

# ICON Developers Meetup

ICON foundation

# Prerequisite for hands-on lab

Docker : Download link - <https://docs.docker.com>

T-Bears Docker :

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

- Guide : <https://github.com/icon-project/t-bears/tree/develop/Docker>

---

# 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

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](#)

[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

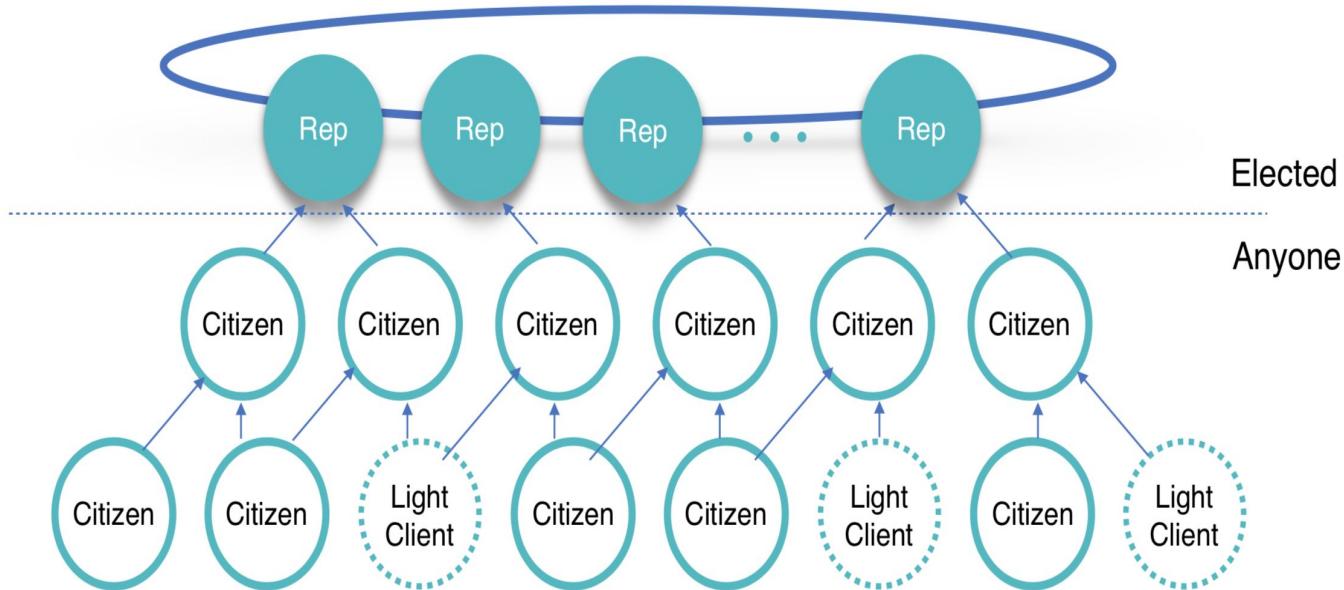
3rd Generation : EOS, AION, Zilliqa, **ICON** [2017] ...

