# ICON Developers Meetup

## ICON foundation

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# ICON Developers Meetup

Step 1. ICON Basics

- 1.1. Introduction to ICON
- 1.2. Architecture and Components

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

- 2.1. Account & Transaction
- 2.2. Test Environment

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

- 3.1. Smart Contract Basics
- 3.2. Introduction to T-Bears
- 3.3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# 1. ICON Basics

# ICON Developers Meetup

Step 1. ICON Basics

- 1.1. Introduction to ICON
- 1.2. Architecture and Components

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# 1.1. Introduction to ICON

# ICON Developers Meetup

Step 1. ICON Basics

- 1.1. Introduction to ICON
- 1.2. Architecture and Components

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# Blockchain - Beginning of new Era

- Blockchain

    - Centralized ⇒ Decentralized world

- Problem

    - Low TPS (Transaction Per Second)

    - Limited scalability

    - Isolated blockchains

# ICON - New Generation Blockchain

1st Generation : Bitcoin and the simple alt-coins (eg. litecoin) [2009]

Programmable Smart Contract

2nd Generation : Ethereum. Turing complete Smart Contract [2015]

Performance Enhancement

3rd Generation : EOS, AION, Zilliqa, ICON [2017] …

# ICON Characteristic

- Delegated proof of contribution (DPoC)

  - One confirmation

  - 1000+ TPS

- Multi channel

  - 1000+ TPS per channel

- Low / flexible transaction fee

- Interchain

- Native python code Smart Contract + JVM

# Summary

- 3rd Generation Blockchain
    - Programmable Smart Contract
    - Performance Enhancement

- ICON Characteristic
    - DPoC
    - Interchain
    - Native python code Smart Contract + JVM
    - Low / flexible transaction fee
    - Multi channel

# 1.2. Architecture and Components

# ICON Developers Meetup

Step 1. ICON Basics

- 1.1. Introduction to ICON
- 1.2. Architecture and Components

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources
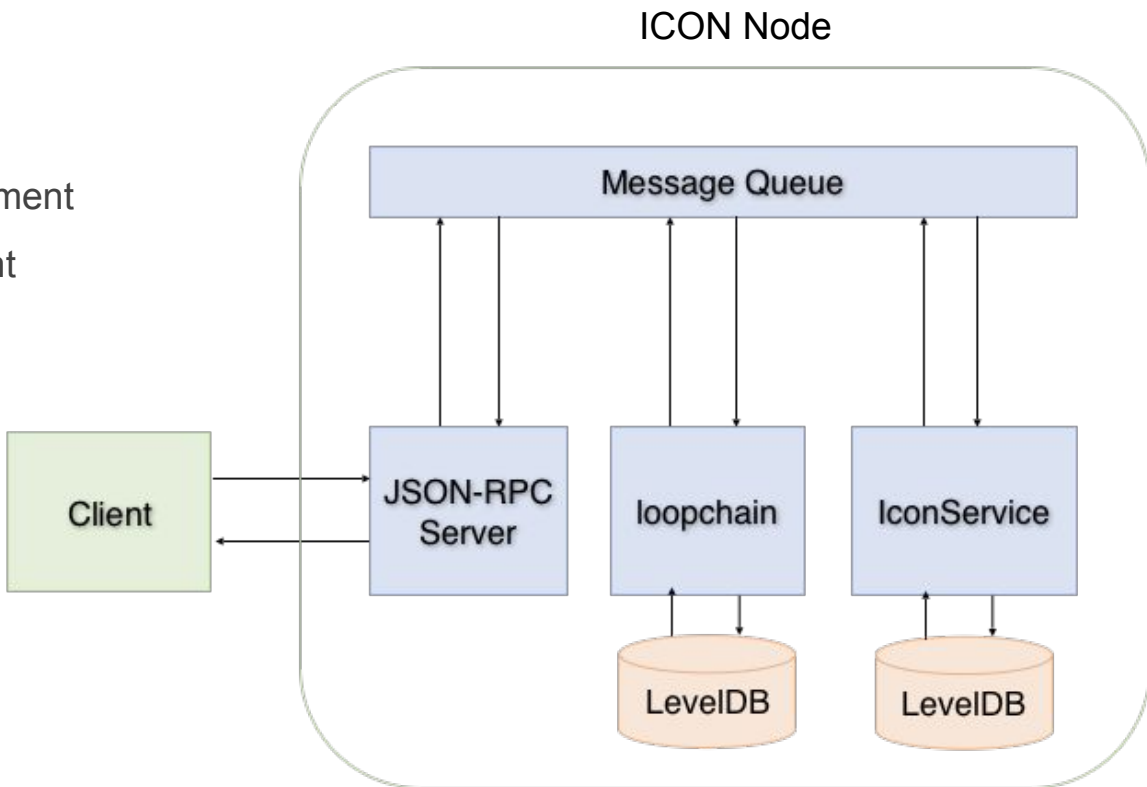
# ICON Network

- Representatives
- Citizen
- Light Client
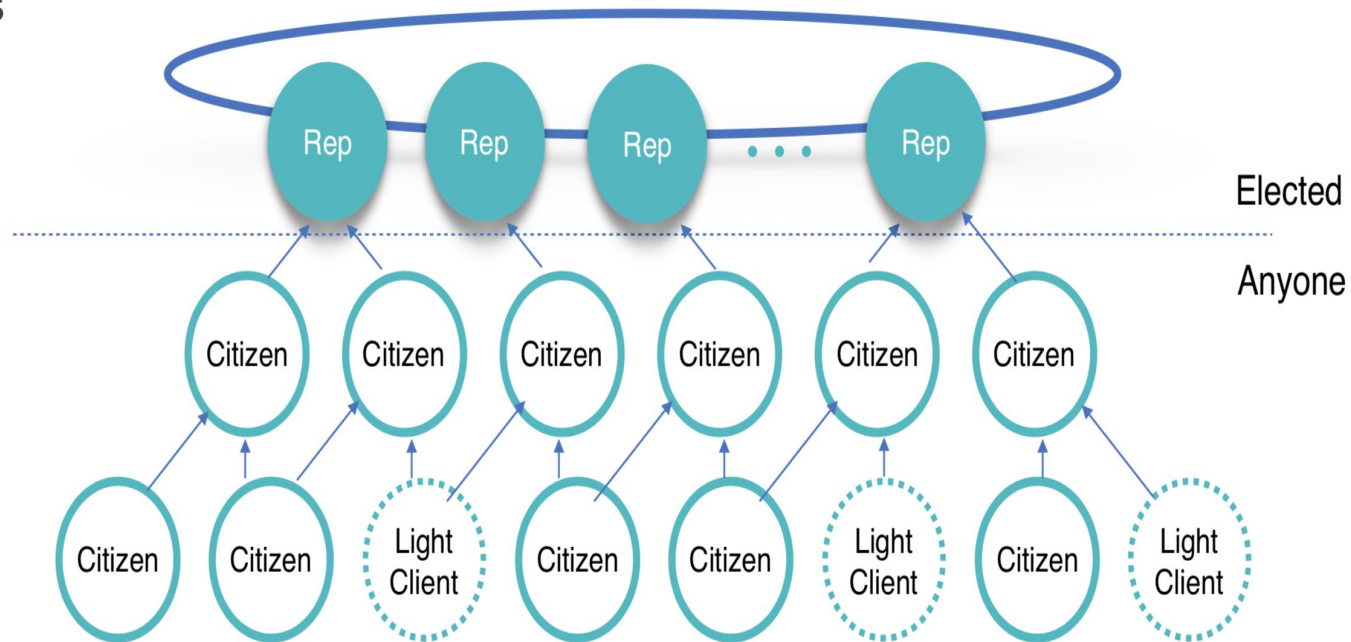
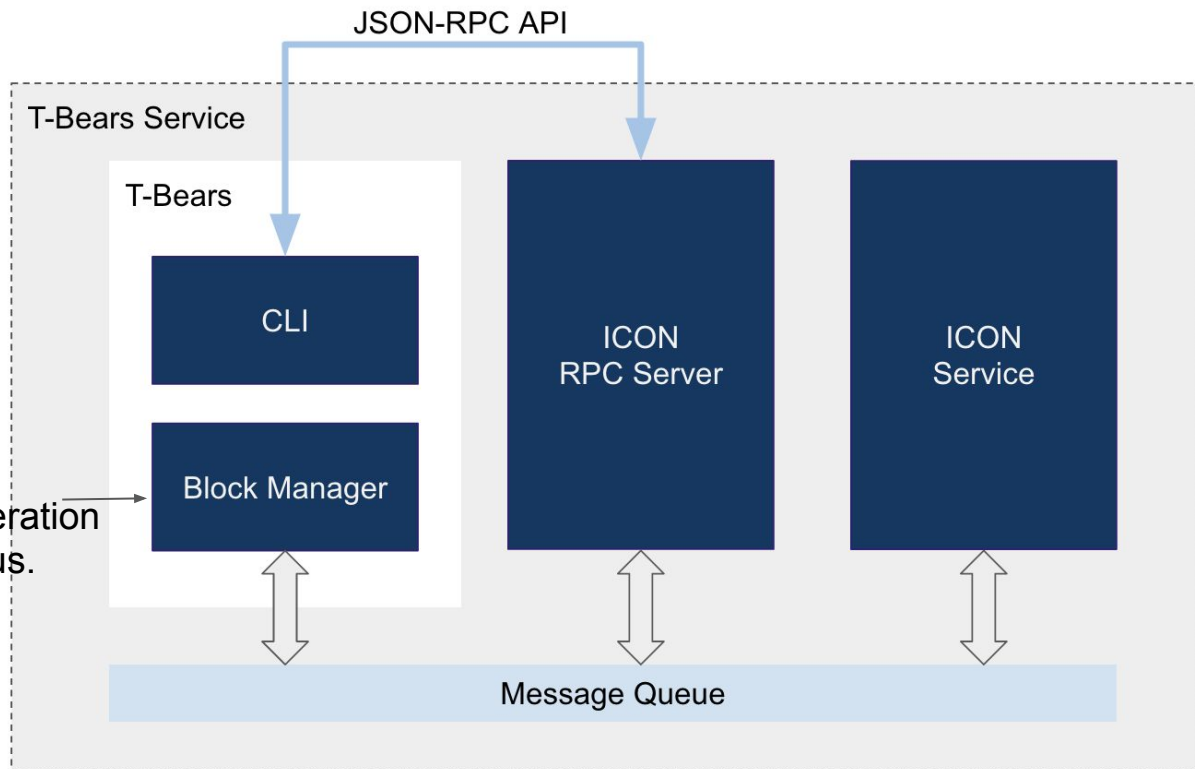# Architecture and Components

