow?` - here the `wolfram_alpha` oracle type is used followed by the question: `What is the average air speed velocity of a laden swallow?`.

1.9.4 LiquidHarmony Consumer Example Contract used in unit tests

in `contract/eos/oracleconsumer/oracleconsumer.cpp` The consumer contract is a great starting point for playing around with the LiquidHarmony syntax.

```
/* INCLUDE ORACLE LOGIC */
#include "../dappservices/oracle.hpp"

/* ADD DAPP NETWORK RELATED ORACLE ACTIONS */
#define DAPPSERVICES_ACTIONS() \
  XSIGNAL_DAPPSERVICE_ACTION \
  ORACLE_DAPPSERVICE_ACTIONS

#define DAPPSERVICE_ACTIONS_COMMANDS() \
  ORACLE_SVC_COMMANDS()

#define CONTRACT_NAME() oracleconsumer

CONTRACT_START()

  /*

    tesget - provide a URI using the DAPP Network Oracle syntax and an expected␣
↪result,
    it the result does not match the expected field, the transaction fails

    testrnd - fetch oracle request based on URI without expected field assertion

  */

 [[eosio::action]] void testget(std::vector<char>  uri, std::vector<char>␣
↪expectedfield) {
    /* USE EOSIO'S ASSERTION TO CHECK FOR REQUIRED THREHSHOLD OF ORACLES IS MET */
    eosio::check(getURI(uri, [&]( auto& results ) {
      eosio::check(results.size() > 0, "require multiple results for consensus");
      auto itr = results.begin();
      auto first = itr->result;
      ++itr;
      /* SET CONSENSUS LOGIC FOR RESULTS */
      while(itr != results.end()) {
        eosio::check(itr->result == first, "consensus failed");
```

(continues on next page)

(continued from previous page)

```
        ++itr;
      }
      return first;
    }) == expectedfield, "wrong data");
  }

  [[eosio::action]] void testrnd(std::vector<char> uri) {
    getURI(uri, [&]( auto& results ) {
      return results[0].result;
    });
  }
CONTRACT_END((testget)(testrnd))
```

1.9.5 Compile

See the unit testing section for details on adding unit tests.

```
zeus compile
# test without compiling
zeus test
# compile and test with
zeus test -c
```

1.9.6 Deploy Contract

```
export DSP_ENDPOINT=https://kylin-dsp-2.liquidapps.io
export KYLIN_TEST_ACCOUNT=<ACCOUNT_NAME>
export KYLIN_TEST_PUBLIC_KEY=<ACTIVE_PUBLIC_KEY>
# Buy RAM:
cleos -u $DSP_ENDPOINT system buyram $KYLIN_TEST_ACCOUNT $KYLIN_TEST_ACCOUNT "200.
↪0000 EOS" -p $KYLIN_TEST_ACCOUNT@active
# Set contract code and abi
cleos -u $DSP_ENDPOINT set contract $KYLIN_TEST_ACCOUNT oracleconsumer -p $KYLIN_TEST_
↪ACCOUNT@active

# Set contract permissions
cleos -u $DSP_ENDPOINT set account permission $KYLIN_TEST_ACCOUNT active "{\
↪"threshold\":1,\"keys\":[{\"weight\":1,\"key\":\"$KYLIN_TEST_PUBLIC_KEY\"}],\
↪"accounts\":[{\"permission\":{\"actor\":\"$KYLIN_TEST_ACCOUNT\",\"permission\":\
↪"eosio.code\"},\"weight\":1}]}" owner -p $KYLIN_TEST_ACCOUNT@active
```

1.9.7 Select and stake DAPP for DSP package | DSP Portal Link

- Use the faucet to get some DAPP tokens on Kylin
- Information on: DSP Packages and staking DAPP/DAPPHDL (AirHODL token)

```
export PROVIDER=heliosselene
export PACKAGE_ID=oracleservic

# select your package:
```

(continues on next page)

(continued from previous page)

```
export SERVICE=oracleservic
cleos -u $DSP_ENDPOINT push action dappservices selectpkg "[\"$KYLIN_TEST_ACCOUNT\",\"
↪$PROVIDER\",\"$SERVICE\",\"$PACKAGE_ID\"]" -p $KYLIN_TEST_ACCOUNT@active

# Stake your DAPP to the DSP that you selected the service package for:
cleos -u $DSP_ENDPOINT push action dappservices stake "[\"$KYLIN_TEST_ACCOUNT\",\"
↪$PROVIDER\",\"$SERVICE\",\"10.0000 DAPP\"]" -p $KYLIN_TEST_ACCOUNT@active
```

1.9.8 Test

Finally you can now test your LiquidHarmony implementation by sending an action through your DSP's API endpoint

```
# oracleconsumer contract (testrnd / testget):
# uri: Buffer.from("https://ipfs.io/ipfs/
↪Qmaisz6NMhDB51cCvNWa1GMS7LU1pAxdF4Ld6Ft9kZEP2a", 'utf8')
export␣
↪URI=68747470733a2f2f697066732e696f2f697066732f516d6169737a364e4d68444235316343764e576131474d53374c5
export EXPECTED_FIELD=48656c6c6f2066726f6d20495046532047617465776179204368656564b65720a
cleos -u $DSP_ENDPOINT push action $KYLIN_TEST_ACCOUNT testrnd "[\"$URI\"]" -p $KYLIN_
↪TEST_ACCOUNT
cleos -u $DSP_ENDPOINT push action $KYLIN_TEST_ACCOUNT testget "[\"$URI\",\"$EXPECTED_
↪FIELD\"]" -p $KYLIN_TEST_ACCOUNT
```

1.10 LiquidScheduler Getting Started

```
  _       _                    _       _   _____           _                   _        ␣
↪  _
 | |     (_)    __ _   _   _  (_)    __| | / ____|    ____  | |__     ___     __| |  _   _␣
↪ | |   ____   _ __
 | |     | |  / _` | | | | | | |   / _` | \___ \    / __| | '_ \    / _ \   / _` | | | | |␣
↪| | |  / _ \ | '__|
 | |___  | | | (_| | | |_| | | | | (_| |  ____) | | (__  | | | | |  __/ | (_| | | |_|␣
↪| | | |  __/ | |
 |_____| |_|  \__, |  \__,_| |_|  \__,_| |_____/   \____| |_| |_|  \____|  \__,_|  \__,_
↪| |_|  \____| |_|
                |_|
```

LiquidScheduler is an on chain cron solution for EOS based actions. One use case would be setting up LiquidHarmony (oracle) fetches on a continual basis. Another great place to understand the service is in the unit tests.

1.10.1 Prerequisites

- Zeus - Zeus installs eos and the eosio.cdt if not already installed
- Kylin Account

1.10.2 Unbox LiquidScheduler DAPP Service box

This box contains the LiquidScheduler smart contract libraries, DSP node logic, unit tests, and everything else needed to get started integrating / testing the DAPP Network LiquidScheduler in your smart contract.

```
# npm install -g @liquidapps/zeus-cmd
zeus unbox cron-dapp-service
cd cron-dapp-service
zeus test
```

1.10.3 LiquidScheduler Consumer Example Contract used in unit tests

in `contract/eos/cronconsumer/cronconsumer.cpp` The consumer contract is a great starting point for playing around with the LiquidScheduler syntax.

```
/* IMPORT DAPP NETWORK SERVICE */
#include "../dappservices/cron.hpp"

/* ADD LIQUIDSCHEDULER ACTIONS */
#define DAPPSERVICES_ACTIONS() \
  XSIGNAL_DAPPSERVICE_ACTION \
  CRON_DAPPSERVICE_ACTIONS \

#define DAPPSERVICE_ACTIONS_COMMANDS() \
  CRON_SVC_COMMANDS()

#define CONTRACT_NAME() cronconsumer

CONTRACT_START()
  /* SETUP COUNTER TABLE FOR HOW MANY TIME TO REPEAT CRON */
  TABLE stat {
      uint64_t   counter = 0;
  };

  typedef eosio::singleton<"stat"_n, stat> stats_def;
  /* CONFIGURE LOGIC FOR EACH CRON TASK */
  bool timer_callback(name timer, std::vector<char> payload, uint32_t seconds){

    stats_def statstable(_self, _self.value);
    stat newstats;
    if(!statstable.exists()){
      statstable.set(newstats, _self);
    }
    else{
      newstats = statstable.get();
    }
    newstats.counter++;
    statstable.set(newstats, _self);

    // reschedule
    /* CAN return true TO CREATE INFINITE LOOP */
    return (newstats.counter < 10);
  }
  /*  */
 [[eosio::action]] void testschedule() {
    std::vector<char> payload;
    /* SCHEDULE TIMER WITH 2 BEING SECONDS BETWEEN EACH CRON */
    schedule_timer(_self, payload, 2);
  }
CONTRACT_END((testschedule))
```

1.10.4 Compile

See the unit testing section for details on adding unit tests.

```
zeus compile
# test without compiling
zeus test
# compile and test with
zeus test -c
```

1.10.5 Deploy Contract

```
export DSP_ENDPOINT=https://kylin-dsp-2.liquidapps.io
export KYLIN_TEST_ACCOUNT=<ACCOUNT_NAME>
export KYLIN_TEST_PUBLIC_KEY=<ACTIVE_PUBLIC_KEY>
# Buy RAM:
cleos -u $DSP_ENDPOINT system buyram $KYLIN_TEST_ACCOUNT $KYLIN_TEST_ACCOUNT "200.
↪0000 EOS" -p $KYLIN_TEST_ACCOUNT@active
# Set contract code and abi
cleos -u $DSP_ENDPOINT set contract $KYLIN_TEST_ACCOUNT vaccountsconsumer -p $KYLIN_
↪TEST_ACCOUNT@active

# Set contract permissions
cleos -u $DSP_ENDPOINT set account permission $KYLIN_TEST_ACCOUNT active "{\
↪"threshold\":1,\"keys\":[{\"weight\":1,\"key\":\"$KYLIN_TEST_PUBLIC_KEY\"}],\
↪"accounts\":[{\"permission\":{\"actor\":\"$KYLIN_TEST_ACCOUNT\",\"permission\":\
↪"eosio.code\"},\"weight\":1}]}" owner -p $KYLIN_TEST_ACCOUNT@active
```

1.10.6 Select and stake DAPP for DSP package | DSP Portal Link

- Use the faucet to get some DAPP tokens on Kylin
- Information on: DSP Packages and staking DAPP/DAPPHDL (AirHODL token)

```
export PROVIDER=heliosselene
export PACKAGE_ID=cronservices

# select your package:
export SERVICE=cronservices
cleos -u $DSP_ENDPOINT push action dappservices selectpkg "[\"$KYLIN_TEST_ACCOUNT\",\"
↪$PROVIDER\",\"$SERVICE\",\"$PACKAGE_ID\"]" -p $KYLIN_TEST_ACCOUNT@active

# Stake your DAPP to the DSP that you selected the service package for:
cleos -u $DSP_ENDPOINT push action dappservices stake "[\"$KYLIN_TEST_ACCOUNT\",\"
↪$PROVIDER\",\"$SERVICE\",\"10.0000 DAPP\"]" -p $KYLIN_TEST_ACCOUNT@active
```

1.10.7 Test

To test the sample contract, first check the stat table for the current counter value which should increase after the cron. Then simply send a `testschedule` action to your smart contract's account name.

Check RAM Table Row for Counter

Add your `account_namez` to the following curl to see if the counter increases.

```
curl --request POST \
  --url $DSP_ENDPOINT/v1/chain/get_table_rows \
  --header 'accept: application/json' \
  --header 'content-type: application/json' \
  --data '{"code":"account_name","table":"stat","scope":"account_name"}'
```

testschedule

```
cleos -u $DSP_ENDPOINT push action $KYLIN_TEST_ACCOUNT testschedule "[\"\"]" -p
↪$KYLIN_TEST_ACCOUNT
```

1.11 DAPP Network Macros

The DAPP Network utilizes a series of macros and service actions to perform different logic. Many of the macros use a special syntax to interact with the DAPP Service Providers.

In this portion of the docs we'll have a look at the macros associated with the DAPP Network's core services (beta and above). We'll also explore some of the macros exposed by the `dappservices` contract (core contract to the DAPP Network that handles staking, the DAPP token, and packages). This is intended to be an additional reading piece to the getting started sections.

- *dappservices*
- *vRAM*
- *LiquidAccounts*
- *LiquidHarmony (oracles)*
- *LiquidScheduler (cron)*

1.11.1 dappservices

CONTRACT_START() | code

```
/**
  *  Wraps the eosio::contract class and provides the DAPP service actions needed for␣
↪each service utilized as well as a specified CONTRACT_NAME. Intended to be used␣
↪with CONTRACT_END.
  *
  *  @param CONTRACT_NAME - defines smart contract's name
  *  @param DAPPSERVICES_ACTIONS - specifies DAPP Service actions that must be␣
↪included to perform a service
  *
  *  @return eosio::contract class with DAPP service actions defined under␣
↪DAPPSERVICES_ACTIONS() and CONTRACT_NAME
  *
  *  Example:
  *
```

(continues on next page)

(continued from previous page)

```
  *  @code
  *  #define DAPPSERVICES_ACTIONS() \
  *  XSIGNAL_DAPPSERVICE_ACTION \
  *  ORACLE_DAPPSERVICE_ACTIONS
  *
  *  #define CONTRACT_NAME() oracleconsumer
  *
  *  CONTRACT_START()
  *  @endcode
  */

#define CONTRACT_START() \
CONTRACT CONTRACT_NAME() : public eosio::contract { \
  using contract::contract; \
public: \
DAPPSERVICES_ACTIONS()
```

CONTRACT_END() | code

```
/**
  *  Generates the EOSIO_DISPATCH_SVC list of actions for a smart contract.  Intended␣
↪to be used with CONTRACT_START.
  *
  *  @param CONTRACT_NAME - contract name for eosio smart contract
  *  @param methods - list of actions to be included in the smart contract's ABI
  *
  *  @return EOSIO_DISPATCH_SVC list of actions
  *
  *  Example:
  *
  *  @code
  *  #define CONTRACT_NAME() oracleconsumer
  *
  *  CONTRACT_END((testget)(testrnd))
  *  @endcode
  */

#define CONTRACT_END(methods) \
}; \
EOSIO_DISPATCH_SVC(CONTRACT_NAME(),methods)
```

1.11.2 vRAM

dapp::multi_index | code

```
/**
  *  DAPP Network version of the eosio::multi_index container.  Enables storage of␣
↪information in IPFS (vRAM) when not needed in RAM.  When data is warmed up, it is␣
↪checked against the merkle root stored on-chain to ensure integrity and prevent a␣
↪DAPP Service Provider from needing to be a trusted entity.
  *
  *  @param {name} code - account that owns table
```

(continues on next page)