# 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

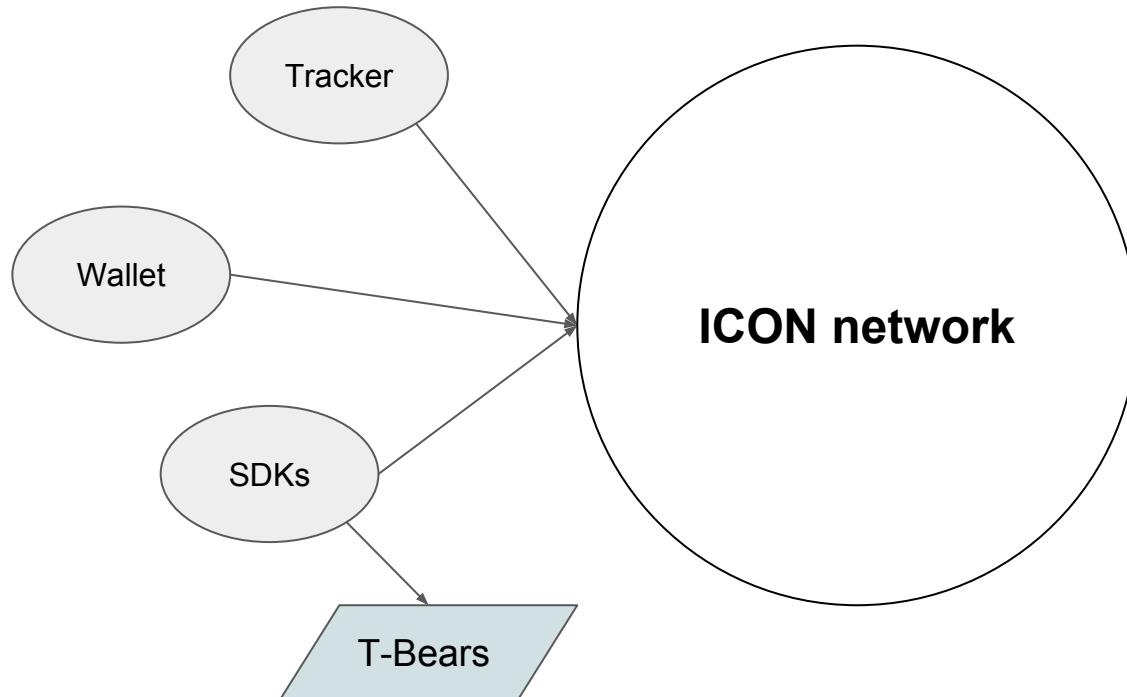
# ICON Network

- Representatives
- Citizen
- Light Client



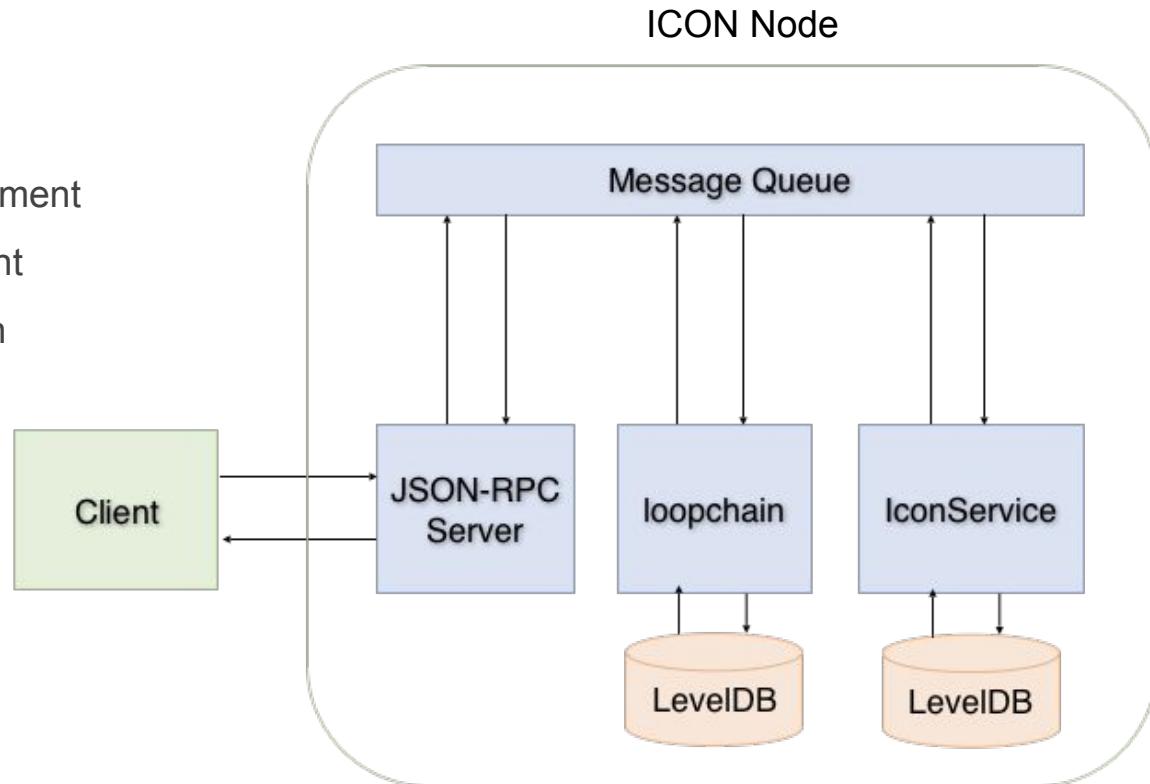
# Applications and Tools

- T-Bears
- SDK
- Wallet
- Tracker

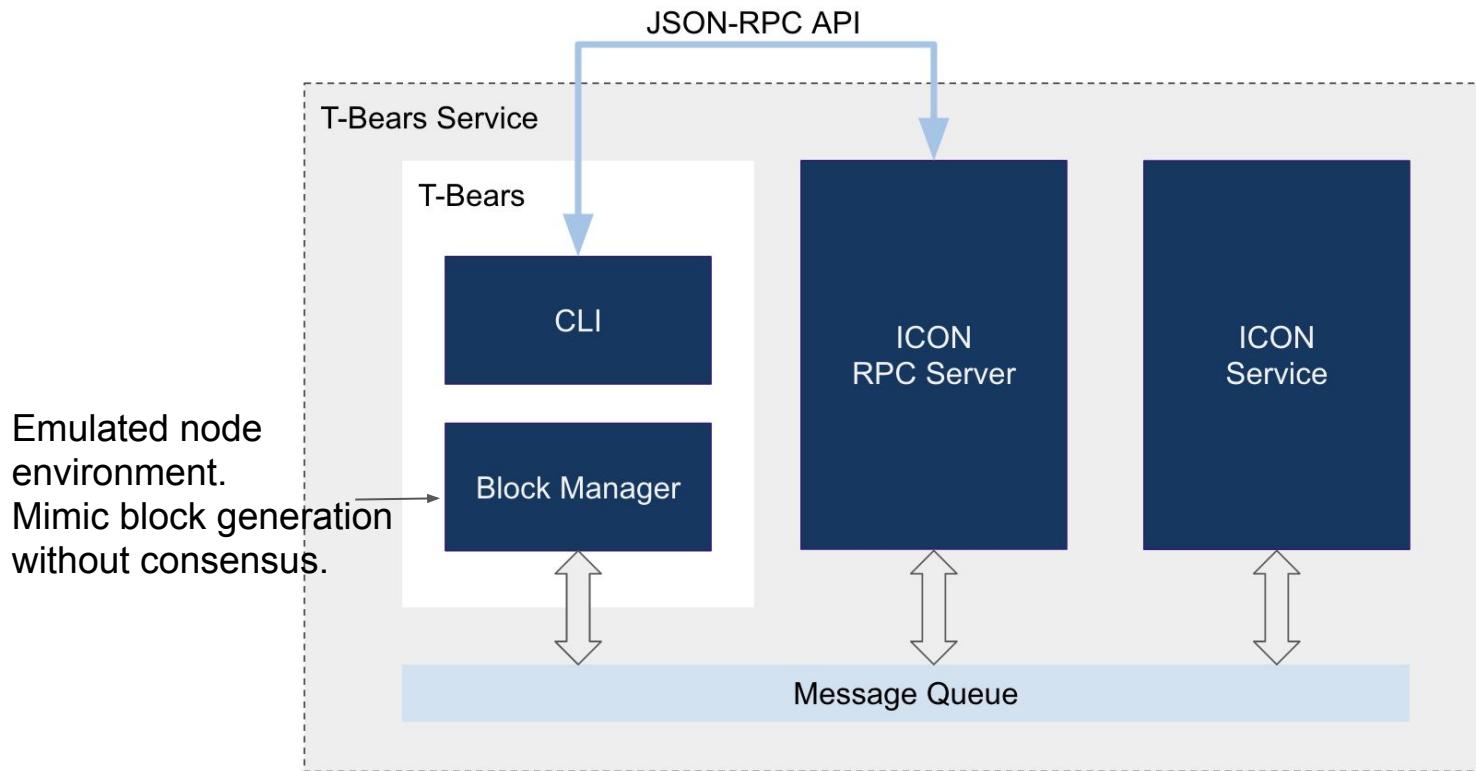


# Node View

- IconService
  - SCORE execution environment
  - ICX base coin management
  - Transaction fee calculation
- Loopchain
  - Peer management
  - Block management
  - LFT consensus engine



