



# Multichain Cloud DApp IDE



# CONTENT

01

## Overview

ChainIDE Overview

02

## Functionalities

Features and functionalities of ChainIDE

03

## Services and Ecosystem

Supporting services and  
the ecosystem of ChainIDE

04

## Future Plan

Roadmap of ChainIDE

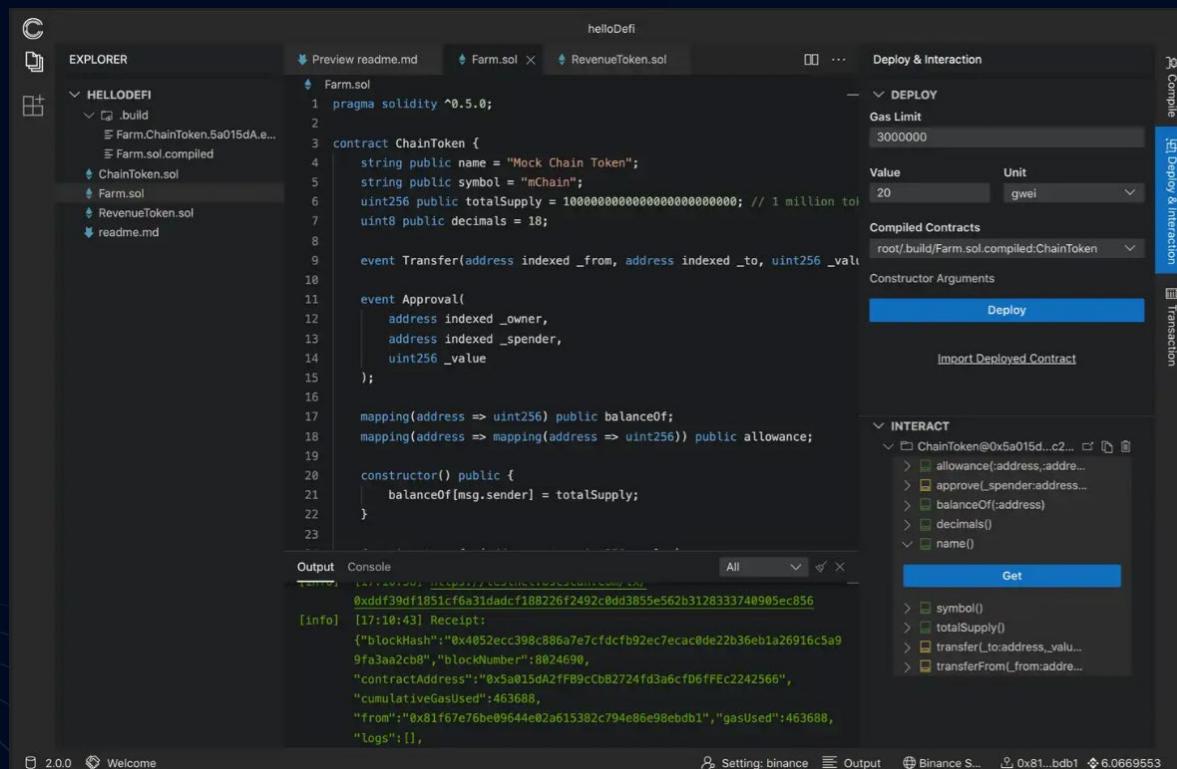


# PART 01

## Overview of ChainIDE



# What is ChainIDE



A cloud blockchain IDE for smart contracts and DApps, which can be deployed on Polygon Ethereum, BSC, Dfinity, Conflux, Flow equivalent or heterogeneous blockchains.

01

Simple Interfaces

02

Multichain support

03

Web-based

# ChainIDE Features



Cloud-based: writing a DApp in the browser

Development, debugging, and deployment in one place

Multichain support

Consortium:



Public:



# Product Comparison

01

## Remix

- Simple and easy to use; browser-based; one-click deployment and debugging.
- Rich features for Solidity
- EVM only IDE
- No backend, running in the browser (e.g. cannot use npm/node runtime)

02

## VSCode

- Most popular desktop IDE
- Biggest user-base, a lot of useful plugins
- For general purpose
- Requires complicated configuration before starting the development

# ChainIDE wants to bring



## Easy Development

Developers can easily use ChainIDE for development. Users, regardless of whether having a solid development background or not, can easily build complex functional applications through IDE templates.



## Customization

ChainIDE provides customized features for different blockchains to meet developers' needs in different scenarios.



## Ecosystem

Besides IDE, we want to bring more useful services and hold more activities help more developers to build and incubate their applications.