(continued from previous page)

```
  *  @param {uint64_t} scope - scope identifier within the code hierarchy
  *  @param {uint32_t} [shards=1024] - amount of shards to include for each table
  *  @param {buckets_per_shard} [buckets_per_shard=64] - number of buckets to use per␣
↪shard
  *  @param {bool} [pin_shards=false] - persist shards to RAM
  *  @param {bool} [pin_buckets=false] - persist shard buckets to RAM
  *  @param {uint32_t} [cleanup_delay=0] - time in seconds before data loaded in RAM␣
↪is committed to vRAM (IPFS)
  *
  *  @return advanced_multi_index container
  *
  *  Notes
  *  - by utilizing the cleanup_delay param, data persists to RAM and can be used␣
↪until committed.  One use case for this is a session based application where a user␣
↪does not need their data committed to RAM after each transaction.  The cleanup_
↪delay is reset each time a user uses the data.  If a user is inactive for say 120␣
↪seconds, then their data can be committed.  Utilizing the cleanup_delay also␣
↪prevents the latency associated with warming up data into RAM from vRAM (IPFS).
  *  - by selecting pin_shards = true, the shards will not be evicted from the␣
↪ipfsentry table after the transaction that required the data is run
  *
  *
  *  Example:
  *
  *  @code
  *  TABLE testindex {
  *   uint64_t id;
  *   uint64_t sometestnumber;
  *   uint64_t primary_key()const {return id;}
  *  };
  *
  *  typedef dapp::multi_index<"test"_n, testindex> testindex_t;
  *  typedef eosio::multi_index<".test"_n, testindex> testindex_t_v_abi;
  *  typedef eosio::multi_index<"test"_n, testindex_shardbucket> testindex_t_abi;
  *  @endcode
  */

  TABLE testindex {
    uint64_t id;
    uint64_t sometestnumber;
    uint64_t primary_key()const {return id;}
  };

  typedef dapp::multi_index<"test"_n, testindex> testindex_t;
  typedef eosio::multi_index<".test"_n, testindex> testindex_t_v_abi;
  typedef eosio::multi_index<"test"_n, testindex_shardbucket> testindex_t_abi;

  // get some data
  [[eosio::action]] void testget(uint64_t id) {
    testindex_t testset(_self,_self.value, 1024, 64, false, false, 0);
    auto const& data = testset.get(id,"data not found");
  }

  // add new data row with .emplace
  [[eosio::action]] void testemplace(uint64_t id) {
    testindex_t testset(_self,_self.value, 1024, 64, false, false, 0);
    testset.emplace(_self, [&]( auto& a ){
```

(continues on next page)

(continued from previous page)

```
      a.id = id;
    });
  }

  // modify existing data row with .modify
  [[eosio::action]] void testmodify(uint64_t id, uint64_t new_id) {
    testindex_t testset(_self,_self.value, 1024, 64, false, false, 0);
    auto existing = testset.find(id);
    testset.modify(existing,_self, [&]( auto& a ){
      a.id = new_id;
    });
  }

  // test adding a delayed cleanup
  [[eosio::action]] void testdelay(uint64_t id, uint64_t value, uint32_t delay_sec) {
    testindex_t testset(_self,_self.value, 1024, 64, false, false, delay_sec);
    auto existing = testset.find(id);
    if(existing == testset.end())
      testset.emplace(_self, [&]( auto& a ){
        a.id = id;
        a.sometestnumber = value;
      });
    else
      testset.modify(existing,_self, [&]( auto& a ){
        a.sometestnumber = value;
      });
  }

  // test loading a new manifest file
  // a manifest file is a snapshot of the current state of the table
  // manifests can be used to version the database or to revert back
  [[eosio::action]] void testman(dapp::manifest man) {
    testindex_t testset(_self,_self.value);
    testset.load_manifest(man,"Test");
  }

  // increment revision number, reset shards and buckets_per_shard params and next_
↪available_key in vconfig table
  [[eosio::action]] void testclear() {
    testindex_t testset(_self,_self.value, 1024, 64, false, false, 0);
    testset.clear();
  }
```

1.11.3 LiquidAccounts

payload definition | code

```
/**
  *  LiquidAccounts use a payload syntax in order to pass params.  This payload is␣
↪setup as a struct and uses the EOSLIB_SERIALIZE to create the payload type
  *
  *  @param {name} vaccount - vaccount that owns table
  *
  *  Notes
```

(continues on next page)

(continued from previous page)

```
  *  - primary key for the vaccount payload table must be "vaccount" for client side␣
↪library support
  *
  *  Example:
  *
  *  @code
  *  struct dummy_action_hello {
  *   name vaccount;
  *   uint64_t b;
  *   uint64_t c;
  *
  *   EOSLIB_SERIALIZE( dummy_action_hello, (vaccount)(b)(c) )
  *  };
  *
  *  [[eosio::action]] void hello(dummy_action_hello payload) {
  *   ...
  *  }
  *  @endcode
  */
```

VACCOUNTS_APPLY | code

```
/**
  *  LiquidAccounts use the VACCOUNTS_APPLY macro to map which payload structs are␣
↪associated with which actions.  It also defines the LiquidAccount action.
  *
  *  @param ((payload_struct)(action_name))
  *
  *  Example:
  *
  *  @code
  *  VACCOUNTS_APPLY(((dummy_action_hello)(hello))((dummy_action_hello)(hello2)))
  *  @endcode
  */
```

require_vaccount | code

```
/**
  *  LiquidAccounts use the require_vaccount macro in place of the require_auth macro␣
↪for authenticating a LiquidAccount against the key assigned when calling regaccount
  *
  *  @param {name} - vaccount from payload
  *
  *  Example:
  *
  *  @code
  *  require_vaccount(payload.vaccount);
  *  @endcode
  */

void required_key(const eosio::public_key& pubkey){ \
    eosio::check(_pubkey == pubkey, "wrong public key"); \
} \
```

(continues on next page)

(continued from previous page)

```
void require_vaccount(name vaccount){ \
    auto pkey = handleNonce(vaccount); \
    required_key(pkey); \
} \
```

VACCOUNTS_DELAYED_CLEANUP | code

```
/**
  *  LiquidAccounts use the VACCOUNTS_DELAYED_CLEANUP time in seconds to prevent data␣
↪from being committed to IPFS from RAM.
  *
  *  @param {uint32_t} VACCOUNTS_DELAYED_CLEANUP - time delay in seconds before data␣
↪is removed from RAM and committed to vRAM (IPFS)
  *
  *  Notes
  *  - VACCOUNTS_DELAYED_CLEANUP is intended to allow DAPPs to operate in a session␣
↪based way.  Data persists to RAM for the time specified to avoid the warmup process␣
↪associated with vRAM data.  After the user has become inactive, the data is␣
↪committed.
  *
  *  Example:
  *
  *  @code
  *  #define VACCOUNTS_DELAYED_CLEANUP 120
  *  @endcode
  */
```

1.11.4 LiquidHarmony

geturi | code

```
/**
  *  LiquidHarmony uses the geturi action to perform an oracle request
  *
  *  @param {std::vector<char>} uri - hex conversion of URL syntax to perform an␣
↪oracle request
  *  @param {Lambda&&} combinator - lambda to return the results of an oracle request
  *
  *  Example:
  *
  *  @code
  *  [[eosio::action]] void testrnd(std::vector<char> uri) {
  *    getURI(uri, [&]( auto& results ) {
  *      return results[0].result;
  *    });
  *  }
  *  @endcode
  */

TABLE oracleentry {  \
   uint64_t                     id; \
```

(continues on next page)

(continued from previous page)

```
   std::vector<char>             uri; \
   std::vector<provider_result>     results; \
   checksum256 hash_key() const { return hashData(uri); }  \
   uint64_t primary_key()const { return id; }  \
};  \
typedef eosio::multi_index<"oracleentry"_n, oracleentry, indexed_by<"byhash"_n, const_
↪mem_fun<oracleentry, checksum256, &oracleentry::hash_key>>> oracleentries_t; \

static std::vector<char> getURI(std::vector<char> uri, Lambda&& combinator){  \
    checksum256 trxId = transaction_id(); \
    auto trxIdp = trxId.data(); \
    std::string trxIdStr(trxIdp, trxIdp + trxId.size()); \
    auto pubTime = tapos_block_prefix(); \
    std::string uristr(uri.begin(), uri.end()); \
    auto s = fc::base64_encode(trxIdStr) + "://" + fc::to_string(pubTime) + "://" +␣
↪uristr;\
    std::vector<char> idUri(s.begin(), s.end()); \
    return _getURI(idUri, combinator); \
}\

static std::vector<char> _getURI(std::vector<char> uri, Lambda&& combinator){  \
    auto _self = name(current_receiver()); \
    oracleentries_t entries(_self, _self.value);  \
    auto cidx = entries.get_index<"byhash"_n>(); \
    auto existing = cidx.find(hashData(uri)); \
    if(existing == cidx.end()){  \
        SEND_SVC_REQUEST(geturi, uri); \
    } \
    else {\
        auto results = _extractResults(*existing, combinator);\
        cidx.erase(existing);\
        return results; \
    } \
    return std::vector<char>();\
}  \
```

1.11.5 LiquidScheduler

schedule_timer | code

```
/**
  *  LiquidScheduler uses the schedule_timer macro to schedule a cron action on chain␣
↪by adding a timer to the timerentry singleton
  *
  *  @param {name} timer - account name to scope the timer within
  *  @param {std::vector<char>} payload - payload to be accessed within the timer_
↪callback function in the consumer contract
  *  @param {uint32_t} seconds - seconds to repeat the cron
  *
  *  Example:
  *
  *  @code
  *  [[eosio::action]] void testschedule() {
  *    std::vector<char> payload;
```

(continues on next page)

(continued from previous page)

```
  *    schedule_timer(_self, payload, 2);
  *  }
  *  @endcode
  */

TABLE timerentry { \
    int64_t   set_timestamp = 0; \
    int64_t   fired_timestamp = 0; \
};\
typedef eosio::singleton<"timer"_n, timerentry> timers_def;\

static void schedule_timer(name timer,std::vector<char> payload, uint32_t seconds){  \
    timers_def timers(current_receiver(), timer.value); \
    timerentry newtimer; \
    if(timers.exists()){ \
        newtimer = timers.get(); \
    } \
    newtimer.fired_timestamp = 0;\
    newtimer.set_timestamp = eosio::current_time_point().time_since_epoch().count();\
    timers.set(newtimer, current_receiver()); \
    SEND_SVC_REQUEST(schedule, timer, payload, seconds); \
}  \

SVC_RESP_CRON(schedule)(name timer,std::vector<char> payload, uint32_t seconds,name␣
↪current_provider){ \
    timers_def timers(current_receiver() , timer.value); \
    if(!timers.exists()) \
        return; \
    auto current_timer = timers.get(); \
    if(current_timer.fired_timestamp != 0 || (current_timer.set_timestamp + (seconds␣
↪* 1000000) > eosio::current_time_point().time_since_epoch().count()))\
        return; \
    current_timer.fired_timestamp = eosio::current_time_point().time_since_epoch().
↪count(); \
    timers.set(current_timer, current_receiver()); \
    if(!timer_callback(timer, payload, seconds)) \
        return; \
    schedule_timer(timer, payload, seconds);\
} \
```

remove_timer | code

```
/**
  *  LiquidScheduler uses the remove_timer macro to remove a timer from the␣
↪timerentry singleton
  *
  *  @param {name} timer - account name to scope the timer within
  *  @param {std::vector<char>} payload - payload to be accessed within the timer_
↪callback function in the consumer contract
  *  @param {uint32_t} seconds - seconds to repeat the cron
  *
  *  Example:
  *
  *  @code
  *  [[eosio::action]] void testsremove() {
```

(continues on next page)

(continued from previous page)

```
  *    std::vector<char> payload;
  *    remove_timer(_self, payload, 2);
  *  }
  *  @endcode
  */

static void remove_timer(name timer,std::vector<char> payload, uint32_t seconds){  \
    timers_def timers(current_receiver(), timer.value); \
    if(timers.exists()){ \
        timers.remove(); \
    } \
}  \
```

timer_callback | code

```
/**
  *  LiquidScheduler uses the timer_callback to run the cron logic within the␣
↪consumer contract
  *
  *  @param {name} timer - account name to scope the timer within
  *  @param {std::vector<char>} payload - payload to be accessed within the timer_
↪callback function in the consumer contract
  *  @param {uint32_t} seconds - seconds to repeat the cron
  *
  *  Notes
  *  - can return true to create an infinite loop, false breaks the loop
  *
  *  Example:
  *
  *  @code
  *  bool timer_callback(name timer, std::vector<char> payload, uint32_t seconds){
  *    stats_def statstable(_self, _self.value);
  *    stat newstats;
  *    if(!statstable.exists()){
  *      statstable.set(newstats, _self);
  *    }
  *    else {
  *      newstats = statstable.get();
  *    }
  *    newstats.counter++;
  *    statstable.set(newstats, _self);
  *    return (newstats.counter < 10);
  *  }
  *  @endcode
  */
```

1.12 Unit Testing

Unit testing with Zeus is highly customizable and easy to configure. The following example explains how to use the main helper functions to write your own test.

1.12.1 Customize your own unit tests

in tests/mycontract.spec.js

```
import 'mocha';
require('babel-core/register');
require('babel-polyfill');
const { assert } = require('chai'); // Using Assert style
const { getCreateKeys } = require('../extensions/helpers/key-utils');
const getDefaultArgs = require('../extensions/helpers/getDefaultArgs');
const { getTestContract } = require('../extensions/tools/eos/utils');
const artifacts = require('../extensions/tools/eos/artifacts');
const deployer = require('../extensions/tools/eos/deployer');
const { genAllocateDAPPTokens, readVRAMData } = require('../extensions/tools/eos/dapp-
↪services');

/*** UPDATE CONTRACT CODE ***/
var contractCode = 'mycontract';
var ctrt = artifacts.require(`./${contractCode}/`);
const delay = ms => new Promise(res => setTimeout(res, ms));

describe(`${contractCode} Contract`, () => {
  var testcontract;

  /*** SET CONTRACT NAME(S) ***/
  const code = 'airairairair';
  const code2 = 'airairairai2';
  var testUser = "tt11";
  var account = code;

  /*** CREATE TEST ACCOUNT NAME ***/
  const getTestAccountName = (num) => {
    var fivenum = num.toString(5).split('');
    for (var i = 0; i < fivenum.length; i++) {
      fivenum[i] = String.fromCharCode(fivenum[i].charCodeAt(0) + 1);
    }
    fivenum = fivenum.join('');
    var s = '111111111111' + fivenum;
    var prefix = 'test';
    s = prefix + s.substr(s.length - (12 - prefix.length));
    console.log(s);
    return s;
  };

  before(done => {
    (async () => {
      try {

        /*** DEPLOY CONTRACT ***/
        var deployedContract = await deployer.deploy(ctrt, code);

        /*** DEPLOY ADDITIONAL CONTRACTS ***/
        var deployedContract2 = await deployer.deploy(ctrt, code2);

        /*** ALLOCATE DAPP TOKENS TO YOUR DEPLOYED CONTRACT ***/
        await genAllocateDAPPTokens(deployedContract, 'ipfs');

        /*** RETURNS EOSJS SMART CONTRACT INSTANCE ***/
```

(continues on next page)

(continued from previous page)

```
      testcontract = await getTestContract(code);

      /*** ENDS UNIT TEST SUCCESSFULLY ***/
      done();
    } catch (e) {
      /*** FAILS UNIT TEST AND PROVIDES ERROR ***/
      done(e);
    }
  })();
});

/*** DISPLAY NAME FOR TEST, REPLACE 'coldissue' WITH ANYTHING ***/
it('coldissue', done => {
  (async () => {
    try {

      /*** SETUP VARIABLES ***/
      var symbol = 'AIR';

      /*** DEFAULT failed = false FOR ASSERTION ERROR ***/
      /*** SET failed = true IN TRY/CATCH BLOCK TO FAIL TEST ***/
      var failed = false;

      /*** SETUP CHAIN OF ACTIONS ***/
      await testcontract.create({
        issuer: code2,
        maximum_supply: `1000000000.0000 ${symbol}`
      }, {
        authorization: `${code}@active`,
        broadcast: true,
        sign: true
      });

      /*** CREATE ADDITIONAL KEYS AS NEEDED ***/
      var key = await getCreateKeys(code2);

      var testtoken = testcontract;
      await testtoken.coldissue({
        to: code2,
        quantity: `1000.0000 ${symbol}`,
        memo: ''
      }, {
        authorization: `${code2}@active`,
        broadcast: true,
        keyProvider: [key.active.privateKey],
        sign: true
      });

      /*** ADD DELAY BETWEEN ACTIONS ***/
      await delay(3000);

      /*** EXAMPLE TRY/CATCH failed = true ***/
      try {
        await testtoken.transfer({
          from: code2,
          to: code,
          quantity: `100.0000 ${symbol}`,
```

