
LiquidApps

Aug 02, 2019

Contents

1	Developers	1
1.1	Getting Started	1
1.2	Overview	1
1.3	Zeus Getting Started	2
1.4	vRAM Getting Started	8
1.5	vRAM Getting Started - without zeus	14
1.6	Packages and Staking	17
1.7	Zeus Boxes	19
2	DSPs	21
2.1	Getting started	21
2.2	Overview	22
2.3	Architecture	22
2.4	Demux Backend	22
2.5	Account	23
2.6	EOSIO Node	24
2.7	IPFS	27
2.8	DSP Node	28
2.9	Packages	30
2.10	Testing	33
2.11	Claim Rewards	35
2.12	Upgrade DSP Node	35
3	Services	37
3.1	LiquidAuthenticator Service	37
3.2	LiquidScheduler Service	38
3.3	LiquidDNS Service	38
3.4	LiquidArchive Service	39
3.5	LiquidVRAM Service	40
3.6	LiquidLog Service	40
3.7	LiquidOracle Service	41
3.8	LiquidLens Service	42
3.9	LiquidLink Service	43
3.10	LiquidStorage Service	43
3.11	LiquidAccounts Service	44
4	DAPP Tokens	47

4.1	DAPP Token Overview	47
4.2	DAPP Tokens Tracks	47
4.3	Claiming DAPP Tokens	48
4.4	DAPP Tokens Distribution	48
4.5	Air-HODL	49
5	FAQs	51
5.1	Frequently Asked Questions The DAPP Token	51
5.2	Frequently Asked Questions DAPP Service Providers (DSPs)	53
5.3	Frequently Asked Questions vRAM	53

1.1 Getting Started

1.1.1 Overview

1.1.2 Packages and Staking

1.1.3 Zeus SDK

zeus-getting-started

1.1.4 vRAM

With Zeus

Without Zeus

1.2 Overview

1.2.1 Articles

- [vRAM guide for experts](#)
- [EOS dApps and Their RAM Expenses](#)

1.2.2 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

1.2.3 Have questions?

- [Join our Dev Telegram channel](#)
- [Join our Telegram channel](#)

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.

- [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 Hardware Requirements

- 16GB RAM
- 2 CPU Cores

1.3.4 Prerequisites

- nodejs == 10.x (nvm recommended, install at bottom of doc)
- curl
- cmake
- make

1.3.5 Recommended eosio.cdt and eosio versions

Automatically installed with `zeus unbox helloworld`

- eosio.cdt v1.6.1
- eosio v1.7.4

1.3.6 Install Zeus

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

1.3.7 Update

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

1.3.8 Test

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

1.3.9 Try out a game!

LiquidApps' take on Elemental Battles: <https://cardgame1112.dnsregistry1.com/> | 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
```

1.3.10 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, zmq/state-history plugin options included

DAPP Services Boxes

- [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

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

1.3.11 Zeus Options

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

Zeus compile

Compile a smart contract

```
zeus compile

# 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](#)

```
zeus import <CONTRACT_ACCOUNT_NAME> --owner-private-key <KEY> --active-private-key
↔<KEY>
zeus create contract-deployment CONTRACT_NAME CONTRACT_ACCOUNT_NAME
zeus migrate

# 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
# 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-all # do not compile contracts
--no-reset # do not reset local testing environment
```

Zeus test

Compile and unit test a smart contract

```
zeus test

# optional flags:

--compile-all # compile all contracts
# default: true
--wallet # keosd wallet to use
# default: zeus
--creator-key # contract creator key
# default: (eosio test key) 5KQwrPbwdL6PhXujxW37FSSQZlJiwsST4cqQzDeyXtP79zkvFD3
--creator # account to set contract to
# 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-all # do not compile contracts
--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

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)
```

(continues on next page)

(continued from previous page)

```
# default: development (local)
--password # password to decrypt the keypair
```