# PART 02

ChainIDE's functionalities

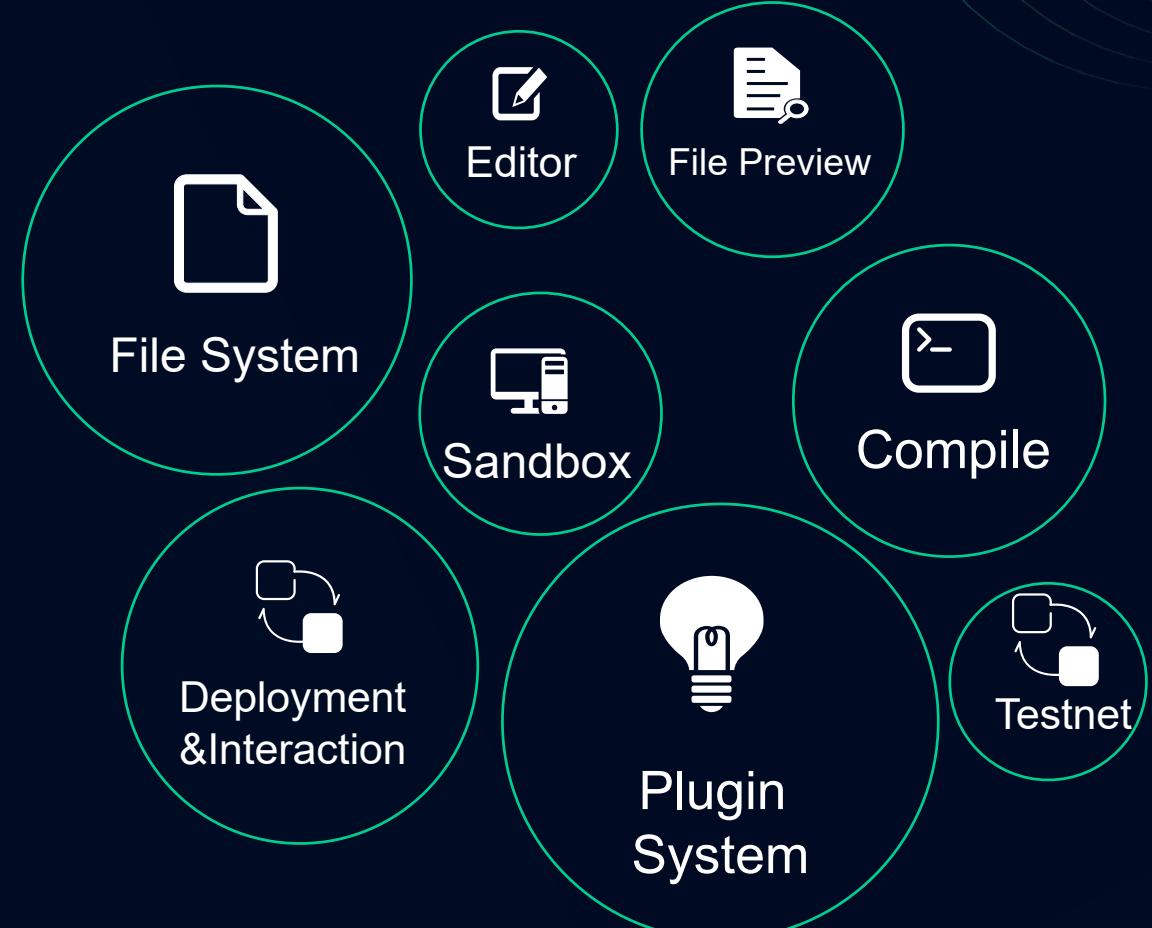
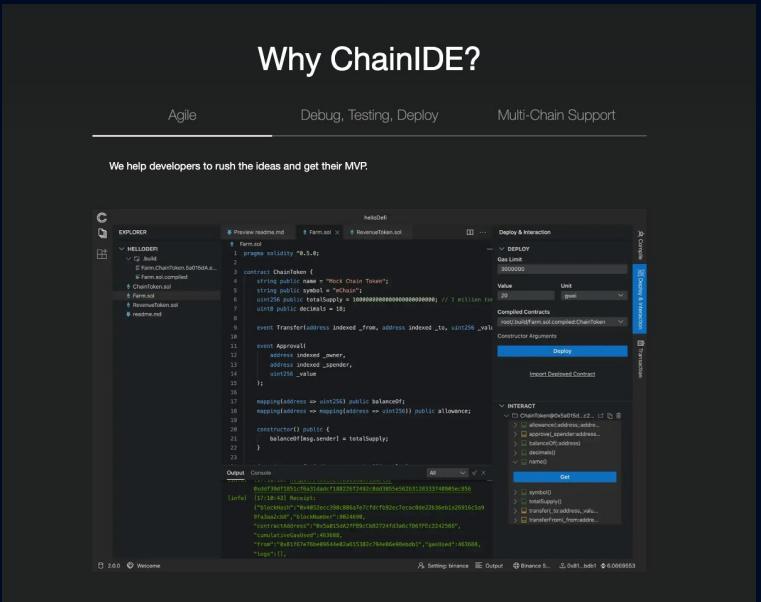