(continues on next page)

(continued from previous page)

```
            memo: ''
          }, {
            authorization: `${code2}@active`,
            broadcast: true,
            keyProvider: [key.active.privateKey],
            sign: true
          });
        } catch (e) {
          failed = true;
        }

        /*** ADD CUSTOM FAILURE MESSAGE ***/
        assert(failed, 'should have failed before withdraw');

        /*** ADDITIONAL ACTIONS ... ***/

        done();
      } catch (e) {
        done(e);
      }
    })();
  });

  /*** USE it.skip TO CONTINUE WITH UNIT TEST IF TEST FAILS ***/
  it.skip('it.skip does not assert and continues test if fails' ...
});
```

1.12.2 Helper Functions

getCreateKeys | Code

The `getCreateKeys` function is intended to create a new key pair in `~/.zeus/networks/development/accounts` if a key pair does not already exist for the account provided. The sub directory to `~/.zeus/network` can be any chain that you are developing on as well, e.g., `kylin`, `jungle`, `mainnet`. If an account name has a contract deployed to it with the `deploy` function, then a key pair will automatically be assigned during the deployment.

```
/**
 * @param account - account name to generate or fetch keys for
 */

const { getCreateKeys } = require('../extensions/helpers/key-utils');
var keys = await getCreateKeys(account);
```

artifacts | Code

The `artifacts` helper pulls the relevant contract files, such as the wasm / ABI, to be used within the unit test.

```
/**
 * @param f - contract name within the /contracts/eos directory
 */

const artifacts = require('../extensions/tools/eos/artifacts');
```

(continues on next page)

(continued from previous page)

```
var contractCode = 'mycontract';
var ctrt = artifacts.require(`./${contractCode}/`);
```

deployer | Code

The `deployer` function deploys a contract to an account based on the contract files and the account name passed. You may also pass your own args, if not specified, the `getDefaultArgs()` function will be passed.

```
/**
 * @param contract - contract file name to deploy
 * @param account - account name to deploy contract to
 * @param [args=getDefaultArgs()] - arguments to be used for configuring the network
↪'s settings
 */

const deployer = require('../extensions/tools/eos/deployer');
var ctrt = artifacts.require(`./${contractCode}/`);
const code = 'airairairair';
var deployedContract = await deployer.deploy(ctrt, code);
```

genAllocateDAPPTokens | Code

The `genAllocateDAPPTokens` function allocates DAPP tokens to the specified contract provided. It also issues, selects a package, stakes, and updates auth to include `eosio.code`.

```
/**
 * @param deployedContract - deployed contract
 * @param serviceName - DAPP Services name as determined in the groups/boxes/groups/
↪services/SELECTED_SERVICE/models/dapp-services/SELECTED_SERVICE.json model file␣
↪under the "name" key
 * @param [provider=''] - DAPP Services Provider name, if non provided, 'pprovider1',
↪'pprovider2' will be used
 * @param [selectedPackage='default'] - package name to select, defaults to "default"
 */

const { genAllocateDAPPTokens } = require('../extensions/tools/eos/dapp-services');
await genAllocateDAPPTokens(deployedContract, 'ipfs');
```

readVRAMData | Code

The `readVRAMData` function makes a vRAM get table row call by providing the `contract`, `key`, `table`, and `scope` arguments.

```
/**
 * @param contract - account name contract was deployed to
 * @param key - primary key of the dapp::multi_index container
 * @param table - table name as specified in the ABI
 * @param scope - scope of dapp::multi_index container to read from
 */

const { readVRAMData } = require('../extensions/tools/eos/dapp-services');
var tableRes = await readVRAMData({
```

(continues on next page)

(continued from previous page)

```
    contract: 'airairairair',
    key: `AIRU`,
    table: "accounts",
    scope: 'tt11'
});
```

getTestContract | Code

The `getTestContract` function creates an EOSJS instance to be used for sending EOS transactions.

```
/**
 * @param code - account name to use in setting up
 */

const code = 'airairairair';
const { getTestContract } = require('../extensions/tools/eos/utils');
testcontract = await getTestContract(code);
await testcontract.create({
    issuer: code,
    maximum_supply: `1000000000.0000 ${symbol}`
}, {
    authorization: `${code}@active`,
    broadcast: true,
    sign: true
});
```

1.12.3 Compile and test

```
zeus test
```

1.13 Packages and Staking

1.13.1 List of available Packages

DSPs who have registered their service packages may be found in the package table under the dappservices account on every supported chain.

DSP Portals for viewing/interacting with packages:

- DSP HQ
- Bloks.io
- EOS Nation
- Malta Block
- Mission Control
- Aloha EOS
- MinerGate

1.13.2 DSP Package Explanation

DSP packages have several fields which are important to understand:

- **api_endpoint** - endpoint to direct DSP related trxs/api requests to
- **package_id** - ID of the package that can be selected with the **selectpkg** action
- **service** - the DSP service to be used. Currently LiquidApps supports 6 DSP services; however DSPs are encouraged to create services of their own as well as create bundled DSP services. The use of these resources is measured in QUOTA.

1. ipfsservice1 - providing IPFS services to the dapp::multi_index container of a smart contract
2. cronservices - enable CRON related tasks such as continuous staking
3. oracleservic - provide oracle related services
4. readfndspsvc - return a result from a smart contract function based on current conditions without sending an EOSIO trx
5. accountless1 - virtual accounts that do not require RAM storage for account related data, instead data and accounts are stored in vRAM

- **provider** - DSP account name
- **quota** - QUOTA represents the amount of actions that a DSP supports based on the package_period. You can think of QUOTA like cell phone minutes in a plan. For a cell phone plan you could pay $10 per month and get 1000 minutes. 1 QUOTA always equals 10,000 actions. Said differently .0001 QUOTA equals 1 action. Instead of $10 per month perhaps you would be required to stake 10 DAPP and/or 10 DAPPHDL (Air-HODL) tokens for a day to receive .001 QUOTA or 10 actions.
- **package_period** - period of the package before restaking is required. Upon restaking the QUOTA and package period are reset.
- **min_stake_quantity** - the minimum quantity of DAPP and/or DAPPHDL (Air-HODL) tokens to stake to receive the designated amount of QUOTA for the specified package_period
- **min_unstake_period** - period of time required to pass before **refund** action can be called after **unstake** command is executed
- **enabled** - bool if the package is available or not

1.13.3 Select a DSP Package

Select a service package from the DSP of your choice.

```
export PROVIDER=someprovider
export PACKAGE_ID=providerpackage
export MY_ACCOUNT=myaccount

# select your package:
export SERVICE=ipfsservice1
cleos -u $DSP_ENDPOINT push action dappservices selectpkg "[\"$MY_ACCOUNT\",\"
↪$PROVIDER\",\"$SERVICE\",\"$PACKAGE_ID\"]" -p $MY_ACCOUNT@active
```

1.13.4 Stake DAPP Tokens for DSP Package

```
# Stake your DAPP to the DSP that you selected the service package for:
cleos -u $DSP_ENDPOINT push action dappservices stake "[\"$MY_ACCOUNT\",\"$PROVIDER\",
↪\"$SERVICE\",\"50.0000 DAPP\"]" -p $MY_ACCOUNT@active
```

1.13.5 Stake DAPPHDL (AirHODL) Tokens for DSP Package

If you were a holder of the EOS token on February 26th, 2019 then you should have a balance of DAPPHDL tokens. These tokens possess the ability to 3rd party stake and unstake tokens throughout the duration of the AirHODL, until February 26th 2021.

```
# Stake your DAPPHDL to the DSP that you selected the service package for:
cleos -u $DSP_ENDPOINT push action dappairhodl1 stake "[\"$MY_ACCOUNT\",\"$PROVIDER\",
↪\"$SERVICE\",\"50.0000 DAPPHDL\"]" -p $MY_ACCOUNT@active
```

1.13.6 Unstake DAPP Tokens

The amount of time that must pass before an unstake executes a refund action and returns DAPP or DAPPHDL tokens is either the current time + the minimum unstake time as stated in the package table, or the end of the current package period, whichever is greater.

```
cleos -u $DSP_ENDPOINT push action dappservices unstake "[\"$MY_ACCOUNT\",\"$PROVIDER\
↪\",\"$SERVICE\",\"50.0000 DAPP\"]" -p $MY_ACCOUNT@active
```

1.13.7 Unstake DAPPHDL (AirHODL) Tokens

```
cleos -u $DSP_ENDPOINT push action dappairhodl1 unstake "[\"$MY_ACCOUNT\",\"$PROVIDER\
↪\",\"$SERVICE\",\"50.0000 DAPPHDL\"]" -p $MY_ACCOUNT@active
# In case unstake deferred trx fails, you can manually refund the unstaked tokens:
cleos -u $DSP_ENDPOINT push action dappairhodl1 refund "[\"$MY_ACCOUNT\",\"$PROVIDER\
↪\",\"$SERVICE\"]" -p $MY_ACCOUNT@active
```

1.13.8 Withdraw DAPPHDL (AirHODL) Tokens

Please note: withdrawing your DAPPHDL tokens immediately makes you ineligible for further vesting and forfeits all your unvested tokens. **This action is irrevocable.** Vesting ends February 26th 2021. Also, you must hold DAPP token before executing this action. If you do not, use the *open* action below to open a 0 balance.

```
# Withdraw
cleos -u $DSP_ENDPOINT push action dappairhodl1 withdraw "[\"$MY_ACCOUNT\"]" -p $MY_
↪ACCOUNT@active
# Open DAPP balance to withdraw if needed
cleos -u $DSP_ENDPOINT push action dappservices open "[\"$MY_ACCOUNT\",\"4,DAPP\",\"
↪$MY_ACCOUNT\"]" -p $MY_ACCOUNT@active
```

1.13.9 Check DAPPHDL (AirHODL) Token Balance & Refresh Data

In the dappairhodl1 contract under the accounts table, enter your account as the scope to retrieve its information.

```
# Refresh accounts table data
cleos -u $DSP_ENDPOINT push action dappservices refresh "[\"$MY_ACCOUNT\"]" -p $MY_
↪ACCOUNT@active
```

1.13.10 Third Party Staking

Third parties may stake to a DSP package on behalf of a DAPP by calling the `staketo` and `unstaketo` actions. In order for a DAPP to select a new package after third parties have staked to them, all third party stakes must be removed.

The following two actions remove third party stakes, `preselectpkg` allows users to be removed in batches by supplying a `depth` parameter to indicate how many accounts to remove at a time. The second action is `retirestakes` which allows for the removal of specific third party stakes from a list of `delegators` account names.

Third party stakers can be identified by supplying the `ID` of the `accountext` table entry that matches the account, provider, service, and package (or pending package) selected by the account being staked to as the scope of the `stakingext` table, for example: `150`.

Examples:

```
cleos -u $DSP_ENDPOINT push action dappservices retirestake "{\"owner\":\"$MY_ACCOUNT\
↪",\"provider\":\"heliosselene\",\"service\":\"cronservices\",\"package\":\
↪"cronservices\",\"delegators\":["dappservice2","natdeveloper","oracletest22"]}" -p
↪$MY_ACCOUNT

cleos -u $DSP_ENDPOINT push action dappservices preselectpkg "{\"owner\":\"$MY_
↪ACCOUNT\",\"provider\":\"heliosselene\",\"service\":\"cronservices\",\"package\":\
↪"cronservices\",\"depth\":\"10\"}" -p $MY_ACCOUNT

cleos -u $DSP_ENDPOINT push action dappservices staketo "{\"from\":\"$MY_ACCOUNT\",\
↪"to\":\"asdfasdfasdy\",\"provider\":\"heliosselene\",\"service\":\"cronservices\",\
↪"quantity\":\"10.0000 DAPP\"}" -p $MY_ACCOUNT

cleos -u $DSP_ENDPOINT push action dappservices unstaketo "{\"from\":\"$MY_ACCOUNT\",\
↪"to\":\"asdfasdfasdy\",\"provider\":\"heliosselene\",\"service\":\"cronservices\",\
↪"quantity\":\"10.0000 DAPP\"}" -p $MY_ACCOUNT

// if deferred trx for refund fails after unstaketo
cleos -u $DSP_ENDPOINT push action dappservices refundto "{\"from\":\"$MY_ACCOUNT\",\
↪"to\":\"asdfasdfasdy\",\"provider\":\"heliosselene\",\"service\":\"cronservices\",\
↪"symcode\":\"DAPP\"}" -p $MY_ACCOUNT
```

More on `preselectpkg` and `retirestakes`

`preselectpkg` allows the DAPP to simply supply a `depth` (number of third party stakes) to remove per tx ordered by the `ID` of "payer". The DAPP may simply execute this as many times as required until no more third party stakes remain. If a depth that is too deep is selected (for example, 100) the trx will simply fail, and a smaller depth can be selected. This method requires no external knowledge of who has staked to the account.

`retirestakes` is a more explicit and selective method for a DAPP. A list of third party stake payers (`name[] delegators`) can be provided. If the list is too large, the tx may time out. In order to discover who has staked to the DAPP's package the following steps can be utilized:

1. Determine the `id` of their package's `accountext` table entry. accountext entries all use the scope `DAPP`. The correct entry is the entry that has that same account, service, provider, and package (or pending_package) as the selected package. bloks.io accountext table
2. Find the `stakingext` entries, using the `id` from the `accountext` table as the scope. bloks.io stakingext table
3. Each entry under this scope will have payers equal to the account names of the third parties. These payers can be used to populate the delegators for the `retirestakes` action.

For both actions, if the `depth` is deeper than the number of stakes, or the `delegators` list includes payers that haven't actually staked, it will succeed gracefully, ignoring the erroneous entries and executing for the correct ones.

The `dappairhodl1` contract is the exception to all of this. It does not get added to the `stakingext` table since while it is a third party stake mechanically, it is essentially the same as a first party stake. We do not support third party staking using AirHODL'd DAPP.

1.14 Zeus Boxes

1.14.1 Browse Boxes:

- regression-tests
- helloworld
- coldtoken
- airhodl
- airdrop
- bancor-extensions
- cardgame
- dapp-services-deploy
- templates-emptycontract-eos-cpp
- all-dapp-services
- sample-zeus-extension
- deepfreeze
- vgrab
- dapp
- game
- ide
- dgoods
- eoscraft
- search

CHAPTER 2

DSPs

2.1 Getting started

2.1.1 Overview

Overview

Architecture

2.1.2 Prerequisites

Account

2.1.3 Deploy

EOSIO Node

IPFS Node

PostgreSQL Database

DSP Service Node

2.1.4 Configuration

Packages

Testing

2.1.5 Claiming Rewards

Claim

2.1.6 Upgrade Version

Upgrade

2.1.7 Replay Contract and Cleanup IPFS Entries

Replay Contract

Cleanup IPFS Entries

2.1.8 Consumer Resource Usage

Consumer Permissions

2.2 Overview

2.2.1 Articles

- vRAM guide for experts
- EOS dApps and Their RAM Expenses

2.2.2 Videos

- Developer Explains - Decentralized Dapp Scaling w/ IPFS! How LiquidApps Dapp Service Providers Work
- EOS Weekly - The LiquidApps Game-Changer
- EOS Weekly - Unlimited DSP Possibilities

2.2.3 Have questions?

- Join our Dev Telegram channel
- Join our Telegram channel
- Email us: support@liquidapps.io

2.2.4 Want more information?

- Read our whitepaper and subscribe to our Medium posts.

2.3 Architecture

A DSP consists of an EOS state history node (non block producing node without a full history setup), an IPFS cluster, a PostgreSQL Database, and a DSP API endpoint. All 4 of these operations may be run on the same machine or separately. All necessary settings may be found in the `config.toml` file.