Help

```
zeus --help
```

List Boxes

```
zeus list-boxes
```

1.3.12 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",
    "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

```
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
        }
      }
    }
  }
};
```

1.3.13 Notes regarding permissions errors:

Recommend using Node Version Manager (nvm)

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash
# use install instructions provided to set PATH
nvm install 10
nvm use 10
```

1.4 vRAM Getting Started



1.4.1 Prerequisites

- Zeus
- Kylin Account
- If testing on Kylin: eosio v1.8.1

1.4.2 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
```

1.4.3 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
```

1.4.4 Add your contract logic

in contract/eos/mycontract/mycontract.cpp

```
#pragma once

#include "../dappservices/log.hpp"
#include "../dappservices/plist.hpp"
#include "../dappservices/plisttree.hpp"
#include "../dappservices/multi_index.hpp"

#define DAPPSERVICES_ACTIONS() \
  X SIGNAL_DAPPSERVICE_ACTION \
  LOG_DAPPSERVICE_ACTIONS \
  IPFS_DAPPSERVICE_ACTIONS

/** IPFS: (xcommit) (xcleanup) (xwarmup) | LOG: (xlogevent) (xlogclear) */
#define DAPPSERVICE_ACTIONS_COMMANDS() \
  IPFS_SVC_COMMANDS() LOG_SVC_COMMANDS()
```

(continues on next page)

(continued from previous page)

```

/** 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 */
    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))

```

1.4.5 Add your contract unit tests

in tests/mycontract.spec.js

```

import 'mocha';
require('babel-core/register');
require('babel-polyfill');
const { assert } = require('chai');
const { getCreateKeys } = require('../extensions/helpers/key-utils');
const { getNetwork } = require('../extensions/tools/eos/utis');
var Eos = require('eosjs');
const getDefaultArgs = require('../extensions/helpers/getDefaultArgs');
const artifacts = require('../extensions/tools/eos/artifacts');
const deployer = require('../extensions/tools/eos/deployer');
const { genAllocatedDAPPTokens } = 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`, () => {

```

(continues on next page)

(continued from previous page)

```

var testcontract;

/** SET CONTRACT NAME(S) */
const code = 'airairairail';
const code2 = 'testuser5';
var account = code;

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

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

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

      await genAllocateDAPPTokens(deployedContract, 'ipfs');
      var selectedNetwork = getNetwork(getDefaultArgs());
      var config = {
        expireInSeconds: 120,
        sign: true,
        chainId: selectedNetwork.chainId
      };
      if (account) {
        var keys = await getCreateKeys(account);
        config.keyProvider = keys.active.privateKey;
      }
      var eosvram = deployedContract.eos;
      config.httpEndpoint = 'http://localhost:13015';
      eosvram = new Eos(config);
      testcontract = await eosvram.contract(code);
      done();
    } catch (e) {
      done(e);
    }
  })();
});

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

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

      /** DEFAULT failed = false, 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`,

```

(continues on next page)

```
        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 ${symbol}`,
            memo: ''
        }, {
            authorization: `${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' ...
});
```


1.4.6 Compile and test

```
zeus test
```

1.4.7 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 "50.0000_
↪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
```

1.4.8 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
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\", \"50.0000 DAPP\"]" -p $KYLIN_TEST_ACCOUNT@active
```

1.4.9 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_ (continues on next page)
```

The result should look like:

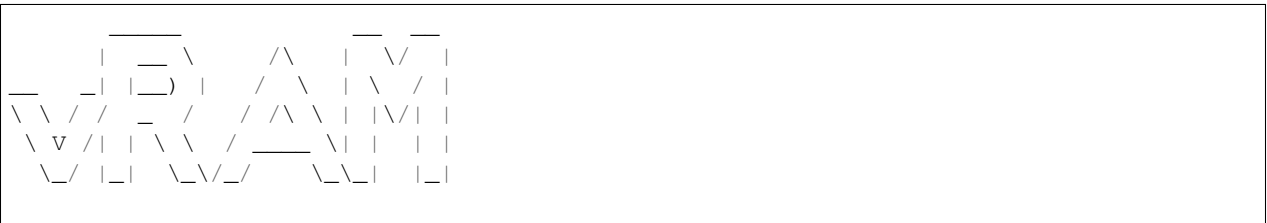
```
executed transaction:
↳865a3779b3623eab94aa2e2672b36dfec9627c2983c379717f5225e43ac2b74a 104 bytes 67049
↳us
# yourcontract <= yourcontract::youraction1 {"param1":"param1","param2":
↳"param2"}
>> {"version":"1.0","etype":"service_request","payer":"yourcontract","service":
↳"ipfsservice1","action":"commit","provider":"","data":"DH....."}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

1.4.10 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 $SKYLIN_TEST_ACCOUNT "accounts" $SKYLIN_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
```

1.5 vRAM Getting Started - without zeus



1.5.1 Hardware Requirements

1.5.2 Prerequisites

- eosio.cdt v1.6.1
- eosio v1.7.4
- If testing on Kylin: eosio v1.8.1
- Kylin Account

1.5.3 Install

Clone into your project directory:

```
git clone --single-branch --branch v1.4 --recursive https://github.com/liquidapps-io/
↳ dist
```

1.5.4 Modify your contract

vRAM provides a drop in replacement for the `eosio::multi_index` table that is also interacted with in the same way as the traditional `eosio::multi_index` table making it very easy and familiar to use. Please note that secondary indexes are not currently implemented for `dapp::multi_index` tables.

To access the vRAM table, add the following lines to your smart contract:

At header:

```
#include "../dist/contracts/eos/dappservices/multi_index.hpp"

#define DAPPSERVICES_ACTIONS() \
    X SIGNAL_DAPPSERVICE_ACTION \
    IPFS_DAPPSERVICE_ACTIONS

#define DAPPSERVICE_ACTIONS_COMMANDS() \
    IPFS_SVC_COMMANDS()

#define CONTRACT_NAME() mycontract
```

After contract class header

```
CONTRACT mycontract : public eosio::contract {
    using contract::contract;
public:

    /** ADD HERE */
    DAPPSERVICES_ACTIONS()
```

Replace `eosio::multi_index`

```
/** REPLACE */
typedef eosio::multi_index<"accounts"_n, account> accounts_t;

/** WITH */
typedef dapp::multi_index<"accounts"_n, account> accounts_t;

/** ADD (for client side query support): */
typedef eosio::multi_index<".accounts"_n, account> accounts_t_v_abi;
TABLE shardbucket {
    std::vector<char> shard_uri;
    uint64_t shard;
    uint64_t primary_key() const { return shard; }
```

(continues on next page)

(continued from previous page)

```
};  
typedef eosio::multi_index<"accounts"_n, shardbucket> accounts_t_abi;
```

Add DSP actions dispatcher

```
/** REPLACE */  
EOSIO_DISPATCH(mycontract, (youraction1) (youraction2) (youraction2))  
  
/** WITH */  
EOSIO_DISPATCH_SVC(mycontract, (youraction1) (youraction2) (youraction2))
```

1.5.5 Compile

```
eosio-cpp -abigen -o contract.wasm contract.cpp
```

1.5.6 Deploy Contract

```
export DSP_ENDPOINT=https://kylin-dsp-1.liquidapps.io  
export KYLIN_TEST_ACCOUNT=  
export KYLIN_TEST_PUBLIC_KEY=  
# 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 $KYLIN_TEST_ACCOUNT active "{\  
↪"threshold\:1,\keys\":[\"$KYLIN_TEST_PUBLIC_KEY\"],\"accounts\":[{\"permission\  
↪:{\"actor\":"eosio.code\", \"permission\":"active\"}, \"weight\":1}}" active -p  
↪$KYLIN_TEST_ACCOUNT@active
```

1.5.7 Select and stake DAPP for DSP package

DSP Package and staking

1.5.8 Test

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

The endpoint can be found in the [package table](#) of the `dappservices` account on all chains.