- T-Bears

- SDK

- Wallet

- Tracker
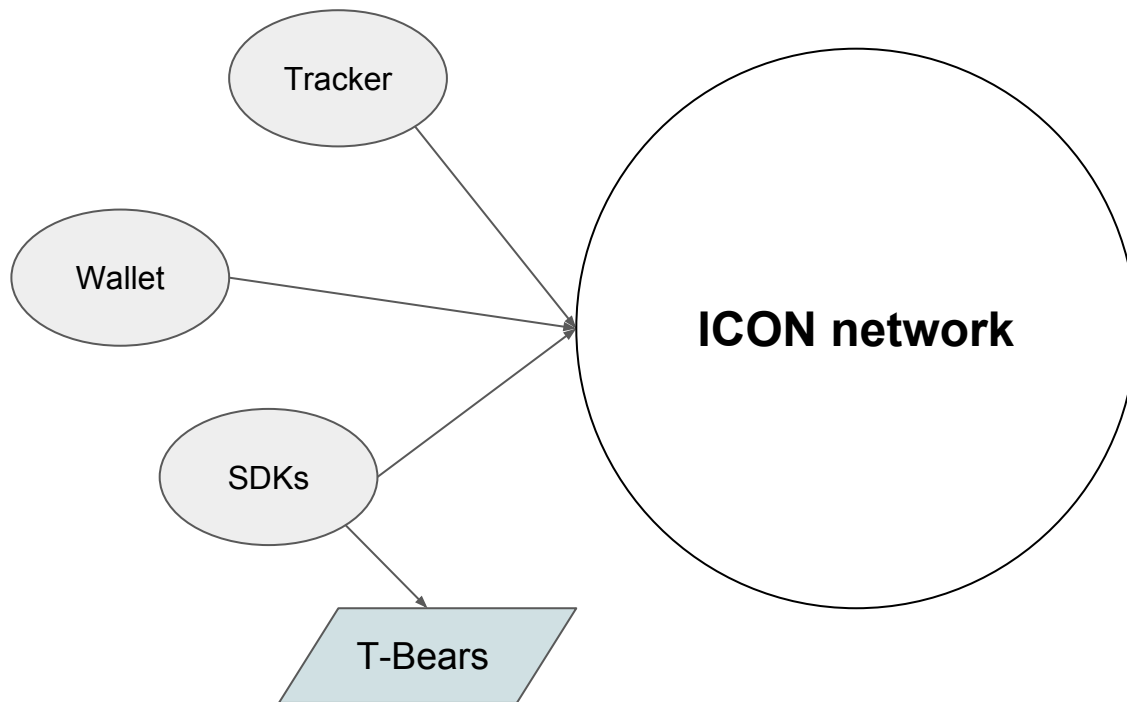
# Node View

- IconService

    - SCORE execution environment

    - ICX base coin management

    - Transaction fee calculation

- Loopchain

    - Peer management

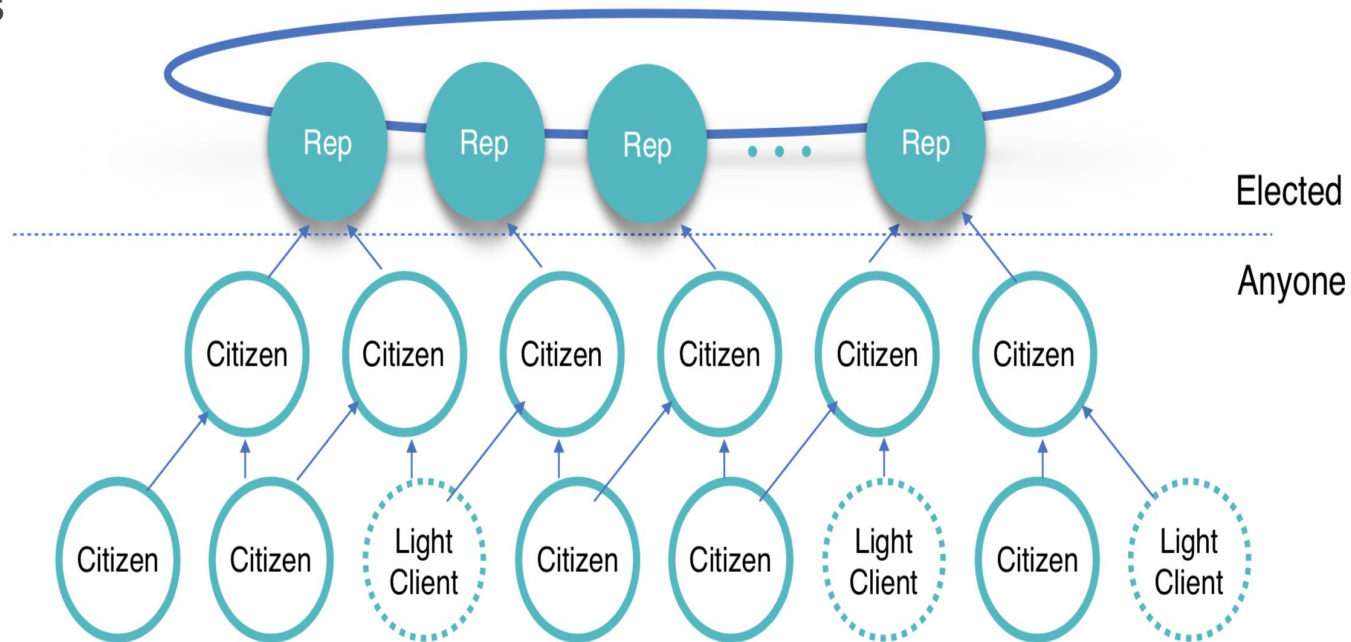    - Block management

    - LFT consensus engine

ICON Node

# T-Bears

# Summary

- Representatives
- Citizen
- Light Client

- T-Bears
- SDK
- Wallet
- Tracker

# 2. Interacting with ICON nodes

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

- 2.1. Account & Transaction
- 2.2. Test Environment

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# 2.1. Account & Transaction

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

- 2.1. Account & Transaction
- 2.2. Test Environment

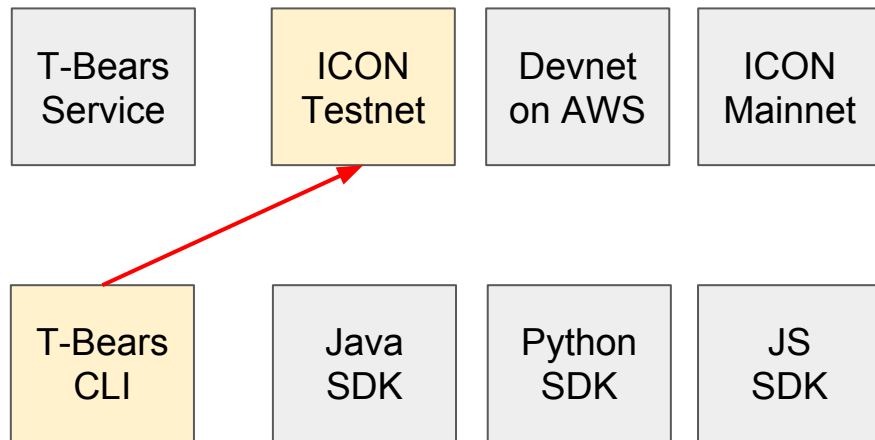Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# Account

- Account : Address, Balance ...