- EOS state history node - to run a DSP requires running a non block producing EOS node. A full history node is also not required. The EOS node is configured with a backend storage mechanism: `state_history_plugin`.
- IPFS Cluster node - the IPFS cluster stores all vRAM information so that it may be loaded into RAM as a caching mechanism. The InterPlanetary File System is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices. The documentation shows how to setup a local IPFS cluster, but you may also configure an external IPFS cluster in the `config.toml` file.
- PostgreSQL Database - a PostgreSQL database is utilized to prevent duplicate transactions by creating, updating, and retrieving transaction data.
- DSP API - the DSP API is responsible for performing all DAPP service logic such as a vRAM `get_table_row` call, parsing and sending a LiquidAccount transaction, servicing an oracle call, etc.

2.4 Demux Backend

In the `config.toml` file in the DSP Node Setup you can configure the `state_history_plugin`.

You can also configure the `HEAD_BLOCK` to sync demux from. If you experience any issues with demux syncing from an older block, you may try syncing demux at the head block by finding it at bloks.io and setting it to `head_block`.

```
[demux]
backend = "state_history_plugin"

# head block to sync demux from
head_block = 35000000
```

2.5 Account

2.5.1 Prerequisites

Install cleos from: https://github.com/EOSIO/eos/releases

2.5.2 Create Account

Mainnet

```
cleos create key --to-console > keys.txt
export DSP_PRIVATE_KEY=`cat keys.txt | head -n 1 | cut -d ":" -f 2 | xargs echo`
export DSP_PUBLIC_KEY=`cat keys.txt | tail -n 1 | cut -d ":" -f 2 | xargs echo`
```

Save keys.txt somewhere safe!

Have an exising EOS Account

- Getting started on eos mainnet

First EOS Account

Fiat:

- EOS Account Creator
- EOS Lynx
- Scatter

Bitcoin/ETH/Bitcoin Cash/ALFAcoins:

- ZEOS

Kylin

Create an account

```
# Create a new available account name (replace 'yourdspaccount' with your account␣
↪name):
export DSP_ACCOUNT=yourdspaccount
curl http://faucet-kylin.blockzone.net//create/$DSP_ACCOUNT > keys.json
curl http://faucet-kylin.blockzone.net//get_token/$DSP_ACCOUNT
export DSP_PRIVATE_KEY=`cat keys.json | jq -e '.keys.active_key.private'`
export DSP_PUBLIC_KEY=`cat keys.json | jq -e '.keys.active_key.public'`
```

Save keys.json somewhere safe!

2.5.3 Account Name

```
# Create wallet
cleos wallet create --file wallet_password.pwd
```

Save wallet_password.pwd somewhere safe!

2.5.4 Import account

```
cleos wallet import --private-key $DSP_PRIVATE_KEY
```

2.6 EOSIO Node

A non block / non full history node is required for the DSP API to interact with. This node may be hosted with the rest of the DSP architecture or standalone.

2.6.1 Hardware Requirements

2.6.2 Prerequisites

- jq
- wget
- curl

2.6.3 Get EOSIO binary

```
# install 1.8 even if chain is sub 1.7.*
VERSION=1.8.8
```

Ubuntu 18.04

```
FILENAME=eosio_$VERSION-1-ubuntu-18.04_amd64.deb
INSTALL_TOOL=apt
```

Ubuntu 16.04

```
FILENAME=eosio_$VERSION-1-ubuntu-16.04_amd64.deb
INSTALL_TOOL=apt
```

Fedora

```
FILENAME=eosio_$VERSION-1.fc27.x86_64.rpm
INSTALL_TOOL=yum
```

Centos

```
FILENAME=eosio_$VERSION-1.el7.x86_64.rpm
INSTALL_TOOL=yum
```

2.6.4 Install

```
wget https://github.com/EOSIO/eos/releases/download/v$VERSION/$FILENAME
sudo $INSTALL_TOOL install ./$FILENAME
```

2.6.5 Prepare Directories

```
#cleanup
rm -rf $HOME/.local/share/eosio/nodeos || true

#create dirs
mkdir $HOME/.local/share/eosio/nodeos/data/blocks -p
mkdir $HOME/.local/share/eosio/nodeos/data/snapshots -p
mkdir $HOME/.local/share/eosio/nodeos/config -p
```

Snapshots

If you would like an up to date snapshot, please visit: snapshots.eosnation.io and find the latest snapshot for the chain you are using. You will want to unpack the file and store it here with the following file name: `$HOME/.local/share/eosio/nodeos/data/snapshots/boot.bin`. EOS Node tools many also be used for mainnet: https://eosnode.tools/snapshots

Kylin

```
URL="http://storage.googleapis.com/eos-kylin-snapshot/snapshot-2019-06-10-09(utc)-
↪0312d3b9843e2efa6831806962d6c219d37200e0b897a0d9243bcab40b2b546b.bin"
P2P_FILE=https://raw.githubusercontent.com/cryptokylin/CryptoKylin-Testnet/master/
↪fullnode/config/config.ini
GENESIS=https://raw.githubusercontent.com/cryptokylin/CryptoKylin-Testnet/master/
↪genesis.json
CHAIN_STATE_SIZE=256000
wget $URL -O $HOME/.local/share/eosio/nodeos/data/snapshots/boot.bin
```

Jungle

You can find more Jungle peers here: https://monitor.jungletestnet.io/#p2p

```
export MONTH=01
export DAY=09
wget https://eosn.sfo2.digitaloceanspaces.com/snapshots/snapshot-2020-$MONTH-$DAY-15-
↪jungle.bin.bz2
bzip2 -d ./snapshot-2020-01-09-15-jungle.bin.bz2
mv snapshot-2020-01-09-15-jungle.bin $HOME/.local/share/eosio/nodeos/data/snapshots/
↪boot.bin
P2P_FILE=https://validate.eosnation.io/jungle/reports/config.txt
GENESIS=https://raw.githubusercontent.com/EOS-Jungle-Testnet/Node-Manual-Installation/
↪master/genesis.json
CHAIN_STATE_SIZE=256000
```

Mainnet

```
URL="https://s3.eu-central-1.wasabisys.com/eosnodetools/snapshots/snap_2019-12-15-13-
↪00.tar.gz"
P2P_FILE=https://eosnodes.privex.io/?config=1
GENESIS=https://raw.githubusercontent.com/CryptoLions/EOS-MainNet/master/genesis.json
CHAIN_STATE_SIZE=16384
cd $HOME/.local/share/eosio/nodeos/data
wget $URL -O - | tar xvz
```

(continues on next page)

(continued from previous page)

```
SNAPFILE=`ls snapshots/*.bin | head -n 1 | xargs -n 1 basename`
mv snapshots/$SNAPFILE snapshots/boot.bin
```

2.6.6 Configuration

```
cd $HOME/.local/share/eosio/nodeos/config

# download genesis
wget $GENESIS
# config
cat <<EOF >> $HOME/.local/share/eosio/nodeos/config/config.ini
agent-name = "DSP"
http-server-address = 0.0.0.0:8888
p2p-listen-endpoint = 0.0.0.0:9876
blocks-dir = "blocks"
abi-serializer-max-time-ms = 3000
max-transaction-time = 150000
wasm-runtime = wabt
reversible-blocks-db-size-mb = 1024
contracts-console = true
p2p-max-nodes-per-host = 1
allowed-connection = any
max-clients = 100
sync-fetch-span = 500
connection-cleanup-period = 30
http-validate-host = false
access-control-allow-origin = *
access-control-allow-headers = *
access-control-allow-credentials = false
verbose-http-errors = true
http-threads=8
net-threads=8
trace-history-debug-mode = true
trace-history = true
plugin = eosio::producer_plugin
plugin = eosio::chain_plugin
plugin = eosio::chain_api_plugin
plugin = eosio::net_plugin
plugin = eosio::state_history_plugin
state-history-endpoint = 0.0.0.0:8887
chain-state-db-size-mb = $CHAIN_STATE_SIZE
EOF

curl $P2P_FILE > p2p-config.ini
cat p2p-config.ini | grep "p2p-peer-address" >> $HOME/.local/share/eosio/nodeos/
↪config/config.ini
```

Please note the following about some `config.ini` settings:

- `wasm-runtime = wabt` must be used as the `wavm` engine still has bugs
- `read-mode = head` (default is: `read-more = speculative` and does not need to be specified in the `config.ini`) must not be used to prevent duplicate `xwarmup` actions | read more about read modes here

2.6.7 Run

First run (from snapshot)

```
nodeos --disable-replay-opts --snapshot $HOME/.local/share/eosio/nodeos/data/
↪snapshots/boot.bin --delete-all-blocks
```

You will know that the node is fully synced once you see blocks being produced every half second at the head block. You can match the block number you are seeing in the nodeos logs to what bloks.io is indicating as the head block on the chain you are syncing (mainnet, Kylin etc). Once you have confirmed that it is synced press `CTRL+C` once, wait for the node to shutdown and proceed to the next step.

2.6.8 systemd

```
export NODEOS_EXEC=`which nodeos`
export NODEOS_USER=$USER
sudo -E su - -p
cat <<EOF > /lib/systemd/system/nodeos.service
[Unit]
Description=nodeos
After=network.target
[Service]
User=$NODEOS_USER
ExecStart=$NODEOS_EXEC --disable-replay-opts
[Install]
WantedBy=multi-user.target
EOF

systemctl start nodeos
systemctl enable nodeos
exit
sleep 3
systemctl status nodeos
```

2.6.9 Optimizations

- atticlab - cpu performance presentation

2.7 IPFS

The InterPlanetary File System is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices. The DSPs utilize this as the storage layer to request and serve information to and from vRAM <> RAM as a caching solution.

2.7.1 Standalone

go-ipfs

Hardware Requirements

Prerequisites

- golang
- systemd

Ubuntu/Debian

```
sudo apt-get update
sudo apt-get install golang-go -y
```

Centos/Fedora/AWS Linux v2

```
sudo yum install golang -y
```

Install

```
sudo su -
VERS=0.4.22
DIST="go-ipfs_v${VERS}_linux-amd64.tar.gz"
wget https://dist.ipfs.io/go-ipfs/v$VERS/$DIST
tar xvfz $DIST
rm *.gz
mv go-ipfs/ipfs /usr/local/bin/ipfs
exit
```

Configure systemd

```
sudo su -
ipfs init
ipfs config Addresses.API /ip4/0.0.0.0/tcp/5001
ipfs config Addresses.Gateway /ip4/0.0.0.0/tcp/8080
cat <<EOF > /lib/systemd/system/ipfs.service
[Unit]
Description=IPFS daemon
After=network.target
[Service]
ExecStart=/usr/local/bin/ipfs daemon
Restart=always
[Install]
WantedBy=multi-user.target
EOF

systemctl start ipfs
systemctl enable ipfs

exit
```

Adding Peers

To connect with your peers you may open port 4001 to the selected IPs that you wish to communicate with or open the port to all addresses.

Bootstrap Peers | Documentation

Bootstrap peers default to IPFS nodes provided by the core development team. They are scattered across the world. These peers are what your IPFS node will initially monitor upon startup. You may add our mainnet and kylin testnet IPFS nodes with the following commands:

```
# kylin

ipfs bootstrap add /ip4/18.212.96.94/tcp/4001/ipfs/
↪QmZ5gLTZwvfD5DkbbaFFX4YJCi7f4C5oQAgq8qpjL8S1ur
ipfs bootstrap add /ip4/18.212.76.109/tcp/4001/ipfs/
↪QmcCX4b3EF3eXaDe5dgxTL9mXbyci4FwcJAjWqpub5vCXM

# mainnet

ipfs bootstrap add /ip4/35.170.64.183/tcp/4001/ipfs/
↪QmZpyMnBJKwyPJNBUYVuCZEJuKQBEwM6qVHsSp179B3yao
```

Swarm Peers | Documentation

Swarm peers are addresses that the local daemon will listen on for connections from other IPFS peers. They are what your IPFS node will look to first when requesting a file that is not cached locally. Both your node as well as the node you are trying to connect to must run the following commands:

```
# kylin

ipfs swarm connect /ip4/18.212.96.94/tcp/4001/ipfs/
↪QmZ5gLTZwvfD5DkbbaFFX4YJCi7f4C5oQAgq8qpjL8S1ur
ipfs swarm connect /ip4/18.212.76.109/tcp/4001/ipfs/
↪QmcCX4b3EF3eXaDe5dgxTL9mXbyci4FwcJAjWqpub5vCXM

# mainnet

ipfs swarm connect /ip4/35.170.64.183/tcp/4001/ipfs/
↪QmZpyMnBJKwyPJNBUYVuCZEJuKQBEwM6qVHsSp179B3yao
```

Reconnecting Periodically | Medium Article

Peers have a tendency to disconnect from each other if not reconnected manually periodically, so to combat this, you may add the following two files to periodically reconnect to your swarm peers.

```
sudo su -
cat <<EOF > /lib/systemd/system/gateway-connector.service
[Unit]
Description=Job that periodically connects this IPFS node to the gateway node
[Service]
ExecStart=/usr/local/bin/ipfs swarm connect <ADD_MULTIPLE_CONNECTIONS_HERE_SPACE_
↪SEPARATED> # /ip4/18.212.96.94/tcp/4001/ipfs/
↪QmZ5gLTZwvfD5DkbbaFFX4YJCi7f4C5oQAgq8qpjL8S1ur /ip4/18.212.76.109/tcp/4001/ipfs/
↪QmcCX4b3EF3eXaDe5dgxTL9mXbyci4FwcJAjWqpub5vCXM /ip4/35.170.64.183/tcp/4001/ipfs/
↪QmZpyMnBJKwyPJNBUYVuCZEJuKQBEwM6qVHsSp179B3yao
```

(continues on next page)

(continued from previous page)

```
Environment="IPFS_PATH=/root/.ipfs"
EOF
exit
```

```
sudo su -
cat <<EOF > /lib/systemd/system/gateway-connector.timer
[Unit]
Description=Timer that periodically triggers gateway-connector.service
[Timer]
OnBootSec=3min
OnUnitActiveSec=1min
[Install]
WantedBy=timers.target
EOF
exit
```

Now you can enable and start the service:

```
sudo systemctl enable gateway-connector.timer
sudo systemctl start gateway-connector.timer
```

To double checked this worked, run:

```
systemctl list-timers
```

You should see an entry for your gateway connector service. You can also check the status of its last execution attempt by running:

```
systemctl status gateway-connector
```

Finally you can monitor the process with:

```
journalctl -f | grep ipfs
```

Running a private network | Documentation

Running a private IPFS network is possible by removing all default IPFS bootstrap peers and only adding those of your private network.

```
ipfs bootstrap rm all - Remove all peers from the bootstrap list
```

2.7.2 Cluster

IPFS-Cluster | Documentation

Bootsrapping from an existing IPFS Cluster | Documentation

IPFS is designed so that a node only stores files locally that are specifically requested. The following is one way of populating a new IPFS node with all existing files from a pre-existing node.