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

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....."}

```

1.5.9 Get table row

```

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

```

1.6 Packages and Staking

1.6.1 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:

- [Blocs.io](#)
- [EOS Nation](#)
- [Malta Block](#)
- [Mission Control](#)
- [Aloha EOS](#)
- [MinerGate](#)
- [DSP HQ](#)

1.6.2 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. conservices - enable CRON related tasks such as continuous staking
 3. oracleservic - provide oracle related services
 4. readfndpsvc - 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

1.6.3 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
```

1.6.4 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
```

1.6.5 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
```

1.6.6 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
```

1.6.7 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
```

1.6.8 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
```

1.6.9 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
```

1.7 Zeus Boxes

1.7.1 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`

2.1 Getting started

2.1.1 Overview

Overview

Architecture

2.1.2 Prerequisites

Account

2.1.3 Deploy

EOSIO Node

IPFS Node

DSP Service Node

2.1.4 Configuration

Packages

Testing

2.1.5 Claiming Rewards

Claim

2.1.6 Upgrade Version

Upgrade

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](#)

2.2.4 Want more information?

- Read our [whitepaper](#) and subscribe to our [Medium](#) posts.

2.3 Architecture

- **EOS full node** - to run a DSP requires running a full EOS node. The EOS node is also configured with a backend storage mechanism, whether that be the `state_history_plugin` or the `zmq_plugin` from eosrio.
- **IPFS Cluster node** - locally hosts your vRAM related data in IPFS for fast response times.
- **DSP Node** - the DSP node is responsible for performing actions like `get_table_row` for vRAM related data and in the case of LiquidAccounts, parsing related trx's and sending them to the chain.

2.4 Demux Backend

In the `config.toml` file in the **DSP Node Setup** you can configure either the `state_history_plugin` or eosrio's `zmq_plugin`.

```
[demux]
backend = "state_history_plugin"
# zmq: "zmq_plugin" only if using nodeos with eosrio's version of the ZMQ plugin:
↪ https://github.com/eosrio/eos\_zmq\_plugin
```

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 existing EOS Account

- Getting started on eos mainnet

First EOS Account

Fiat:

- EOS Account Creator
- EOS Lynx

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.cryptokylin.io/create_account?${DSP_ACCOUNT} > keys.json
curl http://faucet.cryptokylin.io/get_token?${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

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=1.8.1
```

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
```

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
```

Mainnet

```
URL=$(wget --quiet "https://eosnode.tools/api/bundle" -O- | jq -r '.data.snapshot.s3')
P2P_FILE=https://eosnodes.privex.io/?config=1
GENESIS=https://raw.githubusercontent.com/CryptoLions/EOS-MainNet/master/genesis.json
CHAIN_STATE_SIZE=131072
cd $HOME/.local/share/eosio/nodeos/data
wget $URL -O - | tar xvz
SNAPFILE=`ls snapshots/*.bin | head -n 1 | xargs -n 1 basename`
mv snapshots/$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"
p2p-server-address = addr:8888
http-server-address = 0.0.0.0:8888
p2p-listen-endpoint = 0.0.0.0:9876
```

(continues on next page)

(continued from previous page)

```

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
network-version-match = 1
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
read-mode = head
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

```

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

```

Wait until the node fully syncs, then 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

```

(continues on next page)

(continued from previous page)

```
[Service]
User=$NODEOS_USER
ExecStart=$NODEOS_EXEC --disable-replay-opts
[Install]
WantedBy=multiuser.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