- Address
  - Unique Identifier of Account

  - PREFIX
    - hx : **EOA(External Owned Account)**
      - ex) hxe7af5fcfd8dfc67530a01a0e403882687528dfcb
    - cx : **CA(Contract Account)**
      - ex) cx0000000000000000000000000000000000000000

# Transaction

### Transfer

- Used when transferring ICX.

### Message

- Used when transferring a message, and HEX string `data`.

### Call

- Used when calling a function in SCORE, with `data which` has dictionary value.

### Deploy

- Used when installing or updating a SCORE, with `data` which has dictionary value.

# Summary

- Address
    - hx for EOA, cx for CA


- Transactions
    - call, deploy, transfer, message

# 2.2. Test Environment

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

- 2.1. Account & Transaction
- 2.2. Test Environment

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# Test Environment

- T-Bears emulated environment
  - Transaction fee : off
  - SCORE audit : off

- ICON Testnet
  - Transaction fee : on
  - SCORE audit : off

- Private Devnet on AWS
  - Transaction fee : off
  - SCORE audit : off

- https://icon-project.github.io/docs/icon_network.html

| T-Bears Service | ICON Testnet | Devnet on AWS | ICON Mainnet |
|---|---|---|---|
| T-Bears CLI | Java SDK | Python SDK | JS SDK |

# Testnet : Create an Account

- $ tbears keystore [file_path]

- Examine the keystore file

```
{
    "address": "hxe7af5fcfd8dfc67530a01a0e403882687528dfcb",
    "crypto": {
        ....
    },
    "id": "e2ca66c6-b8de-4413-82cb-52c2a2200b8d",
    "version": 3,
    "coinType": "icx"
}
```

# Testnet : Account & Balance

- To receive test ICX, send email to testicx@icon.foundation with following information

  - Testnet node url

  - Address to receive the testnet ICX

  - Faucet : http://52.88.70.222

| Name | Yeouido (여의도) |
|---|---|
| Node | **https://bicon.net.solidwallet.io** |
| API endpoint | https://bicon.net.solidwallet.io/api/v3 |
| Network ID (nid) | 3 |
| Tracker | https://bicon.tracker.solidwallet.io |
| Transaction fee | on |
| SCORE audit | off |

32

# Testnet : Send Queries

- $ tbears **balance [address]** -u https://bicon.net.solidwallet.io/api/v3

- $ tbears totalsupply -u https://bicon.net.solidwallet.io/api/v3

- $ tbears lastblock -u https://bicon.net.solidwallet.io/api/v3

- $ tbears blockbyheight 0x1 -u https://bicon.net.solidwallet.io/api/v3

# ICON Dev Tools

- T-Bears

  - Guide [https://github.com/icon-project/t-bears]


- SDK

  - Java [https://github.com/icon-project/icon-sdk-java]

  - Python [https://github.com/icon-project/icon-sdk-python]

  - Javascript [https://github.com/icon-project/icon-sdk-js]

# Summary

- Test Environment

  1. T-Bears emulated environment
  2. ICON Testnet
  3. Private Devnet on AWS

- Tools

  1. T-Bears
  2. SDK(Java, Python, Javascript)

# 3. Smart Contract Development

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

- 3.1. Smart Contract Basics
- 3.2. Introduction to T-Bears
- 3.3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# 3.1. Smart Contract Basics

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

- 3.1. Smart Contract Basics
- 3.2. Introduction to T-Bears
- 3.3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# Smart Contract Basics

- What is smart contract?


- SCORE(Smart Contract On Reliable Environment)

    - What is SCORE?

        - Written in Python (Easy to learn)

        - State DB abstractions (VarDB, DictDB, ArrayDB)

        - Sandbox policy

# SCORE Implementation Guide

- SCORE
    - Sandbox policy
        - Should be deterministic
        - No random Operation
        - No outgoing network call
        - No system call (e.g. file system access, clock time)
        - No long-running operation inside the SCORE

    - Do not import python packages other than iconservice

# SCORE Model

- Finite state machine

- State transition by transaction

- Must inherit IconScoreBase



IconScoreBase

```
SCORE

__init__(db)
{
      super().__init__(db)
      …
}
on_install()
on_update()
```

ArrayDB, DictDB, VarDB

State DB
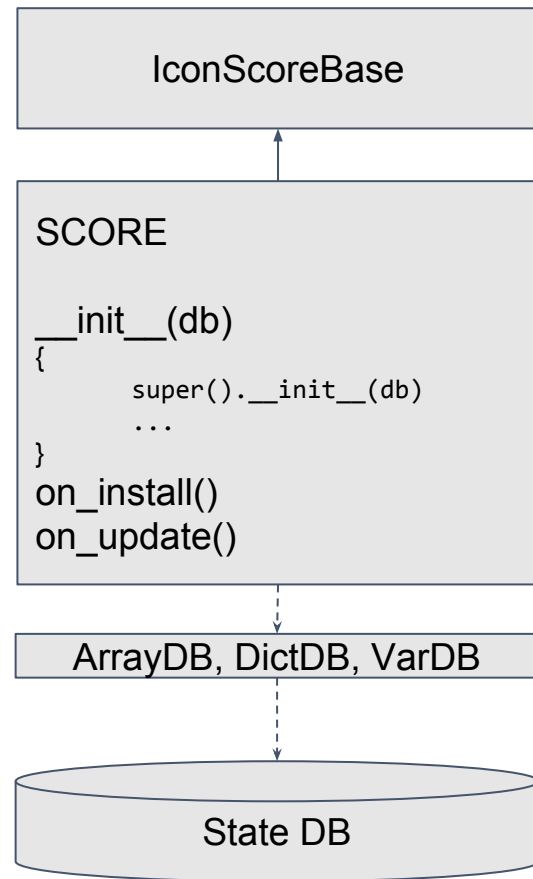
# IconScoreBase - Methods

- __init__
  - Called when the contract is loaded at each nodes.

- on_install
  - Called when the SCORE is deployed

- on_update
  - Called when the SCORE is updated

- fallback
  - Reserved function executed whenever the SCORE receives plain ICX coins without data
  - Without @`payable`, icx coin transfers to the contract will fail

IconScoreBase

```
SCORE

__init__(db)
{
    super().__init__(db)
    ...
}
on_install()
{
    super().on_install()
    ...
}

on_update()
{
    super().on_update()
    ...
}
```
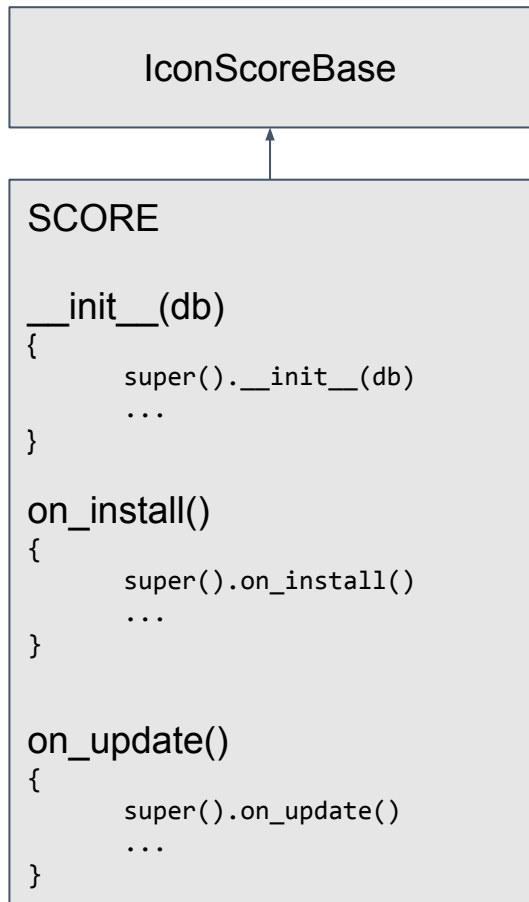
# IconScoreBase - Properties

- msg : Holds information of the account who called the SCORE

    - msg.sender : Address of the account or contract who called this function

    - msg.value : Amount of icx that the sender attempts to transfer to the current SCORE

- tx : Transaction info

    - tx.origin : The account who created the transaction

    - tx.index : Transaction index

    - tx.hash : Transaction hash

    - tx.timestamp : Transaction creation time

    - tx.nonce : (Optional) random value

# IconScoreBase - Properties

- icx : An object used to transfer icx coin

    - icx.transfer : Transfer designated amount of icx coin to addr_to. Return True or Exception can be occured

    - icx.send : Sends designated amount of icx coin to addr_to. Return True or False

- db : db instance used to access state DB

- address : SCORE address

- owner : Address of the account who deployed the contract

- block_height : Current block height

# Decorators

- @external
    - Can be called from outside the SCORE

- @payable
    - Permitted to receive incoming ICX coins

- @eventlog
    - Include logs in its txresult as 'eventLogs'

# Utility Functions

-   revert : Developer can force a revert exception. If the exception is thrown, all the changes in the state DB in current transaction will be rolled back.