To do so, first create a secret from `node0`, the original node, then share that secret with `node1`, the node you want to bootstrap from `node0`. Then `node1` runs the bootstrap command specifying the cluster's address and setting the `CLUSTER_SECRET` as an env variable.

node0

- `export CLUSTER_SECRET=$(od -vN 32 -An -tx1 /dev/urandom | tr -d ' \n')`
- `echo $CLUSTER_SECRET`
- `ipfs-cluster-service init`
- `ipfs-cluster-service daemon`

node1 (bootstrapping from node0)

- `export CLUSTER_SECRET=<copy from node0>`
- `ipfs-cluster-service init`
- `ipfs-cluster-service daemon --bootstrap /ip4/192.168.1.2/tcp/9096/ipfs/QmZjSoXUQgJ9tutP1rXjjNYwTrRM9QPhmD9GHVjbtgWxEn` // replace with what you see from running node0's daemon
- `ipfs-cluster-ctl peers ls` check your peers to see you've added node0 correctly

node0

- if you want to remove a peer after the bootstrapping is complete, the following command will do that and shut down the IPFS cluster
- `ipfs-cluster-ctl peers rm QmYFYwnFUkjFhJcSJJGN72wwedZnpQQ4aNpAtPZt8g5fCd`

2.8 PostgreSQL Database Backend

A PostgreSQL databse is utilized to prevent duplicate DSP transactions by creating, getting, and updating service events as needed. An external database can be set by ensuring the `node_env` variable in the `config.toml` file is set to `production`. The database settings may be specified with the `url` variable, e.g., `postgres://user:pass@example.com:5432/dbname`.

2.8.1 config.toml:

```
[database]

# url syntax: postgres://user:pass@example.com:5432/dbname, only necessary for␣
↪production

url = "postgres://user:pass@example.com:5432/dbname"

# production (uses above database_url for database)

node_env = "production"
```

2.8.2 How to install postgres on Ubuntu

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key␣
↪add -
echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb_release -cs`-pgdg main" |sudo␣
↪tee  /etc/apt/sources.list.d/pgdg.list
sudo apt update
sudo apt -y install postgresql-12 postgresql-client-12
```

2.8.3 Setup database and user

```
sudo su - postgres
psql
CREATE DATABASE dsp;
CREATE USER dsp WITH ENCRYPTED PASSWORD 'Put-Some-Strong-Password-Here';
GRANT ALL PRIVILEGES ON DATABASE dsp to dsp;
```

2.8.4 How to wipe local database

```
sudo su - postgres
psql
\l  # to list database and find your DB name, mine is "dsp"
\c dsp #to connect to the DB
drop table "Settings";
drop table "ServiceRequests";
<CTRL> d # exit
```

2.9 DSP Node

2.9.1 Prerequisites

- git

Linux

```
sudo su -
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash
export NVM_DIR="${XDG_CONFIG_HOME/:-$HOME/.}nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
# latest is 10.17.0 which has issues
nvm install 10.16.3
nvm use 10.16.3
exit
```

Ubuntu/Debian

```
sudo apt install -y make cmake build-essential python npm git
```

Centos/Fedora/AWS Linux:

```
sudo yum install -y make cmake3 python
```

2.9.2 Install

```
sudo su -
npm install -g pm2
npm install -g @liquidapps/dsp --unsafe-perm=true
exit
```

2.9.3 Configure Settings

Any changes to the `config.toml` file will require `setup-dsp` to be run again. Link to `sample-config.toml`

```
sudo su -
mkdir ~/.dsp
cp $(readlink -f `which setup-dsp` | xargs dirname)/sample-config.toml ~/.dsp/config.
↪toml
nano ~/.dsp/config.toml
exit
```

2.9.4 Launch DSP Services

```
sudo su -
setup-dsp
systemctl stop dsp
systemctl start dsp
systemctl enable dsp
exit
```

2.9.5 Check logs

```
sudo su -
pm2 logs
exit
```

2.9.6 Additional Logs

```
cd $(readlink -f `which setup-dsp` | xargs dirname)
cd logs
```

Output sample:

```
/root/.pm2/logs/readfn-dapp-service-node-error.log last 15 lines:
/root/.pm2/logs/dapp-services-node-out.log last 15 lines:
0|dapp-ser | 2019-06-03T00:46:49: services listening on port 3115!
0|dapp-ser | 2019-06-03T00:46:49: service node webhook listening on port 8812!

/root/.pm2/logs/demux-out.log last 15 lines:
1|demux    | 2019-06-05T14:41:12: count 1

/root/.pm2/logs/ipfs-dapp-service-node-out.log last 15 lines:
2|ipfs-dap | 2019-06-04T19:03:04: commited to: ipfs://
↪zb2rhXKc8zSVppFhKm8pHLBuyGb7vPeCnpZqcmjFnDLA9LLBb

/root/.pm2/logs/log-dapp-service-node-out.log last 15 lines:
3|log-dapp | 2019-06-03T00:46:49: log listening on port 13110!
3|log-dapp | 2019-06-03T00:46:52: LOG SVC NODE 2019-06-03T00:46:52.413Z INFO  index.
↪js:global:0             Started Service

/root/.pm2/logs/vaccounts-dapp-service-node-out.log last 15 lines:
4|vaccount | 2019-06-03T00:46:50: vaccounts listening on port 13129!

/root/.pm2/logs/oracle-dapp-service-node-out.log last 15 lines:
5|oracle-d | 2019-06-03T00:46:50: oracle listening on port 13112!

/root/.pm2/logs/cron-dapp-service-node-out.log last 15 lines:
6|cron-dap | 2019-06-03T00:46:50: cron listening on port 13131!

/root/.pm2/logs/readfn-dapp-service-node-out.log last 15 lines:
7|readfn-d | 2019-06-03T00:46:50: readfn listening on port 13141!
```

2.10 Packages

2.10.1 Register

Prepare and host dsp.json

```
{
    "name": "acme DSP",
    "website": "https://acme-dsp.com",
    "code_of_conduct":"https://...",
    "ownership_disclosure" : "https://...",
    "email":"dsp@acme-dsp.com",
    "branding":{
      "logo_256":"https://....",
      "logo_1024":"https://....",
      "logo_svg":"https://...."
    },
    "location": {
```

(continues on next page)

(continued from previous page)

```
      "name": "Atlantis",
      "country": "ATL",
      "latitude": 2.082652,
      "longitude": 1.781132
    },
    "social":{
      "steemit": "",
      "twitter": "",
      "youtube": "",
      "facebook": "",
      "github":"",
      "reddit": "",
      "keybase": "",
      "telegram": "",
      "wechat":""
    }

}
```

Prepare and host dsp-package.json

```
{
    "name": "Package 1",
    "description": "Best for low vgrabs",
    "dsp_json_uri": "https://acme-dsp.com/dsp.json",
    "logo":{
      "logo_256":"https://....",
      "logo_1024":"https://....",
      "logo_svg":"https://...."
    },
    "service_level_agreement": {
        "availability":{
            "uptime_9s": 5
        },
        "performance":{
            "95": 500
        }
    },
    "pinning":{
        "ttl": 2400,
        "public": false
    },
    "locations":[
        {
          "name": "Atlantis",
          "country": "ATL",
          "latitude": 2.082652,
          "longitude": 1.781132
        }
    ]
}
```

Register Package

Warning: packages are read only and can't be disabled yet.

- Mainnet DSP packages
- Kylin DSP packages

```
npm install -g @liquidapps/zeus-cmd
# the package must be chosen from the following list:
# packages: (ipfs, cron, log, oracle, readfn, vaccounts)
export PACKAGE=ipfs
export DSP_ACCOUNT=
# active key to sign package creation trx
export DSP_PRIVATE_KEY=
# customizable and unique name for your package
export PACKAGE_ID=package1
export EOS_CHAIN=mainnet
# or
export EOS_CHAIN=kylin
# the minimum stake quantity is the amount of DAPP and/or DAPPHDL that must be staked␣
↪to meet the package's threshold for use
export MIN_STAKE_QUANTITY="10.0000"
# package period is in seconds, so 86400 = 1 day, 3600 = 1 hour
export PACKAGE_PERIOD=86400
# the time to unstake is the greater of the package period remaining and the minimum␣
↪unstake period, which is also in seconds
export MIN_UNSTAKE_PERIOD=3600
# QUOTA is the measurement for total actions allowed within the package period to be␣
↪processed by the DSP.  1.0000 QUOTA = 10,000 actions. 0.0001 QUOTA = 1 action
export QUOTA="1.0000"
export DSP_ENDPOINT=https://acme-dsp.com
# package json uri is the link to your package's information, this is customizable␣
↪without a required syntax
export PACKAGE_JSON_URI=https://acme-dsp.com/package1.dsp-package.json

cd $(readlink -f `which setup-dsp` | xargs dirname)
zeus register dapp-service-provider-package \
    $PACKAGE $DSP_ACCOUNT $PACKAGE_ID \
    --key $DSP_PRIVATE_KEY \
    --min-stake-quantity $MIN_STAKE_QUANTITY \
    --package-period $PACKAGE_PERIOD \
    --quota $QUOTA \
    --network $EOS_CHAIN \
    --api-endpoint $DSP_ENDPOINT \
    --package-json-uri $PACKAGE_JSON_URI \
    --min-unstake-period $MIN_UNSTAKE_PERIOD
```

Or in cleos:

```
# currently available services: (ipfsservice1, cronservices, logservices1,␣
↪oracleservic, readfndspsvc, accountless1)
# the services use EOS account names to fascilitate usage
# service contract names may be found in the boxes/groups/services/SERVICE_NAME/
↪models/dapp-services/*.json file as the ( contract ) parameter
export SERVICE=ipfsservice1
# zeus command automatically adds QUOTA / DAPP, so we must add it here
```

(continues on next page)

(continued from previous page)

```
export QUOTA="1.0000 QUOTA"
export MIN_STAKE_QUANTITY="10.0000 DAPP"
export EOS_ENDPOINT=https://kylin-dsp-2.liquidapps.io # or mainnet: https://api.
↪eosnewyork.io
cleos -u $EOS_ENDPOINT push action dappservices regpkg "{\"newpackage\":{\"api_
↪endpoint\":\"$DSP_ENDPOINT\",\"enabled\":0,\"id\":0,\"min_stake_quantity\":\"$MIN_
↪STAKE_QUANTITY\",\"min_unstake_period\":\"$MIN_UNSTAKE_PERIOD\",\"package_id\":\"
↪$PACKAGE_ID\",\"package_json_uri\":\"$PACKAGE_JSON_URI\",\"package_period\":\"
↪$PACKAGE_PERIOD\",\"provider\":\"$DSP_ACCOUNT\",\"quota\":\"$QUOTA\",\"service\":\"
↪$SERVICE\"}}" -p $DSP_ACCOUNT
```

Example service contract name for LiquidAccounts: https://github.com/liquidapps-io/zeus-sdk/blob/master/boxes/groups/services/vaccounts-dapp-service/models/dapp-services/vaccounts.json#L7

List of all services, please note some of which may not be testable yet: https://github.com/liquidapps-io/zeus-sdk/tree/master/boxes/groups/services, see the stage for each service to monitor its development maturity (WIP - work in progress, Alpha, Beta, Stable).

output should be:

```
registering package:package1
✓package:package1 registered successfully
```

For more options:

```
zeus register dapp-service-provider-package --help
```

Don't forget to stake CPU/NET to your DSP account:

```
cleos -u $DSP_ENDPOINT system delegatebw $DSP_ACCOUNT $DSP_ACCOUNT "5.000 EOS" "95.
↪000 EOS" -p $DSP_ACCOUNT@active
```

Modify Package metadata:

Currently only `package_json_uri` and `api_endpoint` are modifyable. To signal to DSP Portals / Developers that your package is no longer in service, set your `api_endpoint` to `null`.

To modify package metadata: use the "modifypkg" action of the dappservices contract.

Using cleos:

```
cleos -u $DSP_ENDPOINT push action dappservices modifypkg "[\"$DSP_ACCOUNT\",\"
↪$PACKAGE_ID\",\"ipfsservice1\",\"$DSP_ENDPOINT\",\"https://acme-dsp.com/modified-
↪package1.dsp-package.json\"]" -p $DSP_ACCOUNT@active
```

2.11 Testing

2.11.1 Test your DSP with vRAM

Please note, if you wish to test on the mainnet, this will require the purchase of DAPP tokens or the use of DAPPHDL tokens (Air-HODL). In the case of Kylin, we provide a DAPP token faucet.

Create Mainnet or Kylin Account:

- *Kylin*
- Mainnet

Install Zeus:

```
npm install -g @liquidapps/zeus-cmd
```

Unbox coldtoken contract:

```
zeus unbox coldtoken
cd coldtoken
```

Compile and deploy contract for testing:

```
# your DSP's API endpoint
export DSP_ENDPOINT=
# a new account to deploy your contract to
export ACCOUNT=
# your new account's active public key
export ACTIVE_PUBLIC_KEY=
# compile coldtoken contract
zeus compile
cd contracts/eos
# set eosio.code permission
cleos -u $DSP_ENDPOINT set account permission $ACCOUNT active "{\"threshold\":1,\
↪"keys\":[{\"weight\":1,\"key\":\"$ACTIVE_PUBLIC_KEY\"}],\"accounts\":[{\"permission\
↪":{\"actor\":\"$ACCOUNT\",\"permission\":\"eosio.code\"},\"weight\":1}]}" owner -p
↪$ACCOUNT@active
# set contract
cleos -u $DSP_ENDPOINT set contract $ACCOUNT ./coldtoken
```

Select and stake to DSP:

```
# your DSP's account
export DSP_ACCOUNT=
# your DSP's service
export DSP_SERVICE=
# your DSP's package
export DSP_PACKAGE=
# your DSP's minimum stake quantity in DAPP or DAPPHDL (example: 10.0000 DAPP or 10.
↪0000 DAPPHDL)
export MIN_STAKE_QUANTITY=
# select DSP package
cleos -u $DSP_ENDPOINT push action dappservices selectpkg "{\"owner\":\"$ACCOUNT\",\
↪"provider\":\"$DSP_ACCOUNT\",\"service\":\"$DSP_SERVICE\",\"package\":\"$DSP_
↪PACKAGE\"}" -p $ACCOUNT
# stake to DSP package with DAPP
cleos -u $DSP_ENDPOINT push action dappservices stake "{\"owner\":\"$ACCOUNT\",\
↪"provider\":\"$DSP_ACCOUNT\",\"service\":\"$DSP_SERVICE\",\"quantity\":\"$MIN_STAKE_
↪QUANTITY\"}" -p $ACCOUNT
```

(continues on next page)

(continued from previous page)

```
# stake to DSP package with DAPPHDL, only available on mainnet
cleos -u $DSP_ENDPOINT push action dappairhodl1 stake "{\"owner\":\"$ACCOUNT\",\
↪"provider\":\"$DSP_ACCOUNT\",\"service\":\"$DSP_SERVICE\",\"quantity\":\"$MIN_STAKE_
↪QUANTITY\"}" -p $ACCOUNT
```

Run test commands:

```
# create coldtoken
cleos -u $DSP_ENDPOINT push action $ACCOUNT create "{\"issuer\":\"$ACCOUNT\",\
↪"maximum_supply\":\"1000000000 TEST\"}" -p $ACCOUNT
# issue some TEST
cleos -u $DSP_ENDPOINT push action $ACCOUNT issue "{\"to\":\"$ACCOUNT\",\"quantity\":\
↪"1000 TEST\",\"memo\":\"Testing issue\"}" -p $ACCOUNT
```

Test vRAM get table row:

```
# you must be in the root of the box to run this command
cd ../../
zeus get-table-row $ACCOUNT "accounts" $ACCOUNT "TEST" --endpoint $DSP_ENDPOINT |␣
↪python -m json.tool
# with curl:
curl http://$DSP_ENDPOINT/v1/dsp/ipfsservice1/get_table_row -d '{"contract":"CONTRACT_
↪ACCOUNT","scope":"SCOPE","table":"TABLE_NAME","key":"TABLE_PRIMARY_KEY"}' | python -
↪m json.tool
```

Transfer:

```
cleos -u $DSP_ENDPOINT push action $ACCOUNT transfer "{\"from\":\"$ACCOUNT\",\"to\":\
↪"natdeveloper\",\"quantity\":\"1000 TEST\",\"memo\":\"Testing transfer\"}" -p
↪$ACCOUNT
zeus get-table-row $ACCOUNT "accounts" "natdeveloper" "TEST" --endpoint $DSP_ENDPOINT␣
↪| python -m json.tool
```

Check logs on your DSP Node