# T-Bears



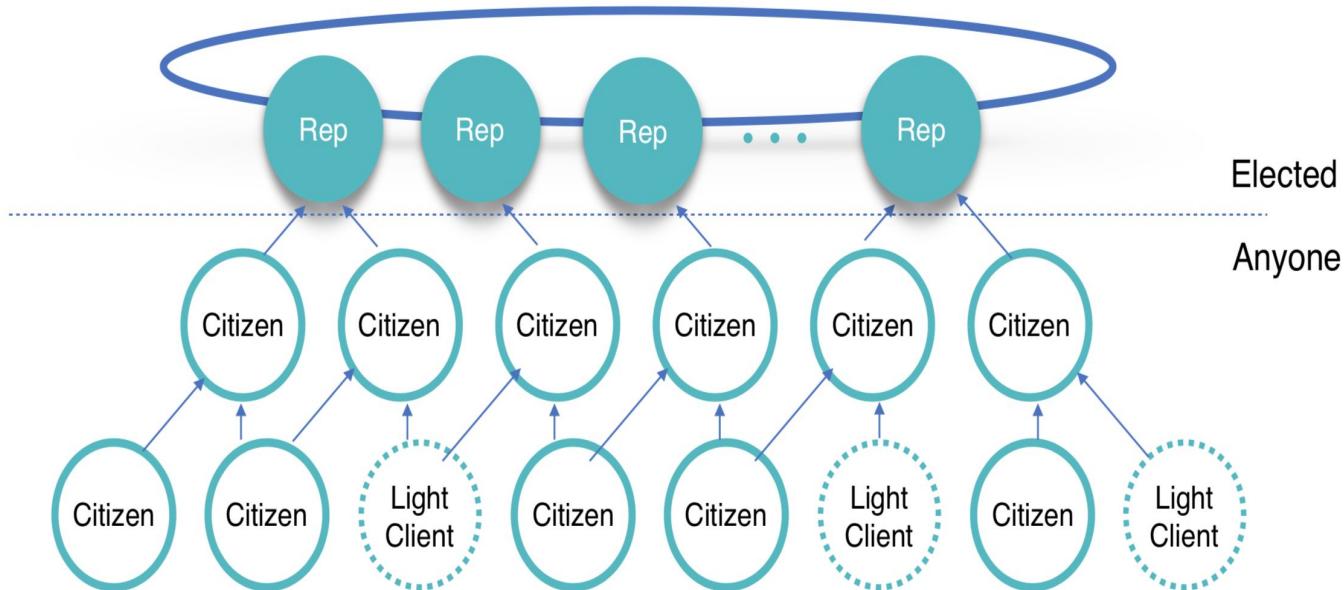
---

# 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

# 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

실습 :

- T-Bears 로 test 계정 생성
- T-Bears 로 testnet 에 query 보내기

## 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

# 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
  - hx6e1dd0d4432620778b54b2bbc21ac3df961adf89 ⇒ EOA
  - cxb25fe7b33016638b80fed733a4b1112cc8dbd27b ⇒ CA
- Transaction types
  - 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

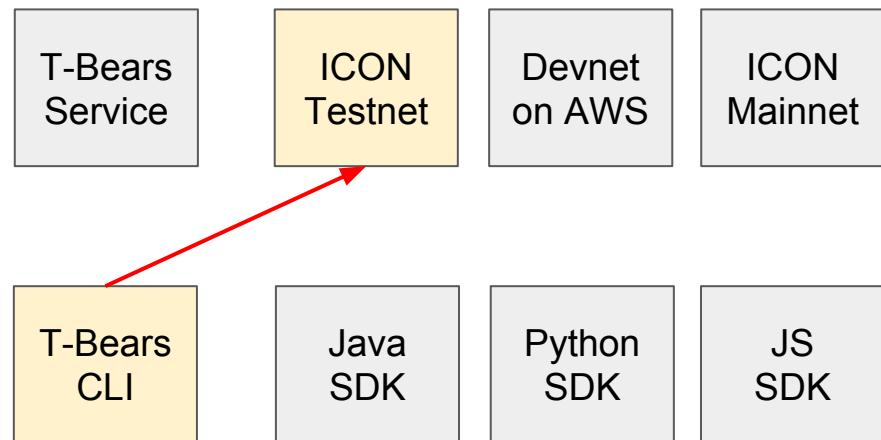
Appendix A. T-Bears Quickstart

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



# Testnet : Create an Account

- \$ tbears keystore [file\_path]
- Examine the keystore file

```
{  
    "address": "hx7af5fcfd8dfc67530a01a0e403882687528dfcb",  
    "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	<a href="https://bicon.net.solidwallet.io">https://bicon.net.solidwallet.io</a>
API endpoint	<a href="https://bicon.net.solidwallet.io/api/v3">https://bicon.net.solidwallet.io/api/v3</a>
Network ID (nid)	3
Tracker	<a href="https://bicon.tracker.solidwallet.io">https://bicon.tracker.solidwallet.io</a>
Transaction fee	on
SCORE audit	off

# 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

실습 :

- greedyHello SCORE

## 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

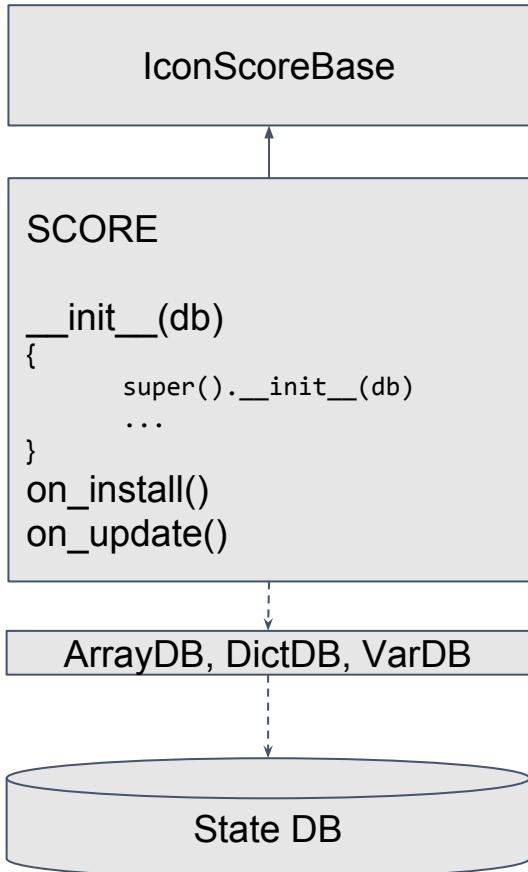
- What is smart contract?
- SCORE(Smart Contract On Reliable Environment)
  - What is SCORE?
    - Written in Python (Easy to learn)
    - State DB abstractions (VarDB, DictDB, ArrayDB)
    - Sandbox policy

# 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

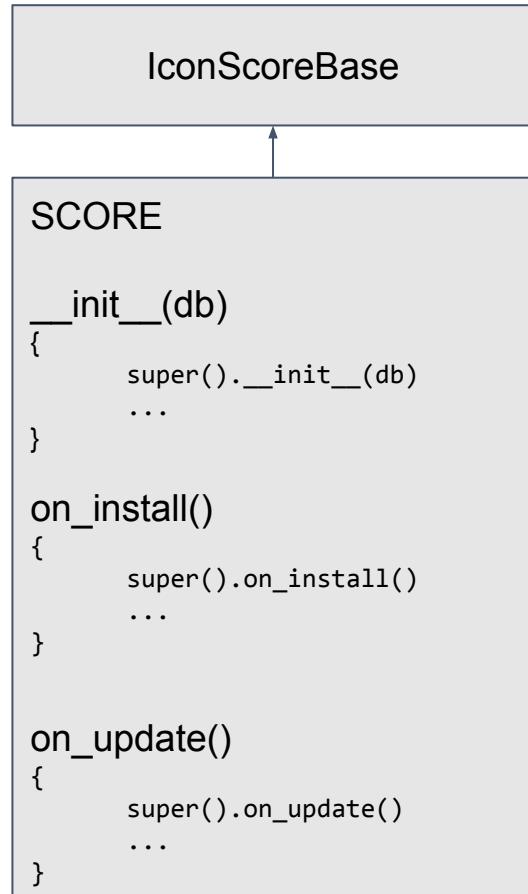
# SCORE Model

- Finite state machine
- State transition by transaction
- Must inherit IconScoreBase



# IconScoreBase - 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 - 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 occurred
  - 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

# Basic commands for SCORE development

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

# Basic commands for SCORE development

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.

# T-Bears help

- 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
- Learning object : Getting familiar with SCORE
  - 1 : Use database [VarDB, ArrayDB, DictDB]
  - 2 : Understand fallback & tokenFallback mechanism
  - 3 : Write and run integration test

Sample SCORE : Git Repository [Link - [Click](#)]

# Create an empty SCORE Project

- `$ tbears init [projectName] [className]`

Sample SCORE : Git Repository [Link - [Click](#)]

# GreedyHello.py