-   sha3_256 : Computes hash using the input data

-   json_dumps : Converts a python object to a JSON string

-   json_loads : Parses a JSON string and converts to a python object

# State DB

- VarDB

  - Simple key-value state

- DictDB

  - Behaves more like python dict. DictDB does not maintain order.

- ArrayDB

  - Supports one dimensional array only. ArrayDB maintains order.

# InterfaceScore

- InterfaceScore

  - InterfaceScore is an interface class used to invoke other SCORE's external function as if it is a local function call.

  - Get InterfaceScore by using IconScoreBase's built-in function create_interface_score

- IconScoreBase's create_interface_score

  - create_interface_score returns an object, through which you have an access to the designated SCORE's external functions

# Logger

- Log Configuration Files

    - T-Bears : tbears_server_config.json

```
TAG = 'MyScore'

Logger.debug(f'on_install: total_supply={total_supply}', TAG)
Logger.info(f'on_install: total_supply={total_supply}', TAG)
Logger.warning(f'on_install: total_supply={total_supply}', TAG)
Logger.error(f'on_install: total_supply={total_supply}', TAG)
```

# Address

- Address

  - prefix : AddressPrefix.EOA(0) or AddressPrefix.CONTRACT(1)

  - body : 20-byte address body part

  - is_contract : Whether the address is SCORE

  - to_bytes : Returns data as bytes from the address object

  - from_string : Static method creates an address object from given 42-char string

  - from_data : Static method creates an address object using given bytes data

# Audit Checklist

- Loop : Make sure that the code always reaches the exit condition

- import : Package import is prohibited generally except iconservice

- Randomness : Execution result of SCORE must be deterministic

    …

Checklist : Github repository [Link - Click]

# Summary

- IconScoreBase
    - Must inherit this class to create SCORE

- InterfaceScore
    - Interface class used to invoke other SCORE's external function as if it is a local function call

- Utility Functions
    - json_dumps, json_loads, sha3_256, revert

- State DB
    - VarDB, DictDB, ArrayDB

# 3.2. Introduction to T-Bears

# ICON Developers Meetup

Week 1. ICON Basics

Week 2. Interacting with ICON nodes

Week 3. Smart Contract Development

- 3.1. Smart Contract Basics
- 3.2. Introduction to T-Bears
- 3.3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# Introduction to T-Bears

- Docker container for T-Bears

```
$ docker run -it -p 9000:9000 --name tbears-container iconloop/tbears
$ docker start -i tbears-container
```

- Server configuration
    - Transaction fee, Log, Test accounts, Block generation ...

- Basic commands for SCORE development
    - init, deploy
    - scoreapi, call
    - sendtx, txresult

# About T-Bears

| No | Command | Description |
|---|---|---|
| 1 | start | Start tbears service |
| 2 | stop | Stop tbears service |
| 3 | deploy | Deploy SCORE |
| 4 | clear | Clear all SCOREs deployed on tbears service |
| 5 | init | Initialize tbears project |

# About T-Bears

| No | Command | Description |
|----|---------|-------------|
| 6 | test | Run the unittest in the SCORE |
| 7 | scoreapi | Get SCORE's API using given SCORE address. |
| 8 | call | Request icx_call with user input json file. |
| 9 | txresult | Get transaction's result by transaction hash |
| 10 | sendtx | Request icx_sendTransaction with user input json file. |

# Basic commands for SCORE development

- tbears <command> -h

```
tbears <command> -h

This will show you <Help> message about <commands>

e.g) tbears init -h
     usage: tbears init [-h] project scoreClass

     Initialize SCORE development environment. Generate <project>.py, package.json and test code in
     <project> directory. The name of the score class is <scoreClass>.

     positional arguments:
      project     Project name
      scoreClass  SCORE class name

     optional arguments:
       -h, --help  show this help message and exit
```

# Summary

- Basic commands for SCORE development

    - init : Initialize T-Bears project

    - scoreapi : Get score's api using given score address

    - call : Request icx_call with user input json file

    - sendtx : Request icx_sendTransaction with user input json file and keystore file.

    - txresult : Get transaction result by transaction hash

    - deploy : Deploy the SCORE

# 3.3. Smart Contract Development

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

- 3.1. Smart Contract Basics
- 3.2. Introduction to T-Bears
- 3.3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

Appendix C. Development Resources

# SCORE Development

- Today's Goal : SCORE Development & Test code implementation

  - $ tbears init [projectName] [className]

  - To do List on today

    - Objective : Getting familiar with SCORE

    - Requirement 1 : Use database [VarDB, ArrayDB, DictDB]

    - Requirement 2 : fallback & tokenFallback

    - Requirement 3 : Test code

Sample SCORE : Git Repository [Link - Click]

# SCORE Review - GreedyHello.py

```python
from iconservice import *
TAG = 'Scrooge'

class Scrooge(IconScoreBase):
    _CONTRIBUTOR_ICX = "contributor_icx"
    _CONTRIBUTOR_VALUE = "contributor_value"
    _CONTRIBUTOR_LIST = "contributor_list"
    _ICX_BALANCE = "icx_balance"
    _TOKEN_BALANCE = "token_balance"

    def __init__(self, db: IconScoreDatabase) -> None:
        super().__init__(db)
        self._ADB_contributor_list = ArrayDB(self._CONTRIBUTOR_LIST, db, value_type=Address)
        self._DDB_contributor_icx = DictDB(self._CONTRIBUTOR_ICX, db, value_type=int)
        self._DDB_contributor_token = DictDB(self._CONTRIBUTOR_VALUE, db, value_type=int)
        self._VDB_icx_balance = VarDB(self._ICX_BALANCE, db, value_type=int)
        self._VDB_token_balance = VarDB(self._TOKEN_BALANCE, db, value_type=int)
        print(f"__init__ : {TAG}")

    def on_install(self) -> None:
        super().on_install()
        print(f"on_install : {TAG}")

    def on_update(self) -> None:
        super().on_update()
        print(f"on_update : {TAG}")

    ...
```

# SCORE Review - GreedyHello.py

```
    ...
        @external
        def tokenFallback(self, _from: Address, _value: int, _data: bytes):
            if self.msg.sender not in self._ADB_contributor_list:
                self._ADB_contributor_list.put(self.msg.sender)
            print("Total contributors : " + str(len(self._ADB_contributor_list)))

            self._DDB_contributor_token[self.msg.sender] += _value
            self._VDB_token_balance.set(self._VDB_token_balance.get() + _value)
            print(f'Token is Acceptable. {_from} sent {_value} tokens to {TAG} SCORE Class')
            print("Get SCORE's token balance : " + str(self._VDB_token_balance.get()))

        @payable
        def fallback(self) -> None:
            if self.msg.sender not in self._ADB_contributor_list:
                self._ADB_contributor_list.put(self.msg.sender)
            print("Total contributors : " + str(len(self._ADB_contributor_list)))

            self._DDB_contributor_icx[self.msg.sender] += self.msg.value
            self._VDB_icx_balance.set(self._VDB_icx_balance.get() + self.msg.value)
            print(f'ICX is Acceptable. {self.msg.sender} sent {self.msg.value} ICX to {TAG} SCORE Class')
            print("Get SCORE's balance : " + str(self._VDB_icx_balance.get()))
```

# How to Run a SCORE Test

- How to run a test & Why?