```
pm2 logs
```

vRAM related actions to look for in a block explorer:

Look for “xcommit” and “xcleanup” actions on your contract: https://bloks.io/

- **xcommit** - The commit request instructs a DSP to write new data to their local IPFS cluster node. A developer can utilize the setData function from within their smart contract to first hash the new data in order to return a URI, before dispatching a commit request which is caught by the DSP node. In a similar way the getData function can be utilized in order to fetch the data for the smart contract or request a Warmup in case it is missing.
- **xcleanup** - A cleanup request sends a request to the DSP to evict a file from the cache. This is an asynchronous request.

More information on vRAM related actions can be found here: https://medium.com/@liquidapps/vram-guide-for-experts-f809c8f82a27

2.12 Claim Rewards

2.12.1 Claim your DAPP daily rewards:

```
cleos push action dappservices claimrewards "[\"$DSP_ACCOUNT\"]" -p $DSP_ACCOUNT
```

2.12.2 With Bloks.io:

Claim

2.13 Replay Contract

As a DSP, you will want the ability to replay a contract's vRAM (IPFS) related transactions to load that data into your IPFS cluster. We provide a file that does just that replay-contract.js.

To do this you will need to sign up for an API key from dfuse.io, you can select the *Server to Server* option from the dropdown when creating it. Dfuse offers free keys that last 24 hours, so there's no need to pay.

There are some mandatory and optional environment variables.

2.13.1 Mandatory:

```
export DFUSE_API_KEY=
# contract to replay
export CONTRACT=
export NODEOS_CHAINID=
↪"aca376f206b8fc25a6ed44dbdc66547c36c6c33e3a119ffbeaef943642f0e906" # < mainnet |␣
↪kylin > "5fff1dae8dc8e2fc4d5b23b2c7665c97f9e9d8edf2b6485a86ba311c25639191"
```

2.13.2 Optional:

```
export LAST_BLOCK= # defaults to 35000000, this is the last block to sync from, find␣
↪the first vRAM transaction for the contract and set the block before it
export DFUSE_ENDPOINT= # defaults to 'mainnet.eos.dfuse.io', can set to `kylin.eos.
↪dfuse.io`
export BLOCK_COUNT_PER_QUERY= # defaults to 1000000
export NODEOS_SECURED= # defaults to true
export NODEOS_HOST= # defaults to localhost
export NODEOS_PORT= # defaults to 13115
```

Once you've set those, simply run with:

```
sudo find / -name replay-contract.js
node /root/.nvm/versions/node/v10.16.0/lib/node_modules/@liquidapps/dsp/utils/ipfs-
↪service/replay-contract.js

# sent 6513 saved 7725.26KB 6.42KB/s Block:77756949
```

2.14 Cleanup IPFS and Oracle Entries

Sometimes IPFS or Oracle entries are not evicted from a developer's contract due to the DSP experiencing unpredictable behavior. This causes the developer's smart contract RAM supply to increase as the `ipfsentry` / `oracleentry` table rows are not evicted. If this happens, you may run the cleanup.js file with the following environment variables:

The cleanup script will auto detect which table to cleanup `ipfsentry` or `oracleentry` depending on which one is present on the contract. If both are set, you can use the `TABLE` env variable to specify which to cleanup.

2.14.1 Mandatory:

- `CONTRACT` contract to clean IPFS / oracle entries
- `DSP_ENDPOINT` the DSP's endpoint that you staked to for IPFS and/or Oracle services

```
export CONTRACT=lqdportfolio
export DSP_ENDPOINT=http://kylin-dsp-2.liquidapps.io
```

2.14.2 Optional:

- `CHUNK_SIZE` represents the number of async requests for cleanups to send to the DSP at a time
- `TABLE` type is auto detected based on the contract name (ipfs table: `ipfsentry` or oracle table: `oracleentry`), if a contract has both tables, you will use this variable to target both

```
export CHUNK_SIZE= # defaults to 5
export TABLE= # defaults to ipfsentry or oracleentry by detecting from contract
```

Then run with:

```
sudo find / -name cleanup.js
node /root/.nvm/versions/node/v10.16.0/lib/node_modules/@liquidapps/dsp/utils/cleanup.
↪js
```

2.15 Consumer Permissions

The consumer of DSP services may optionally create permissions that allow the consumer to pay for all CPU, NET, and RAM costs associated with the DSP services. This permission is optional. Without it, DSP services will continue to operate normally.

Process

- The consuming contract creates a `dsp` permission under the `active` permission

- The `dsp` permissions requires that each provider used by the consumer must be added, for example: provider1@active, provider2@active
- Each `xaction` required for the services used must be Link Authed to the `dsp` permission

2.15.1 Example of adding DSP_ACCOUNT_NAME_HERE@active to a DSP permission level

with Bloks.io

Login with your account's name using the cleos login option: https://kylin.bloks.io/wallet/permissions/advanced, and the cleos command will be auto generated for you.

- click "Add New Permission
- click on permission to get it to open up
- permission name: dsp
- parent: active
- threshold: 1
- add as many DSP to the accounts section with the active permission (heliosslene - active, uuddlrlrbass - active, etc)
- click save permission to have the cleos command auto generated

or Cleos

```
# CONTRACT_ACCOUNT_HERE - the account name that the consumer contract is deployed to
# DSP_ACCOUNT_NAME_HERE - the account name of the DSP staked to, if staked to␣
↪multiple DSPs, must add all DSP permissions
cleos -u https://kylin.eos.dfuse.io push transaction '{"delay_sec":0,"max_cpu_usage_ms
↪":0,"actions":[{"account":"eosio","name":"updateauth","data":{"account":"CONTRACT_
↪ACCOUNT_HERE","permission":"dsp","parent":"active","auth":{"threshold":1,"keys":[],
↪"accounts":[{"permission":{"actor":"DSP_ACCOUNT_NAME_HERE","permission":"active"},
↪"weight":1}],"waits":[]}},"authorization":[{"actor":"CONTRACT_ACCOUNT_HERE",
↪"permission":"active"}]}]}'
```

2.15.2 Example of linkauthing to each DSP xaction

with Bloks.io

Go here: https://kylin.bloks.io/wallet/permissions/link

- login using your contract's name for the cleos option
- Permission: dsp
- Contract name: CONTRACT_ACCOUNT_HERE, not the DSP name
- Contract action: xsignal
- Link Auth -> presto you have your cleos command
- You must repeat this process for all of the contract xactions you are using, you may find them by checking your ABI or using a block explorer to view your actions

or Cleos

```
# CONTRACT_ACCOUNT_HERE - the account name that the consumer contract is deployed to
# ACTION_NAME_HERE - action to linkauth dsp permission levle to
cleos -u https://kylin.eos.dfuse.io push transaction '{"delay_sec":0,"max_cpu_usage_ms
↪":0,"actions":[{"account":"eosio","name":"linkauth","data":{"account":"CONTRACT_
↪ACCOUNT_HERE","code":"CONTRACT_ACCOUNT_HERE","type":"ACTION_NAME_HERE","requirement
↪":"dsp"},"authorization":[{"actor":"CONTRACT_ACCOUNT_HERE","permission":"active"}]}
↪]}'
```

2.16 Upgrade DSP Node

For all new releases, please test on the Kylin testnet for at least one week before deploying to a production environment.

Link: sample-config.toml

```
sudo su -
systemctl stop dsp
systemctl stop ipfs
systemctl stop nodeos
# if changes to sample-config.toml syntax:
nano ~/.dsp/config.toml
pm2 del all
pm2 kill
npm uninstall -g @liquidapps/dsp
exit

# as USER
sudo chown ubuntu:ubuntu /home/ubuntu/.pm2/rpc.sock /home/ubuntu/.pm2/pub.sock
npm uninstall -g @liquidapps/dsp

sudo su -
npm install -g @liquidapps/dsp --unsafe-perm=true
# Ensure no new updates to the `sample-config.toml` file are present, if so, update␣
↪your config.toml accordingly.
sudo find / -name sample-config.toml
# nano <PATH>
setup-dsp
systemctl start nodeos
systemctl start ipfs
systemctl start dsp
exit
```

If a DSP is not updating properly, you may try `pm2 restart all` to restart all processes.

- search

CHAPTER 3

Liquidx

3.1 LiquidX Getting Started

LiquidX enables DAPP Network services to be used between chains. A user can stake DAPP on the eos mainnet for an account on another eosio based chain.

This is accomplished by creating a set of mappings between accounts. DSPs and Users must create a 2 way mapping where by they verify on the mainnet and on the chain in question that each account is linked. This topic is explored in more detail in the docs to follow.

3.1.1 Docs:

Use DAPP Network Services

Become a DSP on another chain

Add a Chain to LiquidX

3.2 Use DAPP Network Services

Guide:

- *Get Mapping Files*
- *Create Account mapping*

3.2.1 Create Account mapping

3.3 Become a DSP on another chain

LiquidX enables DSPs (DAPP Service Providers) to offer services on new chains. To do so, a DSP can aquire the necessary mapping files, verify the DSP's account name on the existing and new networks, and launch an eosio node based on that new network.

Becoming a DSP on a new chain requires having an existing DSP account on the EOS mainnet. The mainnet account will remain the account that claims DSP rewards.

Guide:

- *Get Mapping Files*
- *Create DSP account mapping*
- *Add to config.toml file*

3.3.1 Get mapping files

There are two kinds of mapping when it comes to LiquidX, there are JSON files that must be added on both DSPs and there are actions that must be run on the new chain and the primary chain.

The JSON files mapping the DAPP Network contracts on the new chain should be available from the network that deployed the `dappservicex` contract. This should get you:

- All the service file mappings, e.g., `liquidjungle.ipfs.service.json`

The JSON mapping files are needed to inform the DSP (DAPP Service Provider) of the account names hosting the DAPP Network contracts, e.g., `liquidjungle.ipfs.service.json`.

Here we will setup the JSON mapping files. This step must be performed on the primary network DSP and the new chain DSP.

For Jungle:

```
export BOX=liquidx-jungle
cd $(readlink -f `which setup-dsp` | xargs dirname)
cd models
npm i -g @liquidapps/zeus-cmd
zeus box add $BOX https://s3.us-east-2.amazonaws.com/liquidapps.artifacts/boxes/
↪60295170699618ff20b212195afaa0700e4f17a893974095782a069ad31809d9.zip
zeus unbox $BOX
mv ./$BOX/models/liquidx-mappings/ .
mv ./$BOX/models/local-sidechains/ .
rm -rf ./$BOX
```

Example service file mapping `./models/liquidx-mappings/liquidjungle.ipfs.service.json`

```
{
  "sidechain_name": "liquidjungle",
  "mainnet_account": "ipfsservice1",
  "chain_account": "ipfsservice1"
}
```

Example DSP file mapping `./models/liquidx-mappings/liquidjungle.uuddlrlrbass.json`

```
{
    "sidechain_name":"liquidjungle",
    "mainnet_account":"uuddlrlrbass",
    "chain_account":"uuddlrlrbass"
}
```

Example sidechain file mapping `./models/local-sidechains/liquidjungle.json`

```
{
    "dsp_port":3115,
    "nodeos_port":8888,
    "nodeos_endpoint":"http://localhost:8888",
    "nodeos_host":"localhost",
    "nodeos_state_history_port":8887,
    "nodeos_p2p_port":9876,
    "demux_port":3195,
    "name":"liquidjungle"
}
```

3.3.2 Create DSP account mapping

On the eos mainnet, or jungle, you will need to connect your DSP account to the new chain's DSP account. This is done using the `addaccount` command.

- owner {name} - DSP account name on mainnet or jungle
- chain_account {name} - DSP account name on new chain
- chain_name {name} - Account name LiquidX contract is deployed to

Cleos example:

```
cleos -u https://kylin.eos.dfuse.io push transaction '{"delay_sec":0,"max_cpu_usage_ms
↪":0,"actions":[{"account":"liquidjungle","name":"addaccount","data":{"owner":
↪"uuddlrlrbass","chain_account":"uuddlrlrbass","chain_name":"liquidjungle"},
↪"authorization":[{"actor":"uuddlrlrbass","permission":"active"}]}]}'
```

Then on the new chain, submit an `adddsp` action.

- owner {name} - DSP name on new chain
- dsp {name} - DSP name on mainnet or Jungle

Cleos example:

```
cleos -u https://api.jungle.alohaeos.com push transaction '{"delay_sec":0,"max_cpu_
↪usage_ms":0,"actions":[{"account":"dappservicex","name":"adddsp","data":{"owner":
↪"uuddlrlrbass","dsp":"uuddlrlrbass"},"authorization":[{"actor":"uuddlrlrbass",
↪"permission":"active"}]}]}'
```

With that you have 2 way mapped your DSP account name. On the primary network, the DSP's account has been linked to the new chain's network. And on the new network, the primary network's DSP account has been verified.

3.3.3 Add to config.toml file

In order to enable a sister chain, you must add the following to your mainnet or Jungle's `config.toml` file. This is the file that holds the environment variables for your DSP's API instance.

```
# LIQUIDX_CONTRACT_NAME - this is the name of the account that has the LiquidX code␣
↪deployed to it.

[sidechains]
  [sidechains.LIQUIDX_CONTRACT_NAME]
    dsp_port = 12346
    nodeos_secured = false
    nodeos_host = "localhost"
    nodeos_port = 2424
    nodeos_state_history_port = 12341
    demux_port = 1232
    name = "<LIQUIDX_CONTRACT_NAME>"
    dsp_account = "<DSP SIDECHAIN ACCOUNT>"
    dsp_private_key = "<DSP SIDECHAIN PRIVATE KEY>"
    chainid = "<SIDECHAIN CHAIN ID>"
    mapping = "dappservices:dappservicex,cronservices:cronservicex,
↪ipfsservice1:ipfsservice2"
```

3.4 Add a Chain to LiquidX

The following steps will cover how to enable the DAPP Network on a new chain. We'll start by creating the necessary accounts and loading the contracts. Then we'll create a series of mappings, both on chain and in JSON. Finally after running a few actions, you're good to go.

Guide:

- *Create accounts and set contracts*
- *Creating mapping files for services*
- *Creating mapping files for DSPs*
- *Creating mapping file for Chain*
- *Create accounts and set contracts*
- *Register chain*
- *Create service contract mappings*

3.4.1 Create accounts and set contracts

There are a handlful of accounts that must be created on the new chain. Accounts can be created with the account creator: https://monitor.jungletestnet.io/#account. Note that you do not have to use these names, but it may make more sense for everyone if you tried to.

account name - contract name - pretty name

```
# for key creation
cleos create key --to-console
```

Mainnet:

- This account name is intended to represent the new chain - liquidx

For example I used `liquidjungle` *to link Kylin <> Jungle*

New Chain:

- dappservicex - dappservicex
- ipfsservice1 - ipfsservice - ipfs
- cronservices - cronservice - cron
- readfndspsvc - readfnservice - readfn
- logservices1 - logservice - log
- oracleservic - oracleservice - oracle
- accountless1 - vaccountsservice - vaccounts
- liquidstorag - storageservice - storage
- authfndspsvc - authservice - auth
- vcpuservices - vcpuservice - vcpu

After each account is created, the associated contract must be uploaded to each account. All contracts may be found by unboxing the `liquidx` box from zeus.

```
npm i -g @liquidapps/zeus-cmd
zeus unbox liquidx
cd liquidx
zeus compile
cd contracts/eos
```

Set each contract, will need to hit up the faucet: https://monitor.jungletestnet.io/#faucet.