# Functionalities

Swift

Simple

Smart



# File System

01

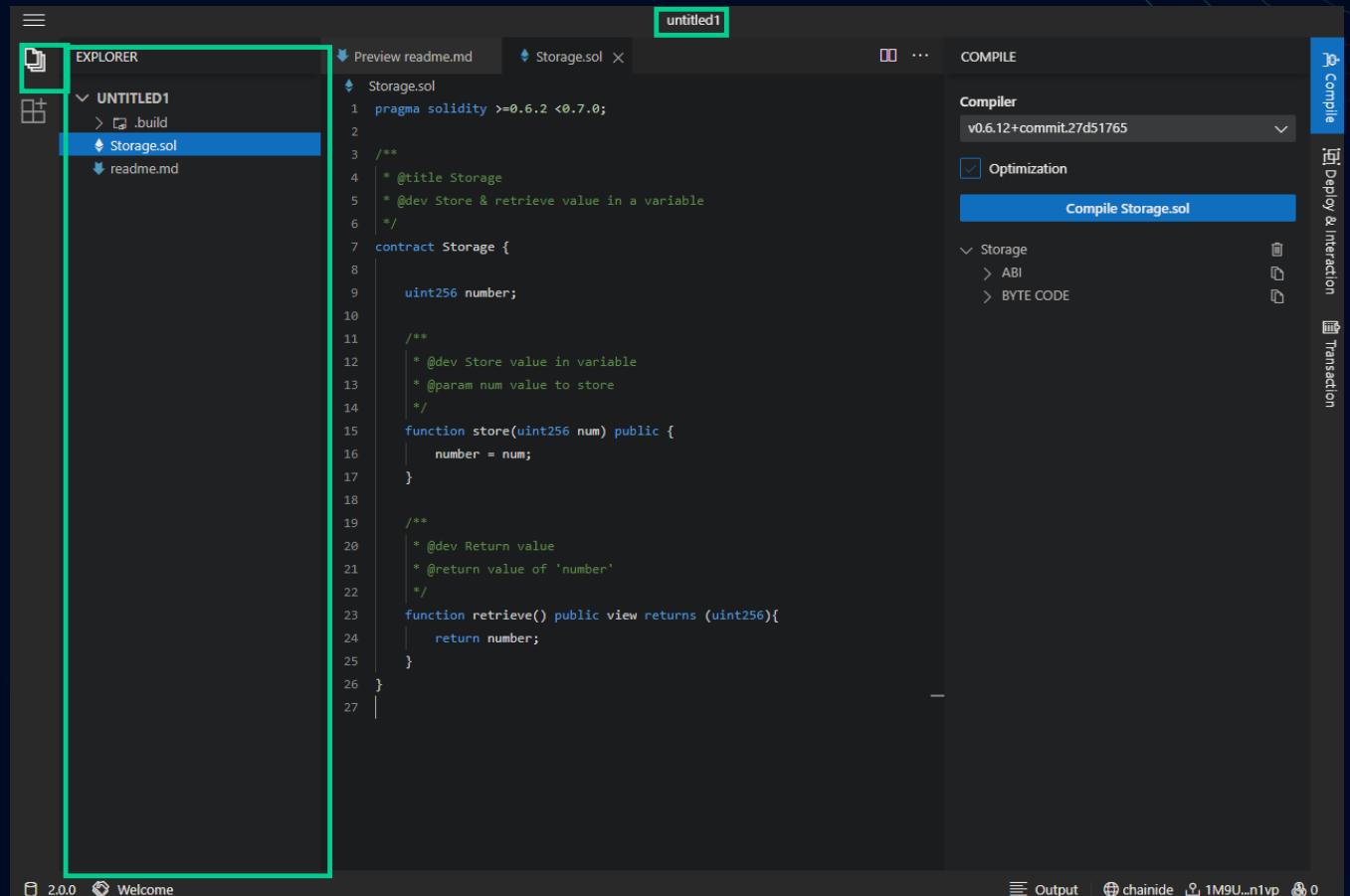
File Explorer

02

Create, Rename, Update,  
Deletion

03

Import and download the  
whole project



The screenshot shows the ChainIDE interface with the following components:

- EXPLORER** (left sidebar): Shows a tree view of the project structure under "UNTITLED1". The "Storage.sol" file is selected and highlighted in blue.
- CODE** (center): A code editor window titled "Storage.sol" containing Solidity code for a storage contract. The code defines a `Storage` contract with a `number` variable and two functions: `store` and `retrieve`.
- COMPILER** (right sidebar): Shows the compiler version as "v0.6.12+commit.27d51765" and an "Optimization" checkbox. A blue button labeled "Compile Storage.sol" is highlighted.
- Storage** (bottom right): A section showing ABI and BYTE CODE for the deployed contract.
- Bottom Bar**: Includes links for "Output", "chainide", and a transaction hash "1M9U...n1vp".