    - $ tbears test <SCORE project's path>


    - Benefits when you write test code

        - Takes less time to test SCORE via serial transaction call

        - Test is available without deployment

        - Can check the variable on running environment by print method

Source Reference : Git Repository [Link - Click]

# SCORE Review - test_GreedyHello.py

```python
import os

from iconsdk.builder.call_builder import CallBuilder
from iconsdk.builder.transaction_builder import DeployTransactionBuilder, TransactionBuilder, CallTransactionBuilder
from iconsdk.libs.in_memory_zip import gen_deploy_data_content
from iconsdk.signed_transaction import SignedTransaction
from tbears.libs.icon_integrate_test import IconIntegrateTestBase, SCORE_INSTALL_ADDRESS

DIR_PATH = os.path.abspath(os.path.dirname(__file__))

class TestScrooge(IconIntegrateTestBase):
    SCORE_PROJECT = os.path.abspath(os.path.join(DIR_PATH, '..'))
    SAMPLE_TOKEN = os.path.abspath(os.path.join(DIR_PATH, '../{SampleTokenSCORE's path}'))

    def setUp(self):
        super().setUp()

        self.icon_service = None
        self._score_address = self._deploy_score(self.SCORE_PROJECT)['scoreAddress']
...
```

```python
...
    def _deploy_score(self, scorepath: str, to: str = SCORE_INSTALL_ADDRESS, _params: dict = None) -> dict:
        transaction = DeployTransactionBuilder() \
            .from_(self._test1.get_address()) \
            .to(to) \
            .step_limit(100_000_000_000) \
            .nid(3) \
            .content_type("application/zip") \
            .content(gen_deploy_data_content(scorepath)) \
            .params(_params) \
            .build()

        signed_transaction = SignedTransaction(transaction, self._test1)
        tx_result = self.process_transaction(signed_transaction, self.icon_service)

        self.assertTrue('status' in tx_result)
        self.assertEqual(1, tx_result['status'])
        self.assertTrue('scoreAddress' in tx_result)

        return tx_result
...
```

# SCORE Review - test_GreedyHello.py

```python
...

    def test_score_update(self):
        tx_result = self._deploy_score(scorepath=self.SCORE_PROJECT, to=self._score_address)

        self.assertEqual(1, tx_result['status'])
        self.assertTrue('scoreAddress' in tx_result)
        self.assertEqual(self._score_address, tx_result['scoreAddress'])

    def test_fallback(self):
        transaction = TransactionBuilder() \
            .from_(self._test1.get_address()) \
            .to(self._score_address) \
            .value(1) \
            .step_limit(10000000000) \
            .nid(3) \
            .build()

        signed_transaction = SignedTransaction(transaction, self._test1)
        tx_result = self.process_transaction(signed_transaction, self.icon_service)

        self.assertTrue('status' in tx_result)
        self.assertEqual(1, tx_result['status'])

...
```

# SCORE Review - test_GreedyHello.py

```python
...
    def test_token_fallback(self):
        input_params = {"_initialSupply": 1000, "_decimals": 1}
        token_deploy_result = self._deploy_score(scorepath=self.SAMPLE_TOKEN, _params=input_params)
        score_address = token_deploy_result['scoreAddress']

        transaction = CallTransactionBuilder().from_(self._test1.get_address()) \
            .to(score_address) \
            .step_limit(10000000000) \
            .nid(3) \
            .method("transfer") \
            .params({"_to": self._score_address, "_value": 10}) \
            .build()

        signed_transaction = SignedTransaction(transaction, self._test1)
        tx_result = self.process_transaction(signed_transaction, self.icon_service)

        self.assertEqual(1, tx_result['status'])

        call = CallBuilder().from_(self._test1.get_address()) \
            .to(score_address) \
            .method("balanceOf") \
            .params({"_owner": self._score_address})\
            .build()

        response = self.process_call(call, self.icon_service)
        self.assertEqual('0xa', response)
```

# SCORE Review - Expected Output

```
$ tbears test <SCORE project's path>
```

```
__init__ : Scrooge
on_install : Scrooge
Total contributors : 1
ICX is Acceptable. hxe7af5fcfd8dfc67530a01a0e403882687528dfcb sent 1 ICX to Scrooge SCORE Class
Get SCORE's balance : 1
.__init__ : Scrooge
on_install : Scrooge
__init__ : Scrooge
on_update : Scrooge
.__init__ : Scrooge
on_install : Scrooge
Total contributors : 1
Token is Acceptable. hxe7af5fcfd8dfc67530a01a0e403882687528dfcb sent 10 tokens to Scrooge SCORE Class
Get SCORE's token balance : 10
.
----------------------------------------------------------------------
Ran 3 tests in 0.244s

OK
```

# Summary

- How to run a test & Why ?

    - $ tbears test <SCORE project's path>


    - Benefits when you write test code

        - Takes less time to test SCORE via serial transaction call

        - Test is available without deployment

        - Can check the variable on running environment by print method

# Appendix A. T-Bears Quickstart

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

- A.1 Deploy, Call and SendTransaction

Appendix B. AWS Development Network

Appendix C. Development Resources

# How to Deploy & Update with T-Bears

```
$ vi deploy.json
```

```json
{
    "uri": <API endpoint URI> e.g) "https://bicon.net.solidwallet.io/api/v3",
    "nid": "0x3",
    "to": "cx0000000000000000000000000000000000000000" for install or <SCORE address> for update,
    "deploy": {
        "stepLimit": <steplimit> e.g) "0x12a05f200",
        "mode": "install" or "update",
        "scoreParams": {
            <key> e.g) "initialSupply" : <value> e.g) "1000"
        }
    },
}
```

```
$ tbears deploy <project> -c deploy.json -k <keystore>
```

# How to Call with T-Bears

```
$ vi call.json
```

```
{
    "jsonrpc": "2.0",
    "method": "icx_call",
    "params": {
        "to": <SCORE address>,
        "dataType": "call",
        "data": {
            "method": <readonly method to call> e.g) "hello"
        }
    },
    "id": 1
}
```

```
$ tbears call call.json -u <endpoint_URL>
```

# How to SendTransaction with T-Bears

```
$ vi sendtx.json
```

```
                                                    ...
{
  "jsonrpc": "2.0",                                     "data": {
  "method": "icx_sendTransaction",                        "method": <method to call> e.g) "setValue",
  "params": {                                             "params": {
    "version": "0x3",                                       <key> e.g) "value": <value> e.g) "0x123"
    "value": <ICX value> e.g) "0x0",                      }
    "stepLimit": <stepLimit> e.g) "0x3000000",          }
    "timestamp": <timestamp> e.g) "0x573117f1d6568",  },
    "nid": "0x3",                                       "id": 1
    "to": <SCORE address>,                            }
    "dataType": "call",

...
```

```
$ tbears sendtx sendtx.json -k <keystore> -u <endpoint_URL>
```

# Summary

- T-Bears Quickstart

    - Deploy & Update : `tbears deploy`

    - Call : `tbears call`

    - SendTransaction : `tbears sendtx`

T-Bears Documentaion : Git Repository [Link - Click]

# Appendix B. AWS Development Network

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

- B.1 AWS Marketplace : ICON Development Network
- B.2 AWS CloudFormation
- B.3 Interacting with Development Network

Appendix C. Development Resources

# AWS Marketplace

AWS Marketplace - ICON Development Network

Why?

- Easy to install

- Scalable to 4 ~ 12 nodes

- Provides various monitoring tools

# AWS Marketplace

1. Visit Marketplace [Link : Click]

# AWS Marketplace

## 2. Go to Usage Information

# AWS Marketplace

3. Download CloudFormation Template

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

- B.1 AWS Marketplace : ICON Development Network
- B.2 AWS CloudFormation
- B.3 Interacting with Development Network