```
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)

    def on_install(self) -> None:
        super().on_install()

    def on_update(self) -> None:
        super().on_update()

    ...
```

# 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)

    self._DDB_contributor_token[self.msg.sender] += _value
    self._VDB_token_balance.set(self._VDB_token_balance.get() + _value)

@payable
def fallback(self) -> None:
    if self.msg.sender not in self._ADB_contributor_list:
        self._ADB_contributor_list.put(self.msg.sender)

    self._DDB_contributor_icx[self.msg.sender] += self.msg.value
    self._VDB_icx_balance.set(self._VDB_icx_balance.get() + self.msg.value)
```

# How to Run a SCORE Test

- \$ tbears test <SCORE project's path>
- **IconIntegrateTestBase**
  - Support python unittest
  - Emulate ICON service for test
    - self.\_test1 : Account with 1,000,000 ICX
    - self.\_wallet\_array[] : 10 empty accounts in list
  - Provides API for SCORE integration test
    - process\_transaction()
    - process\_call()

Source Reference : Git Repository [Link - [Click](#)]

# test\_GreedyHello.py

```
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' ]
    ...

```

# test\_GreedyHello.py

```
...
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
...
```

# test\_GreedyHello.py

```
...  
  
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'])  
  
...
```

# test\_GreedyHello.py

```
...
def test_tokenFallback(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)
```

# Run Test

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

```
...
-----
Ran 3 tests in 0.244s

OK
```

# Summary

- \$ tbears test <SCORE project's path>
- **IconIntegrateTestBase**
  - Support python unittest
  - Emulate ICON service for test
    - self.\_test1 : Account with 1,000,000 ICX
    - self.\_wallet\_array[] : 10 empty accounts in list
  - Provides API for SCORE integration test
    - process\_transaction()
    - process\_call()

# 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
```

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

```
$ 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](#)]

The screenshot shows the AWS Marketplace product page for the "ICON Development Network".

**Product Information:**

- Name:** ICON Development Network
- Provider:** Icon Foundation
- Latest Version:** 1.1
- Description:** ICON Development Network will allow developers to easily run a private instance of the ICON Blockchain Network through AWS.
- Operating System:** Linux/Unix
- Rating:** ★★★★☆ (0)

**Pricing:**

- Typical Total Price:** \$0.192/hr
- Total pricing per instance for services hosted on m4.2xlarge in US East (Virginia). [View Details](#)

**Product Overview:**

ICON Development Network platform will provide an easy-to-use development-ready environment. This enables developers to bootstrap their own private ICON Network, adapted to their own convenience and need, in order to build, test or validate their project running on top of the ICON Protocol. A private network built via AWS Cloud could be used to operate its own service as well in the context of a DApp and its production environment.

**Highlights:**

- Easy to install
- Scalable to 4 ~ 12 nodes
- Provides various monitoring tools

**Pricing Information:**

Use this tool to estimate the software and infrastructure costs based on your configuration choices. Your usage and costs might be different from this estimate. They will be reflected on your monthly AWS billing reports.

**Estimating your costs**

Choose your region and fulfillment option to see the pricing details. Then,

# AWS Marketplace

## 2. Go to Usage Information

The screenshot shows the AWS Marketplace product page for "ICON Development Network". The "Usage" tab is selected. In the "Fulfillment Options" section, there is a CloudFormation template titled "ICON DevNet Setup" which provisions "LoopChain Log Server, Monitoring Server, RadioStation Server, Peer Server Setup". There are three options under "View": "View Template Components", "View Usage Instructions", and "View CloudFormation Template", with the last one being highlighted by a red box. Below this is the "End User License Agreement" section, which contains a link to the "User License Agreement (EULA)".

In the "Additional Resources" section, there is a box titled "CloudFormation Template" explaining what CloudFormation templates are and how they work. It also includes a "Learn more" link.

The "Support" section at the bottom includes a "Support Information" box for "ICON Development Network" (with a note to allow 24 hours) and a "AWS Infrastructure Support" link.

# AWS Marketplace

## 3. Download CloudFormation Template

The screenshot shows the AWS Marketplace product page for 'ICON Development Network'. At the top, there are navigation links for Categories, Delivery Methods, Solutions, Migration Mapping Assistant, Your Saved List, and a search bar. On the right, there are links for Hello, test.bc, Partners, Sell in AWS Marketplace, Amazon Web Services Home, and Help.

The main content area has tabs for Overview, Pricing, Usage (which is selected), Support, and Reviews. The Usage tab displays 'Usage Information' and 'Fulfillment Options' for the 'ICON DevNet Setup' CloudFormation Template. It includes sections for 'LoopChain Log Server, Monitoring Server, RadioStation Server, Peer Server Setup' with links to 'View Template Components', 'View Usage Instructions', and 'Close CloudFormation Template'.

A large diagram illustrates the architecture, showing multiple EC2 instances connected to various services like RDS, Lambda, and API Gateway. Below the diagram, there are buttons for 'Download CloudFormation Template' (highlighted with a red box) and 'View Template in CloudFormation Designer'.