```
# rinse wash repeat
export JUNGLE_TEST_ACCOUNT=
# active key
export JUNGLE_TEST_PUBLIC_KEY=
export CONTRACT_NAME=
export EOS_ENDPOINT=http://jungle.atticlab.net:8888

cleos -u $EOS_ENDPOINT system buyram $JUNGLE_TEST_ACCOUNT $JUNGLE_TEST_ACCOUNT "50.
↪0000 EOS" -p $JUNGLE_TEST_ACCOUNT@active
cleos -u $EOS_ENDPOINT system delegatebw $JUNGLE_TEST_ACCOUNT $JUNGLE_TEST_ACCOUNT
↪"10.0000 EOS" "40.0000 EOS" -p $JUNGLE_TEST_ACCOUNT@active
cleos -u $EOS_ENDPOINT set account permission $JUNGLE_TEST_ACCOUNT active "{\
↪"threshold\":1,\"keys\":[{\"weight\":1,\"key\":\"$JUNGLE_TEST_PUBLIC_KEY\"}],\
↪"accounts\":[{\"permission\":{\"actor\":\"$JUNGLE_TEST_ACCOUNT\",\"permission\":\
↪"eosio.code\"},\"weight\":1}]}" owner -p $JUNGLE_TEST_ACCOUNT@active
cleos -u $EOS_ENDPOINT set contract $JUNGLE_TEST_ACCOUNT $CONTRACT_NAME
```

3.4.2 Creating mapping files for services

Within the `/models/liquidx-mappings` directory, you will see a series of mapping files. You will need to create a mapping file for each of the accounts above on the new chain.

Here is an example:

File name: `liquidjungle.ipfs.service.json` - <sidechain_name><service_pretty_name>.service.json

- sidechain_name - name of sidechain to add liquidx to, name of account liquidx contract on mainnet is deployed to
- service_pretty_name - the name found in the `zeus-sdk/boxes/groups/services/${SERVICE}/models/dapp-services/${service_pretty_name}.json`

For example: https://github.com/liquidapps-io/zeus-sdk/blob/master/boxes/groups/services/ipfs-dapp-service/models/dapp-services/ipfs.json#L2

- mainnet_account - account on EOS mainnet
- chain_account - account on side chain (assuming you got the account above, if not, use the account you got)

For example, if ipfsservice1 was taken, but you got ipfsservice2, ipfsservice2 would be the `chain_account`

```
{
    "sidechain_name":"liquidjungle",
    "mainnet_account":"ipfsservice1",
    "chain_account":"ipfsservice1"
}
```

3.4.3 Creating mapping files for DSPs

Using the same syntax, each DSP needs their own mapping file to be stored in the same `/models/liquidx-mappings` directory.

File name: `liquidjungle.DSP_NAME_HERE.json`

```
{
    "sidechain_name":"liquidjungle",
    "mainnet_account":"uuddlrlrbass",
    "chain_account":"uuddlrlrbass"
}
```

3.4.4 Creating mapping file for Chain

The new chain itself needs its own mapping file to be stored in `/models/local-sidechains` directory. This mapping file represents a DSP operating on the new chain.

File name: `liquidjungle.json`

```
{
    "dsp_port":3115,
    "nodeos_port":8888,
    "nodeos_endpoint":"http://localhost:8888",
    "nodeos_host":"localhost",
    "nodeos_state_history_port":8887,
    "nodeos_p2p_port":9876,
    "demux_port":3195,
    "name":"liquidjungle"
}
```

3.4.5 Register chain

You must execute the `setchain` action on the account that has deployed the `liquidx` code on the eos mainnet. The syntax is as follows:

- chain_name {name} - name of account deploying chain, e.g., liquidjungle
- chain_meta {chain_metadata_t} - chain data
  - is_public {bool} - whether the chain is public

- is_single_node {bool} - whether chain is a single node
- dappservices_contract {std::string} -
- chain_id {std::string} - chain ID of sidechain
- type {std::string} - type of blockchain, e.g., EOSIO
- endpoints {std::vectorstd::string} - list of public endpoints for developers to use
- type {std::string} - type of blockchain, e.g., EOSIO
- chain_json_uri {std::vectorstd::string} - publicly available json file that declares chain statistics

Example cleos command:

```
cleos -u https://kylin.eos.dfuse.io push transaction '{"delay_sec":0,"max_cpu_usage_ms
↪":0,"actions":[{"account":"liquidjungle","name":"setchain","data":{"chain_name":
↪"liquidjungle","chain_meta":{"is_public":true,"is_single_node":false,"dappservices_
↪contract":"dappservicex","chain_id":
↪"e70aaab8997e1dfce58fbfac80cbbb8fecec7b99cf982a9444273cbc64c41473","type":"EOSIO",
↪"endpoints":[],"p2p_seeds":[],"chain_json_uri":""}},"authorization":[{"actor":
↪"liquidjungle","permission":"active"}]}]}'
```

3.4.6 Create service contract mappings

On the new chain you will need to run the `setlink` command for each service you created above. The `setlink` command represents a 1 way mapping, only a 1 way mapping is required for the service contracts.

- `owner` - service contract name on new chain, e.g., `ipfsservice1`
- `mainnet_owner` - mainnet version of service contract name, e.g., `ipfsservice1`

Example cleos command:

```
cleos -u https://api.jungle.alohaeos.com push transaction '{"delay_sec":0,"max_cpu_
↪usage_ms":0,"actions":[{"account":"dappservicex","name":"setlink","data":{"owner":
↪"ipfsservice1","mainnet_owner":"ipfsservice1"},"authorization":[{"actor":
↪"ipfsservice1","permission":"active"}]}]}'
```

And now you're setup to begin configuring DSPs and then enabling user to use DAPP Network services.

- search

CHAPTER 4

Services

4.1 LiquidAuthenticator Service

4.1.1 Overview

Authentication of offchain APIs and services using EOSIO permissions and contract

4.1.2 Stage

Alpha

4.1.3 Version

0.4

4.1.4 Contract

`authfndspsvc`

4.1.5 Box

auth-dapp-service

4.1.6 Service Commands

authusage

4.1.7 Tests

- auth-client.spec.js
- authenticator.spec.js

4.1.8 Implementation

4.2 LiquidBilling Service

4.2.1 Overview

Transaction signing service for resource payment

4.2.2 Stage

WIP

4.2.3 Version

0.0

4.2.4 Contract

`liquidbillin`

4.2.5 Box

bill-dapp-service

4.2.6 Service Commands

sdummy2

4.2.7 Tests

- bill.spec.js

4.2.8 Implementation

4.3 LiquidScheduler Service

4.3.1 Overview

Scheduled Transactions

4.3.2 Stage

beta

4.3.3 Version

0.9

4.3.4 Contract

`cronservices`

4.3.5 Box

cron-dapp-service

4.3.6 Service Commands

schedule

4.3.7 Tests

- cronconsumer.spec.js
- Consumer Contract Example

4.3.8 Implementation

4.4 LiquidDNS Service

4.4.1 Overview

DSP Hosted DNS Service

4.4.2 Stage

WIP

4.4.3 Version

0.5

4.4.4 Contract

`dnsservices1`

4.4.5 Box

dns-dapp-service

4.4.6 Service Commands

dnsq

4.4.7 Tests

- dnsconsumer.spec.js
- Consumer Contract Example

4.4.8 Implementation

4.5 LiquidArchive Service

4.5.1 Overview

History API Provisioning

4.5.2 Stage

WIP

4.5.3 Version

0.0

4.5.4 Contract

`historyservc`

4.5.5 Box

history-dapp-service

4.5.6 Service Commands

hststore

hsthold

hstserve

hstreg

4.5.7 Tests

- history.spec.js

4.5.8 Implementation

4.6 LiquidVRAM Service

4.6.1 Overview

Virtual Memory Service

4.6.2 Stage

Stable

4.6.3 Version

1.1

4.6.4 Contract

`ipfsservice1`

4.6.5 Box

ipfs-dapp-service

4.6.6 Service Commands

commit

cleanup

warmup

4.6.7 Tests

- dappservices.spec.js
- ipfsconsumer.spec.js
- oldipfscons.spec.js
- Consumer Contract Example

4.6.8 Implementation

4.7 LiquidKMS Service

4.7.1 Overview

Key Management Service

4.7.2 Stage

WIP

4.7.3 Version

0.0

4.7.4 Contract

`kmsservices1`

4.7.5 Box

kms-dapp-service

4.7.6 Service Commands

sdummy3

4.7.7 Tests

- kmsconsumer.spec.js

4.7.8 Implementation

4.8 LiquidLog Service

4.8.1 Overview

Log Service

4.8.2 Stage

Beta

4.8.3 Version

0.9

4.8.4 Contract

`logservices1`

4.8.5 Box

log-dapp-service

4.8.6 Service Commands

logevent

logclear

4.8.7 Tests

- logconsumer.spec.js
- Consumer Contract Example

4.8.8 Implementation

4.9 LiquidHarmony Service

4.9.1 Overview

Web/IBC/XIBC/VCPU/SQL Services

4.9.2 Stage

Beta

4.9.3 Version

0.9

4.9.4 Contract

`oracleservic`

4.9.5 Box

oracle-dapp-service

4.9.6 Service Commands

geturi

orcclean

4.9.7 Tests

- oracleconsumer.spec.js
- Consumer Contract Example

4.9.8 Implementation

4.10 LiquidLens Service

4.10.1 Overview

Read Functions Service

4.10.2 Stage

Alpha

4.10.3 Version

0.9

4.10.4 Contract

`readfndspsvc`

4.10.5 Box

readfn-dapp-service

4.10.6 Service Commands

rfnuse

4.10.7 Tests

- readfnconsumer.spec.js
- Consumer Contract Example

4.10.8 Implementation

4.11 LiquidLink Service

4.11.1 Overview

IBC MultiSig Service

4.11.2 Stage

Alpha

4.11.3 Version

0.5

4.11.4 Contract

`signfndspsvc`

4.11.5 Box

sign-dapp-service

4.11.6 Service Commands

signtrx

sgcleanup

4.11.7 Tests

- sign.spec.js

4.11.8 Implementation

4.12 LiquidStorage Service

4.12.1 Overview

Distributed storage and hosting

4.12.2 Stage

Alpha

4.12.3 Version

0.5

4.12.4 Contract

`liquidstorag`

4.12.5 Box

storage-dapp-service

4.12.6 Service Commands

sdummy

4.12.7 Tests

- storage.spec.js

4.12.8 Implementation

4.13 LiquidAccounts Service

4.13.1 Overview

Allows interaction with contract without a native EOS Account

4.13.2 Stage

Beta

4.13.3 Version

0.9

4.13.4 Contract

`accountless1`

4.13.5 Box

vaccounts-dapp-service

4.13.6 Service Commands

vexec

4.13.7 Tests

- vaccountsconsumer.spec.js
- Consumer Contract Example

4.13.8 Implementation

4.14 VCPU Service

4.14.1 Overview

DSP Hosted Computation Service

4.14.2 Stage

PoC

4.14.3 Version

0.1

4.14.4 Contract

`vcpuservices`

4.14.5 Box

vcpu-dapp-service

4.14.6 Service Commands

vrun

vrunclean

4.14.7 Tests

- vcpuconsumer.spec.js
- Consumer Contract Example

4.14.8 Implementation

- search

CHAPTER 5

DAPP Tokens

5.1 DAPP Token Overview

The DAPP token is a multi-purpose utility token that grants access to the DAPP Network. It is designed to power an ecosystem of utilities, resources, and services specifically serving the needs of dApp developers building user-centric dApps.

5.1.1 Videos

- EOS Weekly - The LiquidApps Game-Changer
- EOS Weekly - Unlimited DSP Possibilities

5.1.2 Have questions?

- Join our Telegram channel

5.1.3 Want more information?

- Read our whitepaper and subscribe to our Medium posts.

5.2 DAPP Tokens Tracks

Link to auction: https://liquidapps.io/auction

5.2.1 Instant Track

Users wishing to purchase DAPP with EOS tokens can do so through the instant track. Simply send EOS to the Instant Registration Track Vendor Smart Contract and you will receive your DAPP tokens at the end of the current cycle (see "Claiming DAPP Tokens" for further information about the claiming process).

5.2.2 Regular Track

The Regular Registration Track provides flexibility in purchasing DAPP tokens. You can use EOS tokens for any desired purchase amount. For amounts exceeding 15,000 Swiss Franc (CHF) you may also purchase with ETH, BTC or Fiat.

In order to open up the opportunity to all potential purchasers the DAPP Generation Event includes a verified track for buyers who wish to use their ETH, BTC, FIAT or EOS to purchase DAPP tokens.

If you wish to participate in the DAPP Generation Event through the Regular Registration Track, you must complete a KYC (Know Your Customer) verification process, facilitated by Altcoinomy, a Swiss-based licensed KYC operator.

5.3 Claiming DAPP Tokens

5.3.1 Automatic

The auto claim mechanism does not require participants to push an action themselves to claim the tokens. This is handled by the website automatically at the end of each cycle.

5.3.2 Manual

The manual claim function is always available and participants can claim their DAPP Tokens immediately after the cycle ends by sending an explicit action depending on the track they selected.

Instant Registration Track

Regular Registration Track

Login with the wallet of your choice and enter your account in the "payer" field (**YOUR_ACCOUNT_HERE**) and hit "Push Transaction".

5.4 DAPP Tokens Distribution

The year-long DAPP token Generation Event began on February 26th, 2019 and will last until January 2020, for a total of 333 days. These 333 days will be split into 444 18-hour cycles, with each cycle receiving an allocation of 1,127,127 tokens.

The DAPP tokens are distributed through two unique independent purchase tracks—the Instant Registration Track and the Regular Registration Track. At the end of each cycle, each one of the two Registration Tracks distributes 563,063.0630 DAPP tokens amongst that cycle's participants, proportional to the amount of EOS sent by each purchaser in that cycle.

5.4.1 Integrity is Our Priority

Blockchain technology has the potential to enable a more free and fair economy to emerge by introducing an unprecedented level of transparency and accountability to markets. At LiquidApps, we are firm proponents of the free market ethos. Maintaining the integrity of the DAPP Generation Event is of the utmost importance to us, and, as such, LiquidApps hereby commits to abstaining from participation in the DAPP Token Generation Event.

More information may be found in our whitepaper

5.5 Air-HODL

A total amount of 100,000,000 DAPP will be allocated and divided between all the accounts that hold EOS at block #36,568,000 ("Pioneer Holders") and distributed via our unique Air-HODL mechanism.

You can view all snapshot information here.

The Air-HODLed DAPP tokens will be distributed on a block by block basis, matching up to a maximum of 3 million EOS per account. The tokens will be continuously vested on a block to block basis over a period of 2 years, so the complete withdrawal will only be possible at the end of this period. These 2 years began as soon as the DAPP Generation Event was launched. Any Pioneer Holder choosing to withdraw the Air-HODLed tokens before the end of those 2 years will only receive the vested portion (i.e. 50% of the distributed DAPP tokens will be vested after 1 year). The remainder of their unvested DAPP tokens will be distributed to Pioneer Holders who are still holding their Air-HODL DAPP tokens.

HODLers are allowed to stake their vested Air-HODLed tokens immediately using our new staking mechanics. Withdrawing the tokens will transfer the vested tokens to their DAPP account, forfeiting the unvested portion to be redistributed amongst remaining eligible participants.

You can get more information on the Air-HODL and view your balance at: https://liquidapps.io/air-hodl

- search

CHAPTER 6

FAQs

6.1 Frequently Asked Questions The DAPP Token

6.1.1 What is the DAPP token?

The DAPP token is a multi-purpose utility token designed to power an ecosystem of utilities, resources, & services specifically serving the needs of dApp developers building user-centric dApps.

6.1.2 What is the supply schedule of DAPP token?

DAPP will have an intial supply of 1 billion tokens. The DAPP Token Smart Contract generates new DAPP Tokens on an ongoing basis, at an annual inflation rate of 1-5%.

6.1.3 How are DAPP tokens distributed?

50% of the DAPP tokens will be distributed in a year-long token sale, while 10% will be Air-Hodl'd to EOS holders. The team will receive 20% of the DAPP tokens, of which 6.5% is unlocked and the rest continuously vested (on a block-by-block basis) over a period of 2 years. Our partners and advisors will receive 10% of the DAPP tokens, with the remaining 10% designated towards our grant and bounty programs.

6.1.4 Why do you need to use DAPP Token and not just EOS?

While we considered this approach at the beginning of our building journey, we decided against it for a number of reasons:

- We look forward to growing the network exponentially and will require ever more hardware to provide quick handling of large amounts of data accessible through a high-availability API. It is fair to assume that this kind of service would require significant resources to operate and market, thus it would not be optimal for a BP to take on this as a "side-job" (using a "free market" model that allows adapting price to cost).
- The BPs have a special role as trusted entities in the EOS ecosystem. DSPs are more similar to a cloud service in this respect, where they are less reputational and more technical. Anyone, including BPs, corporate entities, and private individuals, can become a DSP.
- Adding the DAPP Network mechanism as an additional utility of the EOS token would not only require a complete consensus between all BPs, but adoption by all API nodes as well. Lack of complete consensus to adopt this model as an integral part of the EOS protocol would result in a hard fork. (Unlike a system contract update, this change would require everyone's approval, not only 15 out of 21).
- Since the DAPP Network's mechanism does not require the active 21 BPs' consensus, it doesn't require every BP to cache ALL the data. Sharding the data across different entities enables true horizontal scaling. By separating the functions and reward mechanisms of BPs and DSPs, The DAPP Network creates an incentive structure that makes it possible for vRAM to scale successfully.
- We foresee many potential utilities for vRAM. One of those is getting vRAM to serve as a shared memory solution between EOS side-chains when using IBC (Inter-Blockchain Communication). This can be extended to chains with a different native token than EOS, allowing DAPP token to be a token for utilizing cross-chain resources.
- We believe The DAPP Network should be a separate, complementary ecosystem (economy) to EOS. While the EOS Mainnet is where consensus is established, the DAPP Network is a secondary trustless layer. DAPP token, as the access token to the DSPs, will potentially power massive scaling of dApps for the first time.

6.1.5 Why is the sale cycle 18 hours?

An 18 hour cycle causes the start and end time to be constantly changing, giving people in all time zones an equal opportunity to participate.

6.1.6 What is an airHODL?

An Air-HODL is an airdrop with a vesting period. EOS token holders on the snapshot block receive DAPP tokens on a pro-rata basis every block, with the complete withdrawal of funds possible only after 2 years. Should they choose to sell their DAPP tokens, these holders forfeit the right to any future airdrop, increasing the share of DAPP tokens for the remaining holders.

6.1.7 Is this an EOS fork?

The DAPP Network is not a fork nor a side-chain but a trustless service layer (with an EOSIO compatible interface to the mainnet), provided by DSPs (DAPP Service providers). This layer potentially allows better utilization of the existing resources (the RAM and CPU resources provided to you as an EOS token holder). It does not require a change in the base protocol (hard fork) nor a change in the system contract. DSPs don't have to be active BPs nor trusted/elected entities and can price their own services.

6.2 Frequently Asked Questions DAPP Service Providers (DSPs)

6.2.1 What is a DSP?

DSPs are individuals or entities who provide external storage capacity, communication services, and/or utilities to dApp developers building on the blockchain, playing a crucial role in the DAPP network.

6.2.2 Who can be a DSP?

DSPs can be BPs, private individuals, corporations, or even anonymous entities. The only requirement is that each DSP must meet the minimum specifications for operating a full node on EOS.

6.2.3 Are DSPs required to run a full node?

While DSPs could use a third-party node, this would add latency to many services, including vRAM. In some cases, this latency could be significant. LiquidApps does not recommend running a DSP without a full node.

6.2.4 How are DSPs incentivized?

DSPs receive 1-5% of token inflation proportional to the total amount of DAPP tokens staked to their service packages.

6.3 Frequently Asked Questions vRAM

6.3.1 Why do I need vRAM?

RAM is a memory device used to store smart contract data on EOS. However, its limited capacity makes it difficult to build and scale dApps. vRAM provides dApp developers with an efficient and affordable alternative for their data storage needs.

6.3.2 How is vRAM different from RAM?

vRAM is a complement to RAM. It is an alternative storage solution for developers building EOS dApps that are RAM-compatible, decentralized, and enables storing & retrieving of potentially unlimited amounts of data affordably and efficiently. It allows dApp developers to cache all relevant data in RAM to distributed file storage systems (IPFS, BitTorent, HODLONG) hosted by DAPP Service Providers (DSPs), utilizing RAM to store only the data currently in use. vRAM transactions are still stored in chain history and so are replayable even if all DSPs go offline.

6.3.3 How can we be sure that data cached with DSPs is not tampered with?

DSPs cache files on IPFS, a decentralized file-storage system that uses a hash function to ensure the integrity of the data. You can learn more about IPFS here: https://www.youtube.com/watch?time_continue=2&v=8CMxDNuuAiQ

6.3.4 How much does vRAM cost?

Developers who wish to use the vRAM System do so by staking DAPP tokens to their chosen DSP for the amount specified by the Service Package they've chosen based on their needs. By staking DAPP, they receive access to the DSP services, vRAM included.

- search

CHAPTER 7

Patch Notes

7.1 latest

7.1.1 LiquidVRAM Service

- **Backwards Compatability Warning**
  - To Support new features some schema changes have taken place
  - If you already have a vram contract in production, it is recomended that you do not use the changes
  - Migration details and tools will be provided at a later time
  - To use the new features place `#define USE_ADVANCED_IPFS` at the start of your contract
- New Advanced Multi Index features
  - Primary key may be uint32, uint64, uint128, and checksum256
  - Ability to backup, restore, and clear vram datasets with versioning

7.1.2 @liquidapps/dsp

- add DSP console log in `common.js` if minimum stake threshold not met for account's DAPP stake to DSP's service package
- add reconnect mechanism to demux nodeos websocket
- update eos 1.8.7 nodeos
- fixes
  - add custom permissions for xcallback in generic-dapp-service-node file
  - fix cron reschedule on error, use `nextTrySeconds` time.

7.1.3 @liquidapps/zeus-cmd

- add `--type=local` flag to `zeus deploy box` command: deploys boxes locally to `~/.zeus/boxes/` instead of `IPFS` or `s3`. *Must use with the `--update-mapping` flag*. Together both flags (`zeus deploy box --type=local --update-mapping`) updates the `mapping.js` file with `file://..` as the pointer | thank you prcolaco
- made `--type=local` and `--update-mapping` flags default for `zeus deploy box` command
- only use invalidation of ipfs with `zeus deploy box` command when the `--type` is `ipfs` | thank you prcolaco
- modified and fixed ipfs cleanup script to support oracle cleanups
- allow `zeus compile <CONTRACT_NAME>`, zeus now allows you to only compile a contract by its name if you like, or you can run `zeus compile` to run all
- add `kill-port` npm dependency to `eos-extensions` box
- move `ipfs-daemon` dependency from `boxes/groups/core/build-extensions/zeus-box.json` to `boxes/groups/dapp-network/dapp-services/zeus-box.json` as `IPFS` is only needed with the `dapp-services` box
- add `utils/ipfs-service/get-table.js` - Reads all vRAM tables of a smart contract and stores them with the naming syntax: `${contract_name}-${table_name}-table.json`
- add `utils/ipfs-service/get-ordered-keys.js` - Prints ordered vRAM table keys in ascending order account/table/scope. This can be used to iterate over the entire table client side
- allow `zeus test <CONTRACT_NAME>`, zeus now allows you to only compile/test a contract by its name if you like, or you can run `zeus test` to compile/test all
- add `zeus vaccounts push-action test1v regaccount '{"vaccount":"vaccount1"}'`
- add ability to import/export LiquidAccount keys
- update eos 1.8.7 nodeos
- implement storage `dapp-client` into storage service test `storage-dapp-service/test/storage.spec.js`
- build `dapp-client` from source instead of installing by adding step to `start-localenv`
- use base58 instead of default base32 for LiquidStorage's `ipfs.files.add` to match ipfs service
- add `zeus test -c` alias to compile all contracts, `zeus test` now does not compile by default
- update eos 1.8.8 nodeos
- flag ipfsentries as pending commit to prevent duplicate requests
  - If a contract uses a shardbucket multiple times, it will only have unique commits
  - If multiple actions in the same block (or prior to the xcommit) need to lookup the same shardbucket, there will be a single unique commit, and no additional warmups required
  - If a contract uses a delayed commit, this delayed commit won't be overwritten by an immediate commit
- update eos 1.8.9 and eosio.cdt 1.7.0
- add `zeus box create` and `zeus box add` commands
- fixes
  - update example frontend to eosjs2 and latest scatter
  - update cleanup script to work with new dsp logic

  - add CONTRACT_END syntax to example contract
  - fix cardgame unit test
    * use dapp-client for vaccounts
    * move xvinit for vaccounts to happen in migration
    * add xvinit to coldtoken contract
    * update to eosj2
  - fix chess.json to enable migration by updating contract / account
  - fix OSX `zeus deploy box` breaking issue
  - remove prints from vaccount code to prevent `required service` error
  - remove `all-dapp-services` box from `templates-emptycontract-eos-cpp` (zeus create contract)

7.1.4 @liquidapps/dapp-client

- fixes
  - add fix text encode/decode in vaccounts service

7.1.5 docs

- removed `read-mode = head` from default `config.ini` setup for eosio node
- clarified `wasm-runtime = wabt` must be used over `wasm-runtime = wavm` due to bugs in `wavm`
- add `zeus compile <CONTRACT_NAME>` syntax to *zeus-getting-started*
- update path for `cleanup.js` script for DSPs
- add cleanup oracle info to *Cleanup IPFS and Oracle Entries*
- fixed little mistakes in *vram-getting-started*
- added usage docs for `get-table` and `get-ordered-keys`
- update `chain-state-db-size-mb` from `131072` to `16384` see here
- update eos 1.8.7 nodeos
- update cardgame link to: http://elemental.liquidapps.io/
- update eos 1.8.8 nodeos

7.1.6 dappservices contract

- add usagex for LiquidX and other off chain service billing LiquidStorage, LiquidLens, LiquidAuth
- contract pays for CPU/NET/RAM associated with xactions `xwarmup`, `xsignal`, `xcommit`, `xdcommit`, `xvexec`, etc
- fixes
  - add DAPP token assertion to `regpkg` command to ensure DAPP symbol and 4 decimals of precision used

7.2 2.0.3107

7.2.1 @liquidapps/dsp - 2.0.3107-latest

- add `'Content-Type': 'application/json'` to oracle `https+post+json` request
- add timeout proxy for database calls
- add LiquidX
- add LiquidHarmony, extension oracle service that allows plug and play oracle options (Web/IBC/XIBC/VCPU/SQL Services)
- add LiquidSQL, state storage alternative for smart contracts
- dappservicesx contract - add setlink (create link between side chain account and mainnet owner), adddsp (side chain account add DSP name), rmvdsp (side chain - account remove DSP name)
- liquidx contract - add addaccount (add sidechain account to allow billing to another chain account) and rmvaccount (remove link) actions
- add sidechain billing to dapp-services-node/common.js
- add LiquidKMS boilerplate
- add LiquidStorage node logic for unpin / upload_public
- add LiquidBilling boilerplate

7.2.2 @liquidapps/zeus-cmd - 2.0.3107

- update eos 1.8.6
- add example portolio dapp
- update LiquidStorage unit test
- add boxes: oracle-web oracle-self-history oracle-foreign-chain oracle-sister-chain oracle-wolframalpha oracle-random oracle-sql oracle-vcpu
- split up oracle services
- add functional LiquidStorage unit test
- fixes
  - replace unzip install with unzipper, allow node v11
  - update create contract example unit test eosjs2

7.2.3 @liquidapps/dapp-client - 2.0.3107

- add LiquidStorage client extension to upload / unpin

7.2.4 docs

- add local postgresql info
- add zeus-ide

- replace unzip install with unzipper, allow node v11
- update overview section with new links / videos
- add dapp-client section
- add example portolio dapp
- update oracle getting started links
- LiquidAuthenticator - WIP → Alpha
- LiquidBilling - WIP - Transaction signing service for resource payment
- LiquidKMS - WIP - Key Management Service
- LiquidStorage - WIP → Alpha
- LiquidSQL - Alpha

7.2.5 dappservices contract

- add usagex for LiquidX and other off chain service billing LiquidStorage, LiquidLens, LiquidAuth

7.3 2.0.2812

7.3.1 @liquidapps/dsp - 2.0.2812-latest

- separated pm2 log files
- add dsp version endpoint /v1/dsp/version
- add `keytype` parameter to `/v1/dsp/get_table_row` request (`"keytype":"symbol"` if passing a symbol or string as primary key, `"keytype":"number"` if passing number). The keytype field adds precision to ensure the correct primary key is returned and it is an optional parameter
- add support pass body to oracle POST request
- fixes
  - demux database sync issue
  - speed up demux sync and fix log messages
  - allow demux to sync from head_block in config.toml
  - prevent demux block processing from hanging
  - enable DSP API to use non-local nodeos instance
  - fix vram collision issue
  - auto generate dsp node index files
  - fix demux high CPU issue

7.3.2 @liquidapps/zeus-cmd - 2.0.2812

- add vcpu-dapp-service
- add chess game zeus unbox chess
- enable large LiquidAccount payload sizes
- add unit test for oracle POST request
- add `--phase` command to specify dapp services file `dapp-services-eos.js`, install npm files `npm`, or compile eos files `eos`
- fixes
  - change instantiateBuffer to instantiateSync for vcpu vrun.js
  - fix debian install for eosio.cdt due to syntax change in download link

7.3.3 @liquidapps/dapp-client - 2.0.2812

7.3.4 docs

- add unit testing section
- add email support: support@liquidapps.io
- add vCPU & LiquidChess
- add LiquidOracles, LiquidAccounts, LiquidScheduler docs

7.3.5 dappservices contract

7.4 2.0.2527

7.4.1 @liquidapps/dsp - 2.0.2527-latest

- add logging in /dsp/logs
- config.toml
  - Demux: head_block - can now set head block for demux to sync from
  - Demux: deprecated zmq_plugin support
  - Database: url - must set PostgressSQL database URL. Avoid duplicates, last processed block in db, etc.
  - Database: node_env - set to production to enable PostgressSQL database
- fixes
  - demux out-of-sync issue
  - de-duplication of requests and ability to resume dsp from last block
  - read past end of Buffer demux issue

7.4.2 @liquidapps/zeus-cmd - 2.0.2527

- updated to eosjs 20
- added eosjs1 compatibility wrapper
- enable migration to non-local eos chains
- LiquidAccounts - add nonce, chain_id, and expiry to transactions params
- fixes
  - Oracles - K out of N DSP results support. multi-dsp support fixes - adjust results size | code
  - Scheduler - added callback retries and better contract verification of timers. easier rescheduling of timers from callback (by returning 'true' in the function)
  - LiquidAccounts - fixed potential replay attack. added expiry, nonce and chainid verification in contract. Requires `xvinit` action to set chain_id for contract | code

7.4.3 @liquidapps/dapp-client - 2.0.2527

- get dappservices and dappairhodl1 tables
- push readfn and LiquidAccount transactions

7.4.4 docs

- Added IPFS info - bootstrap from existing node, swarm / bootstrap peers
- Added PostgressSQL Database info
- Updated EOS v1.8.4
- Updates IPFS v0.4.22
- Add cleanup and replay-contract information
- added support email: support@liquidapps.io

7.4.5 dappservices contract

- Enable/Disable Package - enablepkg, disablepkg
- 3rd party staking support
- search
- search