Appendix C. Development Resources

# AWS CloudFormation

AWS CloudFormation - Create AWS Stack

Resource

- CloudFormation Template - ICON Development Network

- e.g) icon_dev_network.template

# AWS CloudFormation

## 1. AWS Console ⇒ Service ⇒ CloudFormation
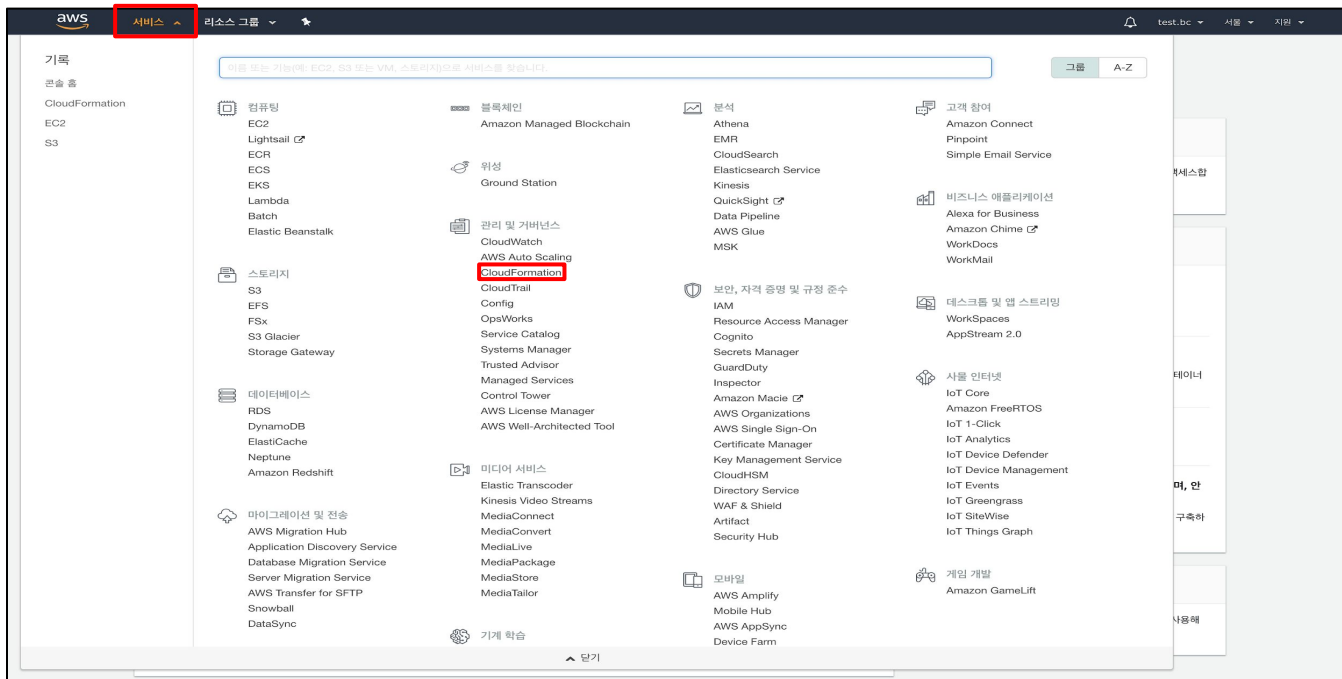
# AWS CloudFormation

## 2. Click Template Design
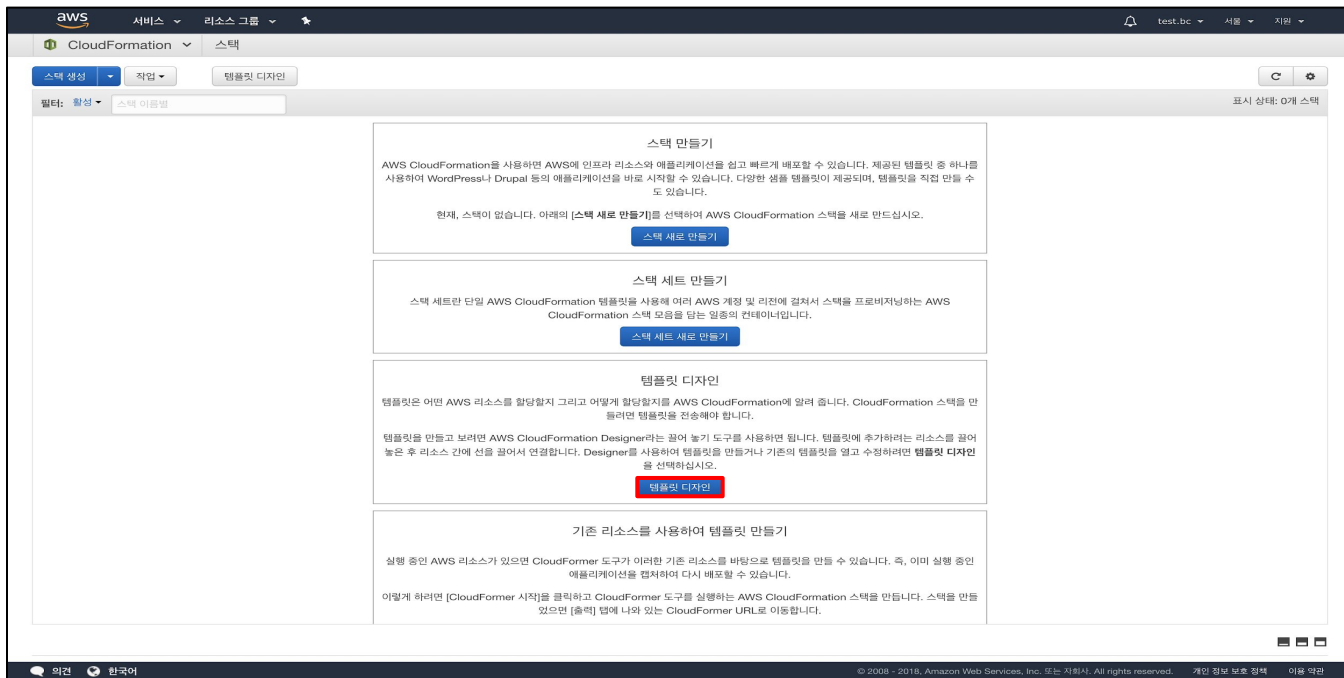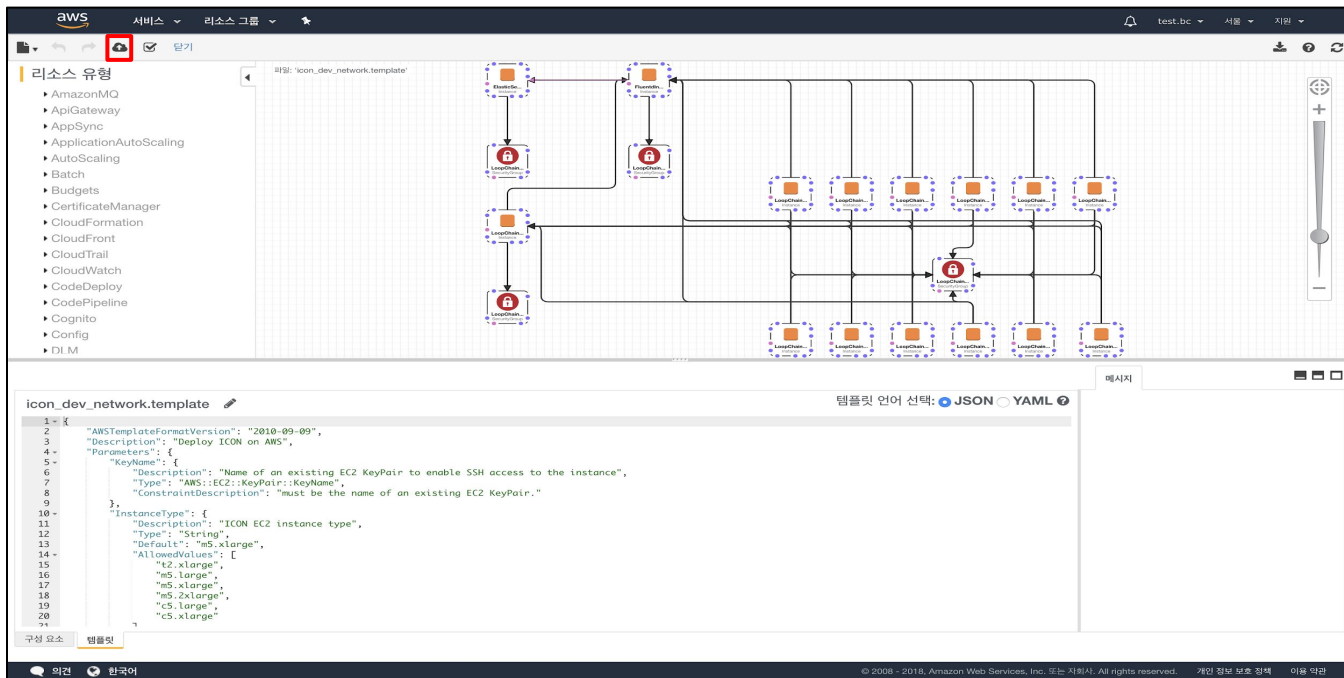
# AWS CloudFormation