On the right side, there are sections for 'Additional Resources' (with a 'How it Works' link) and 'CloudFormation Template' (with a detailed description of what CloudFormation templates are and how they work). There is also a 'Learn more' link.

At the bottom, there is an 'End User License Agreement' section with a note about agreeing to terms and conditions, and a link to the 'User License Agreement (EULA)'.

# 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

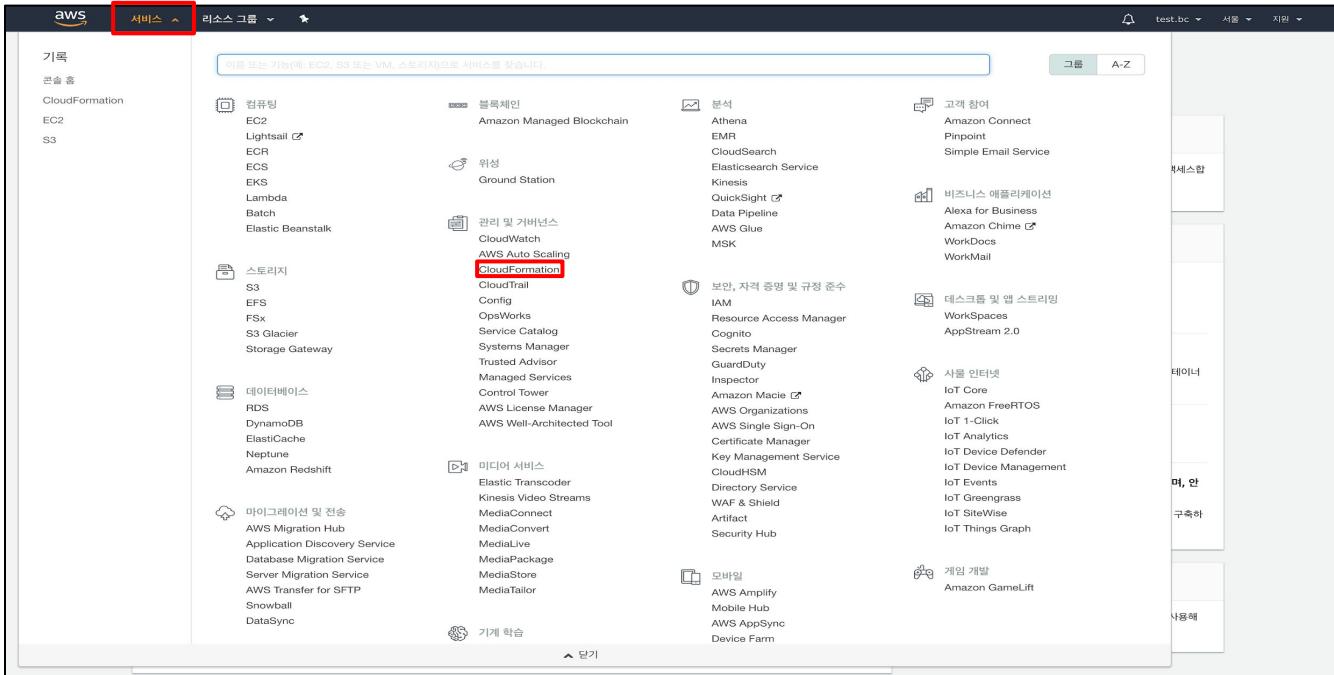
## AWS CloudFormation - Create AWS Stack

### Resource

- CloudFormation Template - ICON Development Network
- e.g) icon\_dev\_network.template

