LiquidApps

May 24, 2019

Contents

CHAPTER 1

Developers

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

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".

1.3.2 Boxes

- EOSIO dApp development support
- DAPP Services support

1.3.3 Hardware Requirements

1.3.4 Prerequisites

- nodejs == 10.x (nvm recommended)
- curl

Recommended (otherwise falling back to docker)

- eosio.cdt v1.6.1
- eosio v1.7.2

1.3.5 Install Zeus

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

Notes regarding docker on mac:

Recommended version: 18.06.1-ce-mac73

1.3.6 Upgrade

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

1.3.7 Test

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

1.3.8 Samples Boxes

vRAM

- coldtoken - vRAM based eosio.token
- deepfreeze - vRAM based cold storage contract
- vgrab - vRAM based airgrab for eosio.token
- cardgame - vRAM supported elemental battles
- registry - Generic Registry - the1registry

Zeus Extensions

- contract-migrations-extensions
- build-extensions
- test-extensions
- eos-extensions
- unbox-extensions
- demux

DAPP Services

- ipfs-dapp-service
- log-dapp-service
- cron-dapp-service
- oracle-dapp-service

Misc.

- microauctions - Micro Auctions
- eos-detective-reports - EOS Detective Reports - by EOSNation
- helloworld - Hello World
- token - Standard eosio.token

1.3.9 Other Options

```
zeus compile #compile contracts
zeus migrate #migrate contracts (deploy to local eos.node)
```

1.3.10 Usage inside a project

```
zeus --help
```

List Boxes

```
zeus list-boxes
```

1.3.11 Project structure

Directory structure

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

zeus-box.json

```
    {
      "ignore": [
        "README.md"
      ],
      "commands": {
        "Compile contracts": "zeus compile",
        "Migrate contracts": "zeus migrate",
        "Test contracts": "zeus test"
      },
      "install":{
          "npm": {

          }
      },
      "hooks": {
        "post-unpack": "echo hello"
      }
    }
```

zeus-config.js

```
    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: "localhost",
                        port: 7545,
                        network_id: "*", // Match any network id
                        secured: false
                    },
                    mainnet:{
                        host: "localhost",
                        port: 7545,
                        network_id: "*", // Match any network id
                        secured: false
                    }
                }
            }
        }
    };
```

1.3.12 Notes regarding permissions errors:

Recommend using Node Version Manager (nvm)

```
sudo apt install curl
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash
exec bash
nvm install 10
nvm use 10
```

Or you can try the following:

```
sudo groupadd docker
sudo usermod -aG docker $USER

#If still getting error:
sudo chmod 666 /var/run/docker.sock
```

1.4 vRAM Getting Started

```
      ______                __  __
     |  __ \      /\       |  \/  |
__   _| |__) |    /  \      | \  / |
\ \ / /  _  /    / /\ \     | |\/| |
 \ V /| | \ \   / ____ \    | |  | |
  \_/ |_|  \_\/_/    \_\_\   |_|  |_|
```

1.4.1 Prerequisites

- Zeus
- Kylin Account

1.4.2 Unbox template

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

1.4.3 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() \
  XSIGNAL_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()

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

using std::string;

CONTRACT_START()
  public:

  /*** YOUR LOGIC ***/
```

(continues on next page)

(continued from previous page)

```
  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))
```

1.4.4 Add your contract test

in tests/mycontract.spec.js

```
import 'mocha';
require('babel-core/register');
require('babel-polyfill');
const { assert } = require('chai');
const { getNetwork, getCreateKeys } = require('../extensions/tools/eos/utils');
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 { genAllocateDAPPTokens } = 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 = 'airairairai1';
  const code2 = 'testuser5';
  var account = code;

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

        /*** DEPLOY CONTRACT ***/
```

(continues on next page)

(continued from previous page)

```
        var deployedContract = await deployer.deploy(ctrt, code);

        /*** DEPLOY ADDITIONAL CONTRACTS ***/
        var deployedContract2 = await deployer.deploy(ctrt, 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.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`,
          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: ''
```

(continues on next page)

(continued from previous page)