3. Open "icon_dev_network.template" on local storage
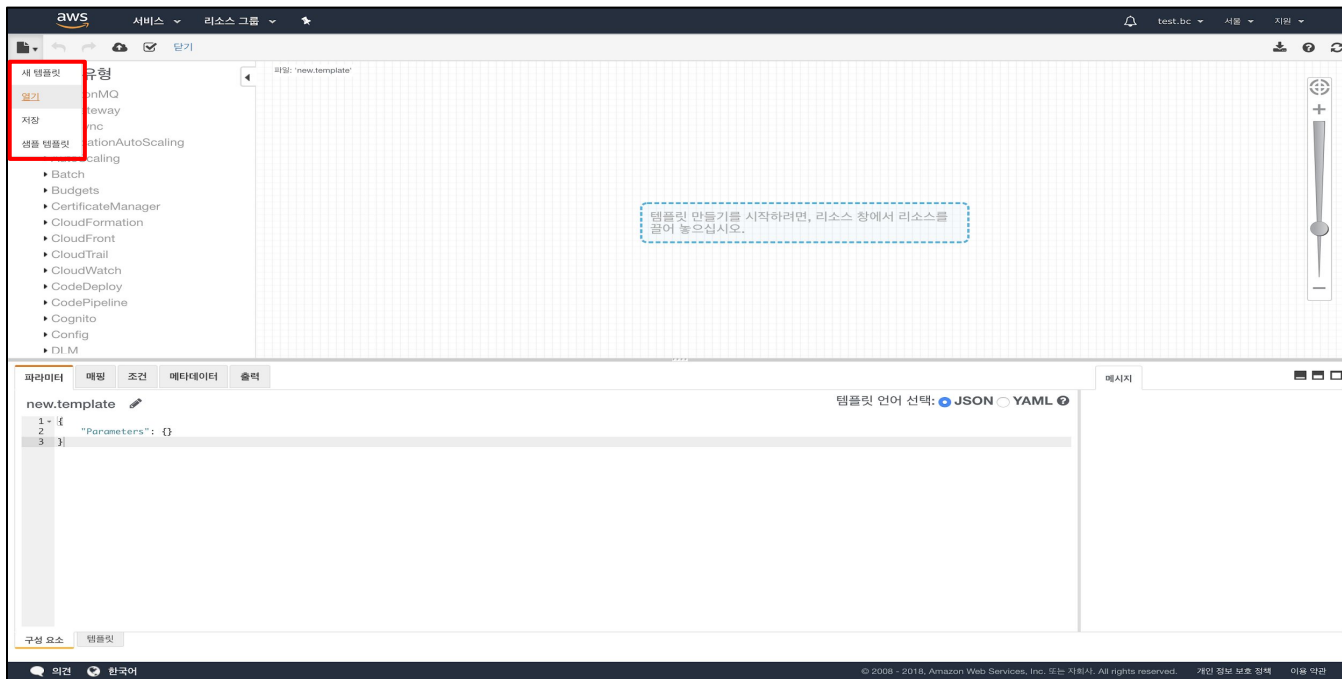
# AWS CloudFormation

### 4. Create AWS Stack

# AWS CloudFormation

## 5. Select CloudFormation Template ⇒ Next

# AWS CloudFormation

## 6. Edit Configuration

# AWS CloudFormation

## 7. Finalize Options

# AWS CloudFormation

## 8. Check Loaded Instances

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network

- B.1 AWS Marketplace : ICON Development Network
- B.2 AWS CloudFormation
- B.3 Interacting with Development Network

Appendix C. Development Resources

# Interacting with Development Network

## How to use ICON Development Network

- Usage Instructions :
    - Access the application via a browser at http://<radiostation_dns>:9002/api/v1/peer/list.
    - To connect to the operating system, use SSH and the username ec2-user.

- JSON-RPC API v3 Document :
    https://github.com/icon-project/icon-rpc-server/blob/master/docs/icon-json-rpc-v3.md

- DApp Developer Guide : https://www.icondev.io

# Interacting with Development Network

Send Query to Each Nodes - vi sendQuery.sh

```
#!/usr/bin/env zsh

echo "-----peer#1-----"
echo "Total Supply"
tbears totalsupply -u http://{peer#1 URL}:9000/api/v3
echo "----------------"
echo "Last Block"
tbears lastblock -u http://{peer#1 URL}:9000/api/v3


echo "-----peer#2-----"
echo "Total Supply"
tbears totalsupply -u http://{peer#2 URL}:9000/api/v3
echo "----------------"
echo "Last Block"
tbears lastblock -u http://{peer#2 URL}:9000/api/v3


...
```

# Interacting with Development Network

Send Query to Each Nodes - vi sendQuery.sh

```
...

echo "-----peer#3-----"
echo "Total Supply"
tbears totalsupply -u http://{peer#3 URL}:9000/api/v3
echo "----------------"
echo "Last Block"
tbears lastblock -u http://{peer#3 URL}:9000/api/v3


echo "-----peer#4-----"
echo "Total Supply"
tbears totalsupply -u http://{peer#4 URL}:9000/api/v3
echo "----------------"
echo "Last Block"
tbears lastblock -u http://{peer#4 URL}:9000/api/v3
```

# Interacting with Development Network

## Query Results - **peer#1**

```
-----peer#1-----
Total Supply
Total supply of ICX in hex: 0x52c3ff441ba8feb88e00000
Total supply of ICX in decimal: 1600919800000000000000000000
------
Last Block
block info : {
    "jsonrpc": "2.0",
    "result": {
        ...
        "merkle_tree_root_hash": "72722dad5bdb0adff8f5fbb061de07a2404b522d463b708d1a62071262616fb3",
        ...
        "block_hash": "b3c99e161f8013b7dabc2ff8be56bacb3209853d139bb54f0a37384388a36505",
        ...
    },
    "id": 1
}
```

# Interacting with Development Network

## Query Results - **peer#2**

```
-----peer#2-----
Total Supply
Total supply of ICX in hex: 0x52c3ff441ba8feb88e00000
Total supply of ICX in decimal: 16009198000000000000000000
------
Last Block
block info : {
    "jsonrpc": "2.0",
    "result": {
        ...
        "merkle_tree_root_hash": "72722dad5bdb0adff8f5fbb061de07a2404b522d463b708d1a62071262616fb3",
        ...
        "block_hash": "b3c99e161f8013b7dabc2ff8be56bacb3209853d139bb54f0a37384388a36505",
        ...
    },
    "id": 1
}
```

# Interacting with Development Network

## Query Results - **peer#3**

```
-----peer#3-----
Total Supply
Total supply of ICX in hex: 0x52c3ff441ba8feb88e00000
Total supply of ICX in decimal: 1600919800000000000000000000
------
Last Block
block info : {
    "jsonrpc": "2.0",
    "result": {
        ...
        "merkle_tree_root_hash": "72722dad5bdb0adff8f5fbb061de07a2404b522d463b708d1a62071262616fb3",
        ...
        "block_hash": "b3c99e161f8013b7dabc2ff8be56bacb3209853d139bb54f0a37384388a36505",
        ...
    },
    "id": 1
}
```

# Interacting with Development Network

### Query Results - **peer#4**

```
-----peer#4-----
Total Supply
Total supply of ICX in hex: 0x52c3ff441ba8feb88e00000
Total supply of ICX in decimal: 1600919800000000000000000000
------
Last Block
block info : {
    "jsonrpc": "2.0",
    "result": {
        ...
         "merkle_tree_root_hash": "72722dad5bdb0adff8f5fbb061de07a2404b522d463b708d1a62071262616fb3",
        ...
         "block_hash": "b3c99e161f8013b7dabc2ff8be56bacb3209853d139bb54f0a37384388a36505",
        ...
    },
    "id": 1
}
```

# Summary

- AWS Marketplace

  https://aws.amazon.com/marketplace/pp/B07KBTZHZD?qid=1544602499452&sr=0-1&ref_=srh_res_product_title

- AWS CloudFormation

  - Create AWS Stack & Load Instances

  - Resource : CloudFormation Template

# Appendix C. Development Resources

# ICON Developers Meetup

Step 1. ICON Basics

Step 2. Interacting with ICON nodes

Step 3. Smart Contract Development

Appendix A. T-Bears Quickstart

Appendix B. AWS Development Network
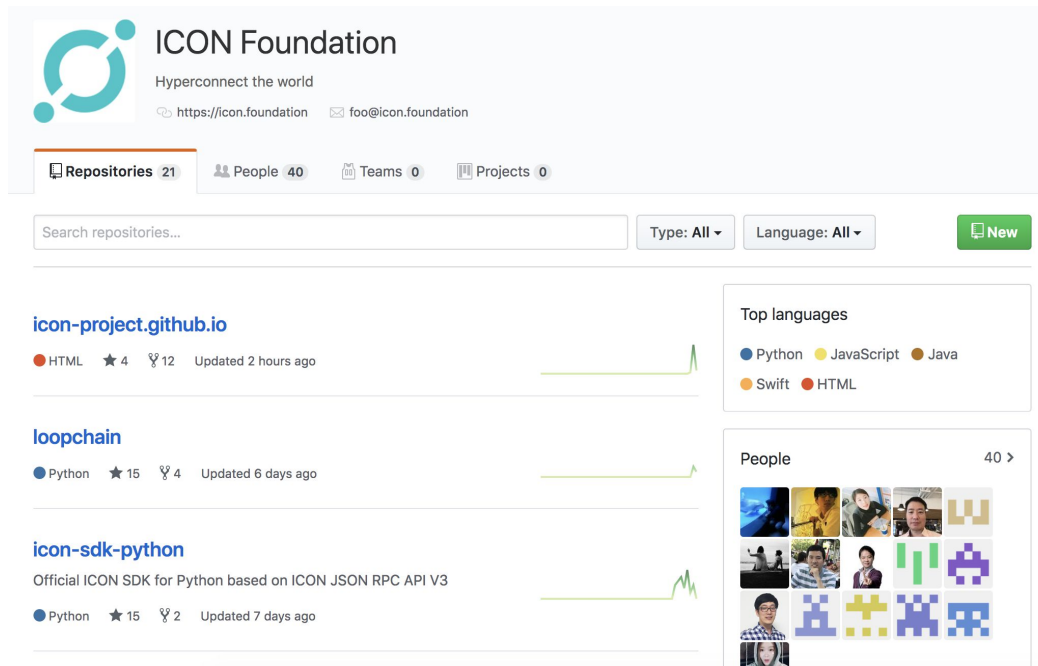
Appendix C. Development Resources