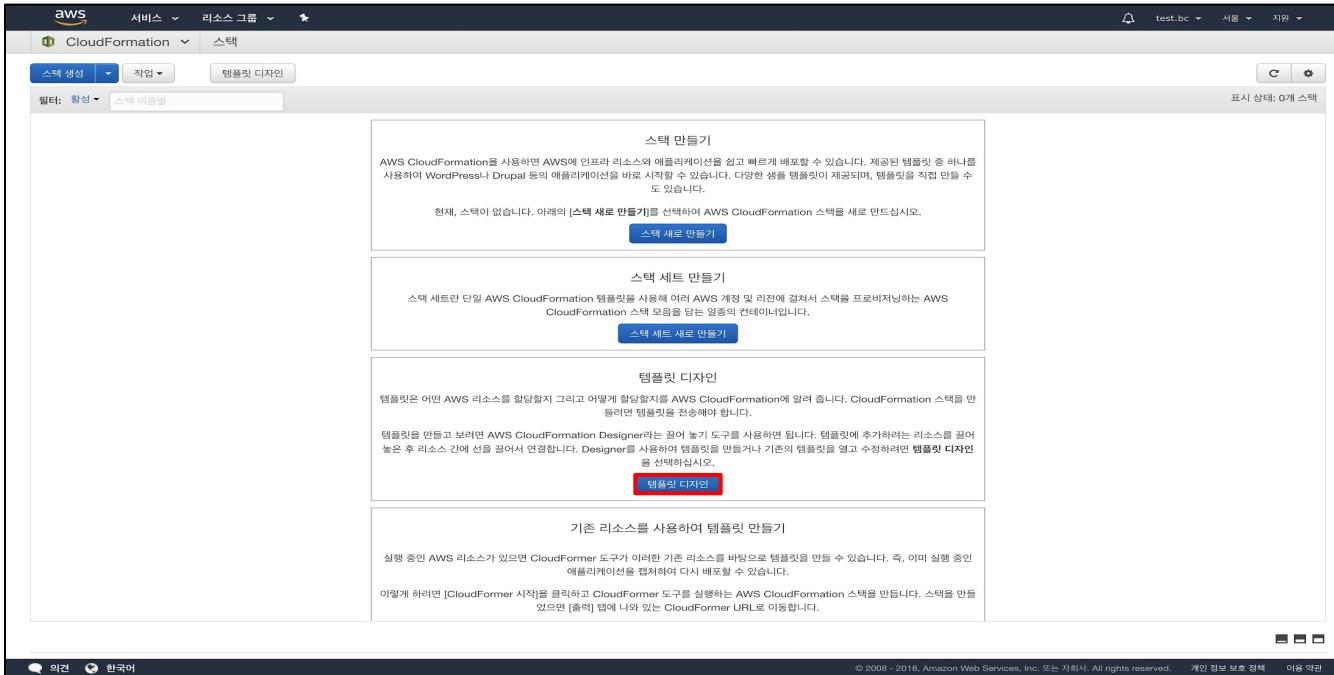
# AWS CloudFormation

## 1. AWS Console ⇒ Service ⇒ CloudFormation



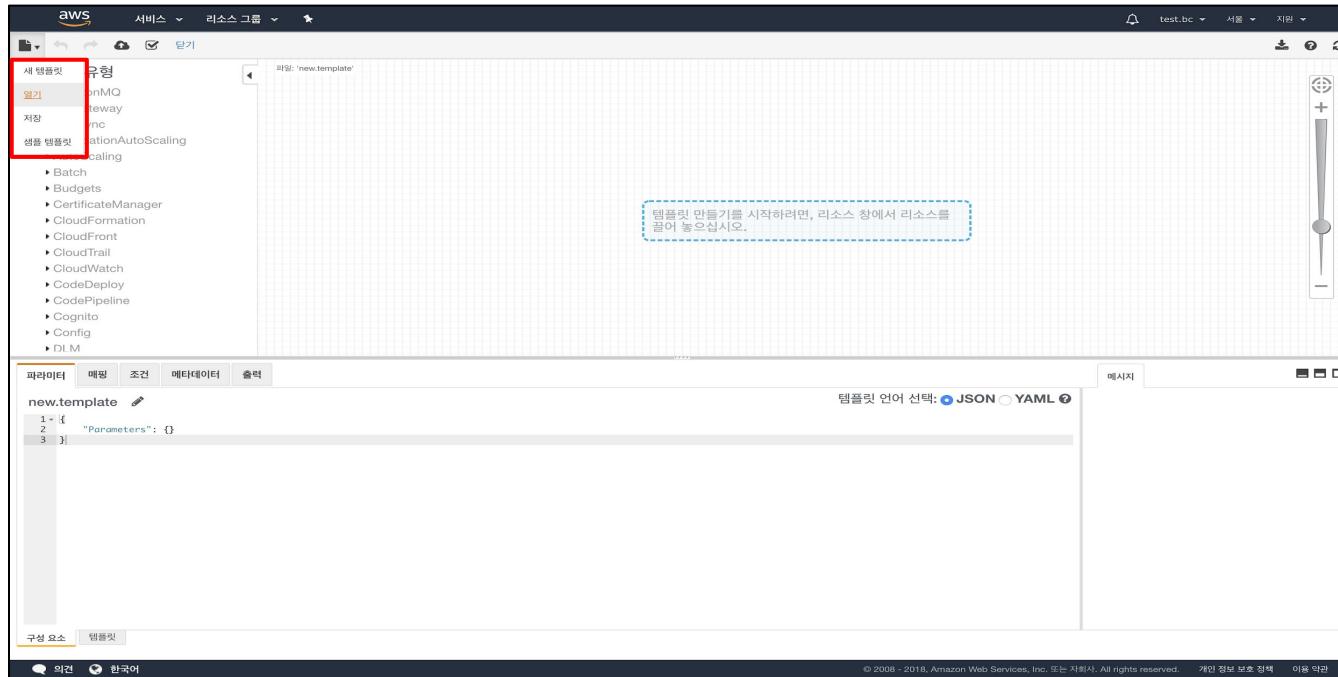
# AWS CloudFormation

## 2. Click Template Design



# AWS CloudFormation

## 3. Open “icon\_dev\_network.template” on local storage



# AWS CloudFormation

## 4. Create AWS Stack

The screenshot shows the AWS CloudFormation console interface. On the left, a sidebar lists various AWS services under '리소스 유형'. The main area displays a network architecture diagram with several EC2 instances (represented by orange squares) connected by lines. Some instances have a red lock icon on them. Below the diagram is the template code for 'icon\_dev\_network.template'.

```
icon_dev_network.template
1: {
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Deploy ICON on AWS",
  "Parameters": {
    "KeyName": {
      "Description": "Name of an existing EC2 KeyPair to enable SSH access to the instance",
      "Type": "AWS::EC2::KeyPair::KeyName",
      "ConstraintDescription": "must be the name of an existing EC2 KeyPair."
    },
    "InstanceType": {
      "Description": "ICON EC2 instance type",
      "Type": "String",
      "Default": "m5.xlarge",
      "AllowEmptyValues": [
        "m2.xlarge",
        "m5.large",
        "m5.xlarge",
        "m5.2xlarge",
        "c5.large",
        "c5.xlarge"
      ]
    }
  }
}
```

The bottom of the screen includes navigation buttons for '설정 모드' and '템플릿', and footer links for '의견', '한국어', '2008-2018, Amazon Web Services, Inc. 또는 그 자회사. All rights reserved.', '개인 정보 보호 정책', and '이용 약관'.

# AWS CloudFormation

## 5. Select CloudFormation Template ⇒ Next

The screenshot shows the AWS CloudFormation console interface. The top navigation bar includes '서비스' (Services), '리소스 그룹' (Resource Groups), and a search bar. Below the navigation is a breadcrumb trail: 'CloudFormation' > '스택' > '스택 생성'. The main content area is titled '스택 생성' (Stack Creation) and '템플릿 선택' (Template Selection). A descriptive text states: '만들리는 스택을 나타내는 템플릿을 선택하십시오. 스택이란 하나의 단위로 관리하는 여러 관련 리소스의 그룹입니다.' (Select the template that represents the stack you want to create. A stack is a group of related resources managed as a single unit.) There are three options for selecting a template:

- 템플릿 디자인**: AWS CloudFormation Designer를 사용하여 템플릿을 만들거나 기존 템플릿을 수정할 수 있습니다. 자세히 알아보기.
- 샘플 템플릿 선택**: 템플릿은 스택 리소스와 해당 속성을 나타내는 JSON/YAML 형식의 텍스트 파일입니다. 자세히 알아보기.
- Amazon S3에 템플릿 업로드**: 파일 선택: 선택된 파일 없음
- Amazon S3 템플릿 URL 지정**: 선택된 URL: <https://s3.ap-northeast-2.amazonaws.com/cf-templates-8ba17y868mik-ap-northeast-2> Designer에서 템플릿 보기/편집

At the bottom right of the page are '취소' (Cancel) and '다음' (Next) buttons. The footer contains links for '의견' (Feedback), '한국어' (Korean), and copyright information: '© 2008 - 2018, Amazon Web Services, Inc. 또는 저희사. All rights reserved.' and links for '개인 정보 보호 정책' (Privacy Policy) and '이용 약관' (Terms of Service).

# AWS CloudFormation

## 6. Edit Configuration

템플릿 선택  
세부 정보 지정  
옵션  
검토

세부 정보 지정

스택 이름

파라미터

AWS Instance and Network Settings

InstanceType: m5.xlarge (ICON EC2 instance type)

NodeCount: 4 (ICON Node Cluster size; must be between 4 and 12)

InstanceName: icon (EC2 Instance Name Prefix)

VpcId: ID 또는 이름 태그 값으로 검색 (VpcId of your existing Virtual Private Cloud (VPC))

SubnetId: ID 또는 이름 태그 값으로 검색 (Subnet should be a public subnet.)