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=0.4.19
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=multiuser.target
EOF

systemctl start ipfs
systemctl enable ipfs

exit
```

2.7.2 Cluster

IPFS-Cluster

[IPFS-Cluster Documentation](#)

Kubernetes

[IPFS Helm Chart](#)

2.8 DSP Node

2.8.1 Prerequisites

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
nvm install 10
nvm use 10
exit

```

Ubuntu/Debian

```

sudo apt install -y make cmake build-essential python

```

Centos/Fedora/AWS Linux:

```

sudo yum install -y make cmake3 python

```

2.8.2 Install

```

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

```

2.8.3 Configure Settings

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

```

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

```

2.8.4 Launch DSP Services

```

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

```

2.8.5 Check logs

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

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: committed 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!
```

2.9 Packages

2.9.1 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://..."
  }
}
```

(continues on next page)

(continued from previous page)

```
    "logo_svg": "https://...."
  },
  "location": {
    "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 removed 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=packagel
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
# 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
```

output should be:

```
registering package:packagel
✓package:packagel 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 modifiable. 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
```

2.10 Testing

2.10.1 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
```

(continues on next page)

(continued from previous page)

```
# 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
# stake to DSP package with DAPPHDL, only available on mainnet
cleos -u $DSP_ENDPOINT push action dappairhodll stake "{\"owner\":\"$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/dsp/ipfsservice/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>

2.11 Claim Rewards

2.11.1 Claim your DAPP daily rewards:

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

2.11.2 With Bloks.io:

Claim

2.12 Upgrade DSP Node

Ensure no new updates to the sample-config.toml file are present, if so, update your config.toml accordingly.

Link: [sample-config.toml](#)

```
sudo su -
systemctl stop dsp
systemctl stop ipfs
systemctl stop nodeos
# if changes to sample-config.toml syntax:
```

(continues on next page)

(continued from previous page)

```
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
setup-dsp
systemctl start nodeos
systemctl start ipfs
systemctl start dsp
exit
```

- search

3.1 LiquidAuthenticator Service

3.1.1 Overview

Authentication of offchain APIs and services using EOSIO permissions and contract