# Development Resources

- GitHub

- Developer Portal

- ICON official Documentation Project

- ICON Improvement Proposal

# GitHub  https://github.com/icon-project

- loopchain

- icon-service

- icon-rpc-server

Node

- t-bears

- icon-sdk-python

- icon-sdk-java

- icon-sdk-js

Dev tools

- iconex_android

- iconex_ios

- iconex_chrome_extension

## ICON Foundation
Hyperconnect the world

https://icon.foundation    ✉ foo@icon.foundation

Repositories 21    People 40    Teams 0    Projects 0

Search repositories...    Type: All ▾    Language: All ▾    New

**icon-project.github.io**

🔴 HTML    ★ 4    ⑂ 12    Updated 2 hours ago

**loopchain**

🔵 Python    ★ 15    ⑂ 4    Updated 6 days ago

**icon-sdk-python**

Official ICON SDK for Python based on ICON JSON RPC API V3

🔵 Python    ★ 15    ⑂ 2    Updated 7 days ago

Top languages

🔵 Python    🟡 JavaScript    🟤 Java
🟠 Swift    🔴 HTML

People    40 ▸

# Developer Portal  https://www.icondev.io

- Community portal for ICON DApp ecosystem

**Getting Started**
Tutorials for developers to get started
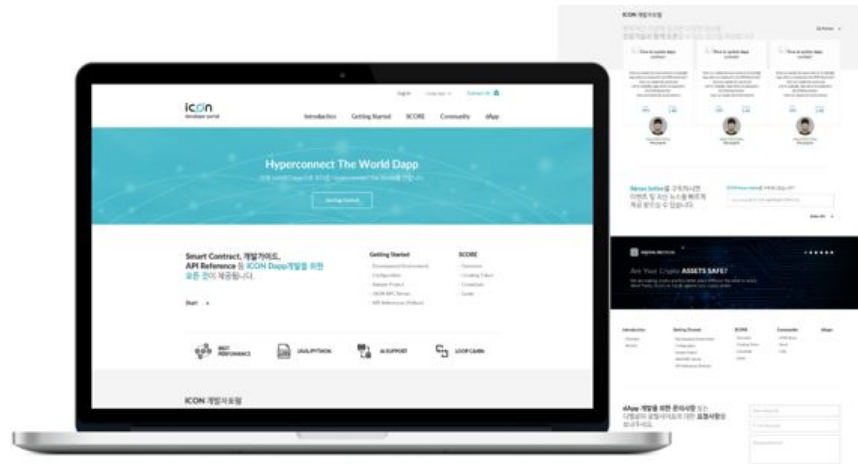
**SCORE**
Details on ICON's Smart Contract, SCORE

**Community**
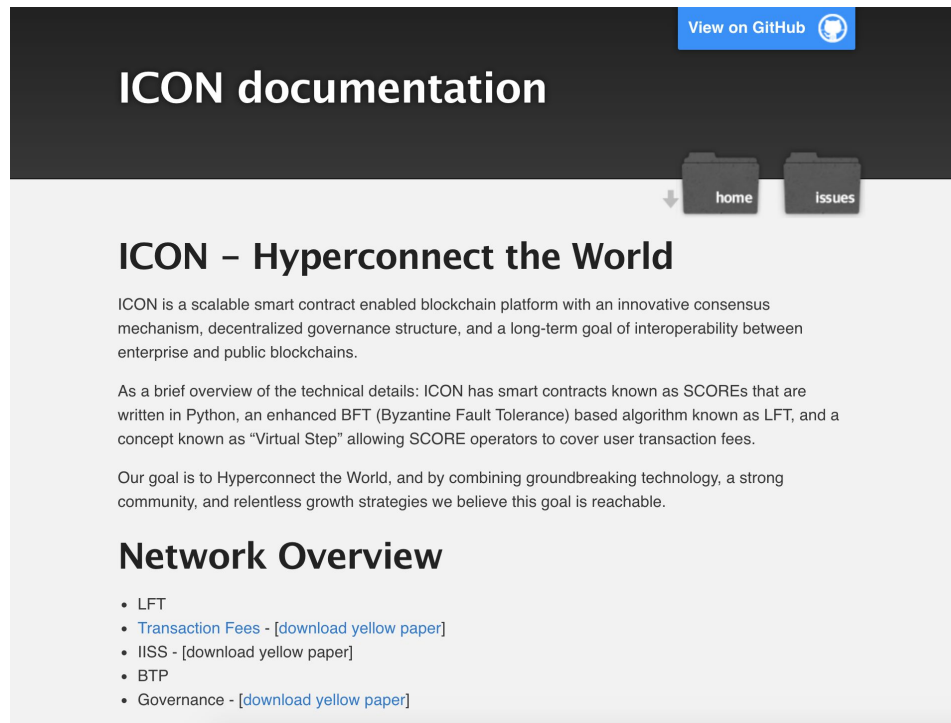Forum for Korean/English developers to discuss and communicate

**DApp**
Overview of ICON DApp Partners

# Documentation Project https://icon-project.github.io

- GitHub Pages
  - ICON overview
  - Network info
  - Account management
  - Client SDKs
  - SCORE development
  - Mainnet SCORE audit guideline
  - Tutorials with sample codes

- We welcome contributors !
  - Read CONTRIBUTING.md
  - French / Chinese translation provided by volunteer contributors.

View on GitHub

## ICON documentation

home   issues

## ICON – Hyperconnect the World

ICON is a scalable smart contract enabled blockchain platform with an innovative consensus mechanism, decentralized governance structure, and a long-term goal of interoperability between enterprise and public blockchains.

As a brief overview of the technical details: ICON has smart contracts known as SCOREs that are written in Python, an enhanced BFT (Byzantine Fault Tolerance) based algorithm known as LFT, and a concept known as "Virtual Step" allowing SCORE operators to cover user transaction fees.

Our goal is to Hyperconnect the World, and by combining groundbreaking technology, a strong community, and relentless growth strategies we believe this goal is reachable.

## Network Overview

- LFT
- Transaction Fees - [download yellow paper]
- IISS - [download yellow paper]
- BTP
- Governance - [download yellow paper]

# ICON Improvement Proposal https://github.com/icon-project/IIPs

- IIP describes a standard for ICON platform.

- Anyone can prompt suggestions and discussions on new functions or improvement.

- Selected items will be implemented on ICON network.

- **For all other IIPs**, open a PR changing the state of your IIP to 'Final'. An editor will review your draft and ask if anyone objects to its being finalised. If the editor decides there is no rough consensus - for instance, because contributors point out significant issues with the IIP - they may close the PR and request that you fix the issues in the draft before trying again.

## IIP Status Terms

- **Draft** - an IIP that is open for consideration.
- **Last Call** - an IIP that is calling for last review before finalizing. IIPs that has been more than 2 weeks in Last Call without any technical changes or objections enters either Accepted or Final state.
- **Accepted** - an IIP that is planned for immediate adoption, i.e. expected to be included in the next release (for Core/Consensus layer IIPs only).
- **Final** - an IIP that has been adopted. For Core/Consensus layer IIPs, the implementation has been adopted in the mainnet.
- **Deferred** - an IIP that is not being considered for immediate adoption. May be reconsidered in the future.

## IIPs

| Number | Title | Author | Type | Status |
|--------|-------|--------|------|--------|
| 1 | IIP Purpose and Guidelines | Sojin Kim | Meta | Active |
| 2 | ICON Token Standard | Jaechang Namgoong | IRC | Final |
| 3 | ICON Non-Fungible Token Standard | Jaechang Namgoong | IRC | Draft |
| 6 | ICON Name Service Standard | Phyrex Tsai, Portal Network Team | IRC | Draft |

# Summary

- GitHub

- Developer Portal

- ICON official Documentation Project

- ICON Improvement Proposal