```
        }, {
          authorization: `${code2}@active`,
          broadcast: true,
          keyProvider: [key.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.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);
      }
    })();
  });
});
```

1.4.5 Compile and test

```
zeus test
```

1.4.6 Deploy Contract

```
export EOS_ENDPOINT=https://kylin-dsp-1.liquidapps.io
# Buy RAM:
cleos -u $EOS_ENDPOINT system buyram $KYLIN_TEST_ACCOUNT $KYLIN_TEST_ACCOUNT "50.0000␣
↪EOS" -p $KYLIN_TEST_ACCOUNT@active
# Set contract code and abi
cleos -u $EOS_ENDPOINT set contract $KYLIN_TEST_ACCOUNT ../contract -p $KYLIN_TEST_
↪ACCOUNT@active
```

(continues on next page)

(continued from previous page)

```
# Set contract permissions
cleos -u $EOS_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
```

```
# TBD:
#   zeus import key $KYLIN_TEST_ACCOUNT $KYLIN_TEST_PRIVATE_KEY
#   zeus create contract-deployment contractcode $KYLIN_TEST_ACCOUNT
#   zeus migrate --network=kylin
```

1.4.7 Select and stake DAPP for DSP package

```
export PROVIDER=uuddlrlrbass
export PACKAGE_ID=package1
export MY_ACCOUNT=$KYLIN_TEST_ACCOUNT

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

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

DSP Package and staking

1.4.8 Test

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

```
cleos -u $EOS_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 vRAM Getting Started - without zeus

```
       ______                __  __
      |  __ \      /\       |  \/  |
__    _| |__) |    /  \      | \  / |
\ \  / /  _  /    / /\ \     | |\/| |
 \ V /| | \ \   / ____ \|  |  | |
  \_/ |_|  \_\/_/    \_\_|  |_|
```

1.5.1 Hardware Requirements

1.5.2 Prerequisites

- eosio.cdt v1.6.1
- eosio v1.7.1
- Kylin Account

1.5.3 Install

Clone into your project directory:

```
git clone  --single-branch --branch v1.2 --recursive https://github.com/liquidapps-io/
↪dist
```

1.5.4 Modify your contract

vRAM provides a drop in replacement for the multi_index table that is also interacted with in the same way as the traditional 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() \
    XSIGNAL_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; }
      };
      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

```
# Buy RAM:
cleos -u $EOS_ENDPOINT system buyram $KYLIN_TEST_ACCOUNT $KYLIN_TEST_ACCOUNT "50.0000␣
↪EOS" -p $KYLIN_TEST_ACCOUNT@active
# Set contract code and abi
cleos -u $EOS_ENDPOINT set contract $KYLIN_TEST_ACCOUNT ../contract -p $KYLIN_TEST_
↪ACCOUNT@active

# Set contract permissions
cleos -u $EOS_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.

```
export EOS_ENDPOINT=https://dspendpoint
cleos -u $EOS_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......"}
```

1.6 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.

1.6.2 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 $EOS_ENDPOINT push action dappservices selectpkg "[\"$MY_ACCOUNT\",\"
↪$PROVIDER\",\"$SERVICE\",\"$PACKAGE_ID\"]" -p $MY_ACCOUNT@active
```

1.6.3 Stake DAPP Tokens for DSP Package

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

- 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

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

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

2.4 Storage Backend

2.5 Demux Backend

2.6 Account

2.6.1 Prerequisites

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

2.6.2 Account Name

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

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

Save wallet_password.pwd somewhere safe!

2.6.3 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

Bitcoin/ETH/Bitcoin Cash/ALFAcoins:

- ZEOS

Kylin

Create an account

```
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!