Fee, Audit Parameters

Fee: false (Pay commission to execute the transaction by ICX or not)

Audit: false (Prevent to deploy SCORE by anybody or not)

EC2 Parameters

KeyName: 검색 (Name of an existing EC2 KeyPair to enable SSH access to the instance)

SSHLocation: (The IP address range that can be used to SSH to the EC2 instances)

취소 이전 다음

# AWS CloudFormation

## 7. Finalize Options

템플릿 선택  
세부 정보 지정

**옵션**

태그  
사용자 스택에 대한 태그(키-값 페어)를 지정할 수 있습니다. 각 스택에 대해 최대 50개의 고유한 키-값 페어를 추가할 수 있습니다. 자세히 알아보기

키 (최대 127자)	값 (최대 255자)
1	

권한  
CloudFormation에서 스택에 리소스를 생성, 수정 또는 삭제하는 데 사용할 IAM 역할을 선택할 수 있습니다. 역할을 선택하지 않으면 해당 계정에 정의된 권한을 CloudFormation에서 사용합니다. 자세히 알아보기

IAM 역할: 역할 선택(선택 사용)  
역할 ARN 입력:

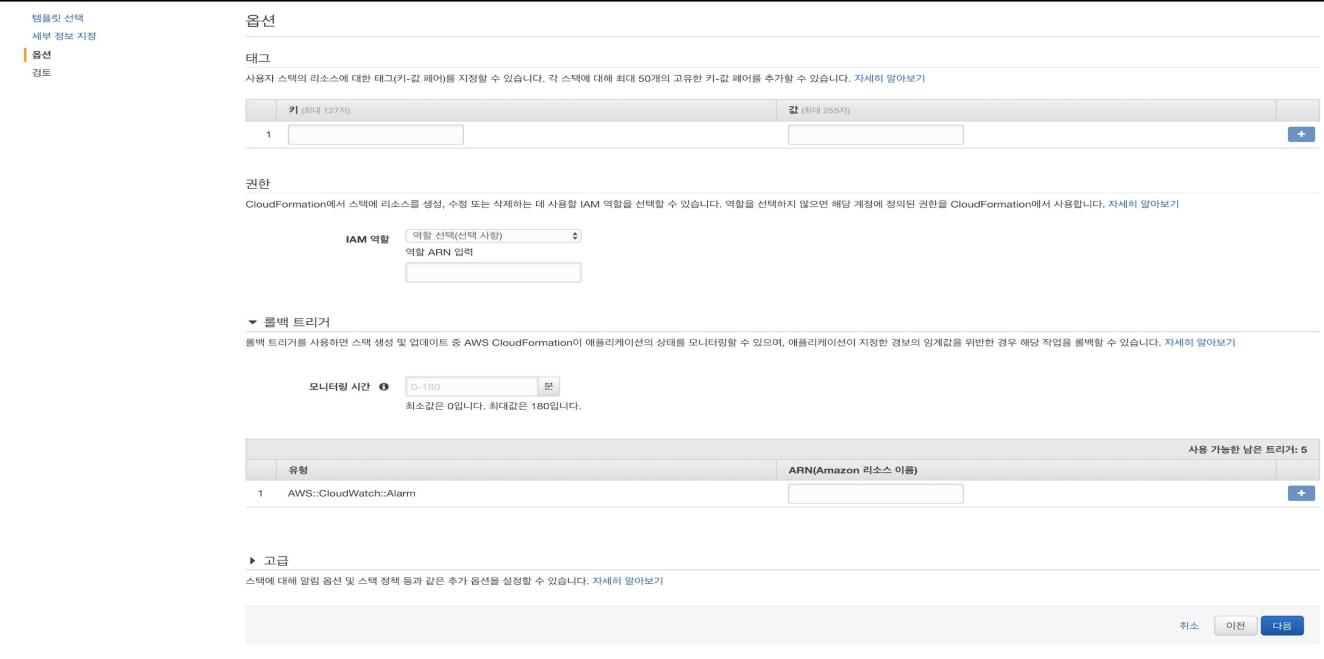
▶ 블랙 트리거  
블랙 트리거를 사용하면 스택 생성 및 업데이트 중 AWS CloudFormation이 애플리케이션의 상태를 모니터링할 수 있으며, 애플리케이션이 지정한 경보의 임계값을 위반한 경우 해당 작업을 블랙될 수 있습니다. 자세히 알아보기

모니터링 시간: 0-180 분  
최소값은 0입니다. 최대값은 180입니다.

유형	ARN(Amazon 리소스 이름)	사용 가능한 남은 트리거: 5
1	AWS::CloudWatch::Alarm	

▶ 고급  
스택에 대해 알림 옵션 및 스택 정책 등과 같은 추가 옵션을 설정할 수 있습니다. 자세히 알아보기

취소 이전 다음



# AWS CloudFormation

## 8. Check Loaded Instances

The screenshot shows the AWS CloudFormation Instances page. On the left, there's a sidebar with navigation links like 'AWS 대시보드', '이벤트', '태그', '보고서', '제한', '인스턴스' (selected), 'Launch Templates', '스팟 요청', '예약 인스턴스', '전용 호스트', and 'Capacity Reservations'. Below that are sections for '이미지', 'AMI', '번들 작업', 'ELASTIC BLOCK STORE', '볼륨', '스냅샷', 'Lifecycle Manager', '네트워크 및 보안', '보안 그룹', '탄력적 IP', '배치 그룹', '키 페어', '네트워크 인터페이스', '로드 밸런서', '대상 그룹', 'AUTO SCALING', '시작 구성', 'Auto Scaling 그룹', and 'SYSTEMS MANAGER 서비스', '명령 실행'.