3.1.2 Stage

WIP

3.1.3 Contract

`authfndspsvc`

3.1.4 Box

`auth-dapp-service`

3.1.5 Service Commands

`authusage`

3.1.6 Tests

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

3.1.7 Implementation

3.2 LiquidScheduler Service

3.2.1 Overview

Scheduled Transactions

3.2.2 Stage

Alpha

3.2.3 Contract

`cronservices`

3.2.4 Box

`cron-dapp-service`

3.2.5 Service Commands

`schedule`

3.2.6 Tests

- `cron.spec.js`
- Consumer Contract Example

3.2.7 Implementation

3.3 LiquidDNS Service

3.3.1 Overview

DSP Hosted DNS Service

3.3.2 Stage

WIP

3.3.3 Contract

`dnsservices1`

3.3.4 Box

dns-dapp-service

3.3.5 Service Commands

dnsq

3.3.6 Tests

- dnsconsumer.spec.js
- Consumer Contract Example

3.3.7 Implementation

3.4 LiquidArchive Service

3.4.1 Overview

History API Provisioning

3.4.2 Stage

WIP

3.4.3 Contract

historyserve

3.4.4 Box

history-dapp-service

3.4.5 Service Commands

hststore

hsthld

hstserve

hstreg

3.4.6 Tests

- history.spec.js

3.4.7 Implementation

3.5 LiquidVRAM Service

3.5.1 Overview

Virtual Memory Service

3.5.2 Stage

Stable

3.5.3 Contract

```
ipfsservice1
```

3.5.4 Box

```
ipfs-dapp-service
```

3.5.5 Service Commands

commit

cleanup

warmup

3.5.6 Tests

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

3.5.7 Implementation

3.6 LiquidLog Service

3.6.1 Overview

Log Service

3.6.2 Stage

Beta

3.6.3 Contract

logservices1

3.6.4 Box

log-dapp-service

3.6.5 Service Commands

logevent

logclear

3.6.6 Tests

- logconsumer.spec.js
- Consumer Contract Example

3.6.7 Implementation

3.7 LiquidOracle Service

3.7.1 Overview

Web/IBC/XIBC Oracle Service

3.7.2 Stage

Beta

3.7.3 Contract

oracleservic

3.7.4 Box

oracle-dapp-service

3.7.5 Service Commands

geturi

orcclean

3.7.6 Tests

- oracleconsumer.spec.js
- Consumer Contract Example

3.7.7 Implementation

3.8 LiquidLens Service

3.8.1 Overview

Read Functions Service

3.8.2 Stage

Alpha

3.8.3 Contract

readfndspsvc

3.8.4 Box

readfn-dapp-service

3.8.5 Service Commands

rfnuse

3.8.6 Tests

- readfnconsumer.spec.js
- Consumer Contract Example

3.8.7 Implementation

3.9 LiquidLink Service

3.9.1 Overview

IBC MultiSig Service

3.9.2 Stage

Alpha

3.9.3 Contract

signfndspsvc

3.9.4 Box

sign-dapp-service

3.9.5 Service Commands

signtrx

sgcleanup

3.9.6 Tests

- sign.spec.js

3.9.7 Implementation

3.10 LiquidStorage Service

3.10.1 Overview

Distributed storage and hosting

3.10.2 Stage

WIP

3.10.3 Contract

liquidstorag

3.10.4 Box

storage-dapp-service

3.10.5 Service Commands

strstore

strhold

strserve

3.10.6 Tests

- storage.spec.js

3.10.7 Implementation

3.11 LiquidAccounts Service

3.11.1 Overview

Allows interaction with contract without a native EOS Account

3.11.2 Stage

Alpha

3.11.3 Contract

accountless1

3.11.4 Box

vaccounts-dapp-service

3.11.5 Service Commands

vexec

3.11.6 Tests

- vaccountsconsumer.spec.js
- Consumer Contract Example

3.11.7 Implementation

- search

4.1 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.

4.1.1 Videos

- [EOS Weekly - The LiquidApps Game-Changer](#)
- [EOS Weekly - Unlimited DSP Possibilities](#)

4.1.2 Have questions?

- [Join our Telegram channel](#)

4.1.3 Want more information?

- Read our [whitepaper](#) and subscribe to our [Medium](#) posts.

4.2 DAPP Tokens Tracks

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

4.2.1 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).

4.2.2 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.

4.3 Claiming DAPP Tokens

4.3.1 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.

4.3.2 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”.

4.4 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.

4.4.1 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](#)

4.5 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](#)

5.1 Frequently Asked Questions The DAPP Token

- *What is the DAPP token?*
- *What is the supply schedule of DAPP token?*
- *How are DAPP tokens distributed?*
- *Why do you need to use DAPP Token and not just EOS?*
- *Why is the sale cycle 18 hours?*
- *What is an airHODL?*
- *Is this an EOS fork?*

5.1.1 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.

5.1.2 What is the supply schedule of DAPP token?

DAPP will have an initial 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%.

5.1.3 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.

5.1.4 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.

5.1.5 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.

5.1.6 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.

5.1.7 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.

5.2 Frequently Asked Questions DAPP Service Providers (DSPs)

- *What is a DSP?*
- *Who can be a DSP?*
- *Are DSPs required to run a full node?*
- *How are DSPs incentivized?*

5.2.1 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.

5.2.2 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.

5.2.3 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.

5.2.4 How are DSPs incentivized?

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

5.3 Frequently Asked Questions vRAM

- *Why do I need vRAM?*
- *How is vRAM different from RAM?*
- *How can we be sure that data cached with DSPs is not tampered with?*
- *How much does vRAM cost?*

5.3.1 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.

5.3.2 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, BitTorrent, 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.

5.3.3 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

5.3.4 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
- search