2.6.4 Import account

```
cleos wallet import $DSP_PRIVATE_KEY
```

2.7 EOSIO Node

2.7.1 Hardware Requirements

2.7.2 Prerequisites

- jq
- wget
- curl

2.7.3 Get EOSIO binary

```
# install nodeos
VERSION=1.7.2
```

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
```

2.7.4 Install

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

2.7.5 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=https://s3-ap-northeast-1.amazonaws.com/eosbeijing/snapshot-
↪0276f607955f3008bae69fc47a23ac2eb989af1adebeced2d7462ef30423b194.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=65535
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
```

2.7.6 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
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
```

(continues on next page)

(continued from previous page)

```
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
plugin = eosio::producer_plugin
plugin = eosio::chain_plugin
plugin = eosio::chain_api_plugin
plugin = eosio::net_plugin
plugin = eosio::state_history_plugin
trace-history = true
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
```

2.7.7 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.

2.7.8 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=multiuser.target
EOF

systemctl start nodeos
systemctl enable nodeos
exit
```

(continues on next page)

(continued from previous page)

```
sleep 3
systemctl status nodeos
```

2.7.9 Optimizations

- atticlab - cpu performance presentation

2.8 IPFS

2.8.1 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
[Install]
WantedBy=multiuser.target
EOF

systemctl start ipfs
systemctl enable ipfs

exit
```

2.8.2 Cluster

IPFS-Cluster

IPFS-Cluster Documentation

Kubernetes

IPFS Helm Chart

2.9 DSP Node

2.9.1 Hardware Requirements

2.9.2 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
```

2.9.3 Install

```
sudo su -
nvm use 10
npm install -g pm2
npm install -g @liquidapps/dsp --unsafe-perm=true
exit
```

2.9.4 Configuration

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

And fill in the following details:

Demux Backend

DEMUX_BACKEND

- state_history_plugin
- zmq_plugin - only if using nodeos with eosrio's version of the ZMQ plugin: https://github.com/eosrio/eos_zmq_plugin

IPFS Cluster

IPFS_HOST - ipfs hostname IPFS_PORT (5001) - ipfs port IPFS_PROTOCOL (http) - ipfs protocol

hostname, port and protocol of IPFS Cluster

DSP Account

DSP_ACCOUNT and DSP_PRIVATE_KEY - Account and private key of Generated DSP Account

nodeos ENVS

EOS Node Settings

NODEOS_HOST - nodeos hostname

NODEOS_PORT (8888) - nodeos port

NODEOS_ZMQ_PORT (5557) - if using zmq_plugin

NODEOS_WEBSOCKET_PORT (8887) - if using state_history_plugin

NODEOS_CHAINID:

- mainnet chainID: aca376f206b8fc25a6ed44dbdc66547c36c6c33e3a119ffbeaef943642f0e906
- kylin chainID: 5fff1dae8dc8e2fc4d5b23b2c7665c97f9e9d8edf2b6485a86ba311c25639191

2.9.5 Check logs

```
sudo pm2 logs
```

output should look like:

```
1|demux    | demux listening on port 3195!
1|demux    | ws connected
1|demux    | got abi
0|dapp-services-node  | services listening on port 3115!
0|dapp-services-node  | service node webhook listening on port 8812!
2|ipfs-dapp-service-node  | ipfs listening on port 13115!
2|ipfs-dapp-service-node  | commited to: ipfs://
↪zb2rhmy65F3REf8SZp7De11gxtECBGgUKaLdiDj7MCGCHxbDW
2|ipfs-dapp-service-node  | ipfs connection established
```

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": {
      "name": "Atlantis",
      "country": "ATL",
      "latitude": 2.082652,
```

(continues on next page)

(continued from previous page)

```
      "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
        }
    ]
}
```

If not using Kubernetes

```
npm install -g @liquidapps/zeus-cmd
cd $(readlink -f `which setup-dsp` | xargs dirname)
```

Register Package

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

```
export PACKAGE_ID=package1
export EOS_CHAIN=mainnet
#or
export EOS_CHAIN=kylin

export DSP_ENDPOINT=https://acme-dsp.com
zeus register dapp-service-provider-package \
    ipfs $DSP_ACCOUNT $PACKAGE_ID \
    --key $DSP_PRIVATE_KEY \
    --min-stake-quantity "10.0000" \
    --package-period 86400 \
    --quota "1.0000" \
    --network $EOS_CHAIN \
    --api-endpoint $DSP_ENDPOINT \
    --package-json-uri https://acme-dsp.com/package1.dsp-package.json
```

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

Using cleos:

```
cleos -u $EOS_ENDPOINT push action dappservices modifypkg "[\"$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

Run a sample contract using your DSP:

On a remote machine, follow The *vRAM Getting Started Tutorial*

Check logs on your DSP Node

```
pm2 logs
```

in kubernetes:

```
kubectl logs dsp-dspnode-0 -c dspnode-ipfs-svc
```

Look for "xcommit" and "xcleanup" actions for your contract:

mycoldtoken1 at bloks.io

2.12 Claim Rewards

2.12.1 Claim your DAPP daily rewards:

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

2.13 Upgrade DSP Node

2.13.1 Upgrade NPM Package

```
sudo su -
nvm use 10
npm update -g @liquidapps/dsp --unsafe-perm=true
setup-dsp
systemctl stop dsp
systemctl start dsp
exit
```

2.13.2 Check logs

```
sudo pm2 logs
```

output should look like:

```
1|demux    | demux listening on port 3195!
1|demux    | ws connected
1|demux    | got abi
0|dapp-services-node  | services listening on port 3115!
0|dapp-services-node  | service node webhook listening on port 8812!
2|ipfs-dapp-service-node  | ipfs listening on port 13115!
2|ipfs-dapp-service-node  | commited to: ipfs://
↪zb2rhmy65F3REf8SZp7De11gxtECBGgUKaLdiDj7MCGCHxbDW
2|ipfs-dapp-service-node  | ipfs connection established
```

- search

CHAPTER 3

Services

3.1 IPFS Service

3.1.1 Overview

Articles

- vRAM guide for experts

Videos

- Developer Explains - Decentralized Dapp Scaling w/ IPFS! How LiquidApps Dapp Service Providers Work

3.1.2 Contract

`ipfsservice1`

3.1.3 Contract Libraries

Zeus Boxes

- ipfs-dapp-service

Raw IPFS Access

```
// TBD
```

vRAM Multi-Index Table

vRAM Getting Started

```
// TBD
```

Shards and buckets

```
// TBD
```

Indexes

vRAM graph

```
// TBD
```

3.1.4 Cache Strategies

On Demand

Delayed

Manual

Session Based

TBD

LRU

TBD

MRU

TBD

3.1.5 Batched and Nested Warmups

TBD

3.1.6 Tools

Garbage Collection

Garbage Collection

Recovery

Recovery

3.2 Log Service

3.2.1 Overview

3.2.2 Contract

`logservices1`

3.3 Cron Service

3.3.1 Overview

3.3.2 Contract

`cronservices`

3.3.3 Contract Libraries

Zeus Boxes

- cron-dapp-service

3.4 Oracles Service

3.4.1 Overview

Request Protocols

Web

```
- http
- https
- https+post
- http+post
- wolfram_alpha
```

Blockchains

```
- self_history
- sister_chain_history
- sister_chain_block
```

Other

```
- random
```

3.4.2 Contract

`oracleservic`

3.4.3 Contract Libraries

Zeus Boxes

- oracle-dapp-service

3.5 vAccounts Service

3.5.1 Overview

Articles

All Aboard The EOS Train

3.5.2 Contract

`accountless1`

3.5.3 Contract Libraries

Zeus Boxes

- vaccounts-dapp-service

3.6 Donation Service

3.6.1 Overview

3.6.2 Contract

`donationssvc`

- search

CHAPTER 4

DAPP Tokens

4.1 DAPP Token Overview

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

4.2.1 Instant Track

4.2.2 Regular Track

4.3 Claiming DAPP Tokens

4.3.1 Automatic

4.3.2 Manual

4.4 DAPP Tokens Distribution

4.5 AirHODL

- search

CHAPTER 5

FAQs

5.1 Frequently Asked Questions The DAPP Token

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 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%.

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)

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

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, 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.

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