The main content area displays a table of EC2 instances:

Name	인스턴스 ID	인스턴스 유형	가용 영역	인스턴스 상태	상태 검사	경보 상태	파블릭 DNS(IPv4)	IPv4 퍼블릭 IP	IPv6 IP
loopchain_icon_monitoringserver	i-011728970ec386edf	m5.xlarge	ap-northeast-2a	running	2/2 검사 통과	없음	ec2-13-209-18-38.ap-n...	13.209.18.38	-
loopchain_icon_peer#2	i-026289595aae0ceca88	m5.xlarge	ap-northeast-2a	running	2/2 검사 통과	없음	ec2-13-125-79-254.ap...	13.125.79.254	-
loopchain_icon_peer#4	i-02d5c2a66fb52017f	m5.xlarge	ap-northeast-2a	running	2/2 검사 통과	없음	ec2-13-124-230-127.ap...	13.124.230.127	-
loopchain_icon_radiostation	i-03122488639e54d...	m5.xlarge	ap-northeast-2a	running	2/2 검사 통과	없음	ec2-13-209-16-24.ap-n...	13.209.16.24	-
loopchain_icon_logserver	i-03fa30e9521cc2eed	m5.xlarge	ap-northeast-2a	running	2/2 검사 통과	없음	ec2-52-79-170-120.ap...	52.79.170.120	-
loopchain_icon_peer#3	i-0438d8487ea67d...	m5.xlarge	ap-northeast-2a	running	2/2 검사 통과	없음	ec2-54-180-135-65.ap...	54.180.135.65	-
loopchain_icon_peer#1	i-0464ebc4c18c06e2d	m5.xlarge	ap-northeast-2a	running	2/2 검사 통과	없음	ec2-13-125-122-192.ap...	13.125.122.192	-

At the bottom, it says '위에서 인스턴스를 선택하십시오'.

# 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: 16009198000000000000000000000000
-----
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: 16009198000000000000000000000000
-----
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: 16009198000000000000000000000000
-----
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: 16009198000000000000000000000000
-----
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/B07KBTZHJD?qid=1544602499452&sr=0-1&ref\\_=srh\\_res\\_product\\_title](https://aws.amazon.com/marketplace/pp/B07KBTZHJD?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
  - t-bears
  - icon-sdk-python
  - icon-sdk-java
  - icon-sdk-js
  - iconex\_android
  - iconex\_ios
  - iconex\_chrome\_extension
- Node
- Dev tools

The screenshot shows the GitHub organization page for the ICON Foundation. At the top, there's a banner for the ICON Foundation with the tagline "Hyperconnect the world". Below the banner, it shows 21 repositories, 40 people, 0 teams, and 0 projects. There are filters for "Type: All" and "Language: All", and a "New" button. The main content area lists three repositories:

- icon-project.github.io**: Updated 2 hours ago, 12 stars, 4 forks.
- loopchain**: Updated 6 days ago, 15 stars, 4 forks.
- icon-sdk-python**: Updated 7 days ago, 15 stars, 2 forks.

On the right side, there are sections for "Top languages" (Python, JavaScript, Java, Swift, HTML) and "People" (a grid of user profiles and icons).

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

- Community portal for ICON DApp ecosystem

## Getting Started

Tutorials for developers to get started

## SCORE

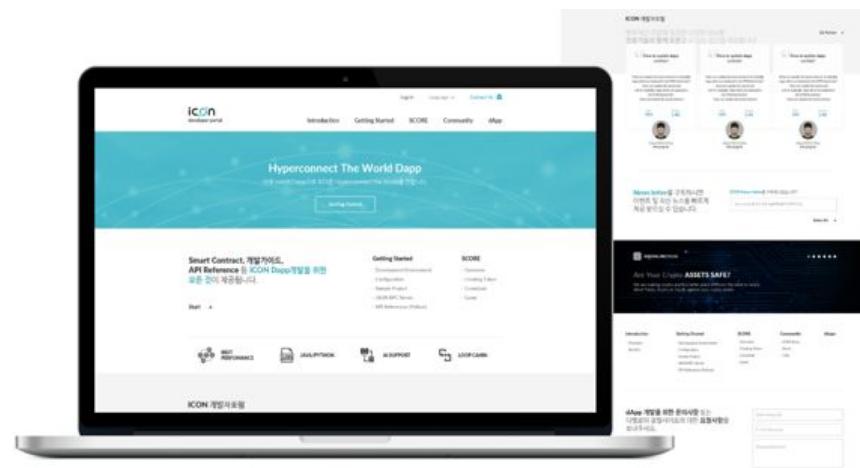
Details on ICON's Smart Contract, SCORE

## Community

Forum for Korean/English developers to discuss and communicate

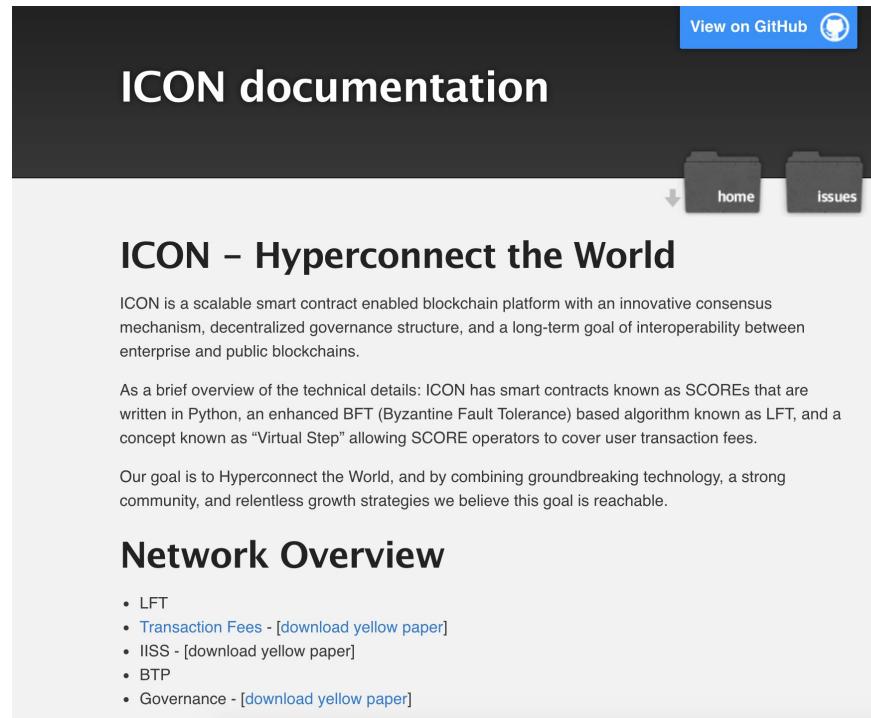
## DApp

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.



The screenshot shows the homepage of the ICON documentation. At the top right is a "View on GitHub" button with a GitHub icon. Below it is the title "ICON documentation". To the right are navigation links for "home" and "issues". The main content area has a dark header with the text "ICON – Hyperconnect the World". Below the header, there is a brief introduction: "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." It also mentions 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." The goal is described as "Hyperconnect the World, and by combining groundbreaking technology, a strong community, and relentless growth strategies we believe this goal is reachable." A section titled "Network Overview" lists several key components: LFT, Transaction Fees (with a link to a yellow paper), IISS (with a link to a yellow paper), BTP, and Governance (with a link to a yellow paper).

**ICON documentation**

**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 have been more than 2 weeks in Last Call without any technical changes or objections enter 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