# Editor

01 Split Tabs

02 Syntax highlighting

03 Auto-save

The screenshot displays the ChainIDE Editor interface. On the left, the Explorer sidebar shows a project structure with files: IntegerCalc.sol, Storage.sol, and readme.md. The main area contains two tabs: IntegerCalc.sol and Storage.sol. The IntegerCalc.sol tab shows Solidity code for a contract with methods calculate, getAdd, getMin, getMul, and getDiv. The Storage.sol tab shows a contract with a store function and a retrieve function. The right side of the interface includes a COMPILER section with a Compiler dropdown set to v0.6.12+commit..., an Optimization checkbox, and a Compile Integer... button. Below the compiler are tabs for Deployment & Interaction and Transaction. At the bottom, there are status indicators for version 2.0.0, Welcome, Output, chainide, and a file count of 0.

```
pragma solidity >=0.4.2;
contract IntegerCalc{
    uint addResult;
    uint minResult;
    uint mulResult;
    uint divResult;

    function calculate(uint x, uint y) public{
        addResult = x + y;
        minResult = x - y;
        mulResult = x * y;
        divResult = x / y;
    }

    function getAdd() public view returns(uint)
    {
        return (addResult);
    }

    function getMin() public view returns(uint)
    {
        return (minResult);
    }

    function getMul() public view returns(uint)
    {
        return (mulResult);
    }

    function getDiv() public view returns(uint)
    {
        return (divResult);
    }
}

contract Storage {
    uint256 number;

    /**
     * @dev Store value in variable
     * @param num value to store
     */
    function store(uint256 num) public {
        number = num;
    }

    /**
     * @dev Return value
     * @return value of 'number'
     */
    function retrieve() public view returns (uint)
    {
        return number;
    }
}
```

# Preview

01

Interactive

02

Supported languages:  
HTML, JS, JSX, Markdown

03

React/Vue and Node  
support incoming.

The screenshot shows a blockchain development environment with the following components:

- EXPLORER:** Shows a project structure under "UNTITLED2" with files like "index.html", "ContentMixin.sol", "ERC721Base.sol", "Initializable.sol", "NativeMetaTransaction.sol", "Creature.sol", "ERC721Tutorial.md", "ERC721Tradable.sol", "ERC721部署教程.md", and "README.md".
- Code Editor:** An open file "index.html" containing the following code:

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<meta http-equiv='X-UA-Compatible' content='IE=edge'>
<meta name='viewport' content='width=device-width, initial-scale=1.0'>
<style>
input {
    display: block;
    clear: both;
    margin: 10px 0;
}
</style>
<script type="text/javascript" src="https://cdn.jsdelivr.net/npm/react@16.13.1/umd/react.development.js"></script>
<script type="text/javascript" src="https://cdn.jsdelivr.net/npm/react-dom@16.13.1/umd/react-dom.development.js"></script>
</head>
<body>
<h2>ERC721 showcase</h2>
<label for="avatar">Name :</label>
<input id="demo-name" />
<label for="description">Description :</label>
<input id="demo-des" />
<label for="image">Choose a picture:</label>
<input type="file" id="demo-image" accept="image/*" />
<button onclick="onSubmit();">submit</button>
<div id="tokenURIs"></div>
<label for="abi">ABI :</label>
<div><textarea id="abi" rows="5" cols="30" ></textarea>
</div>
</body>

```

- Preview:** A right-hand panel titled "ERC721 showcase" with fields for "Name", "Description", "Choose a picture" (with a file input), "submit", "ABI" (with a text area), and "Contract Address".
- Toolbar:** On the far right, there are buttons for "Compile", "Deploy & Interaction", and "Transaction".

# Log Console

01

Multiple log windows

02

Blockchain related logs

03

Interactive Shell + Sandbox incoming

The screenshot shows the ChainIDE interface with the 'Log Console' tab active. The left side features an Explorer panel with project files like IntegerCalc.sol, Storage.sol, and readme.md. The main area displays a code editor with Solidity smart contract code for integer calculations. To the right is a 'COMPILER' sidebar with options for 'Compiler' (v0.6.8+commit.0...), 'Optimization' (checked), and 'Compile Integer...' button. Below the code editor is an 'Output' section with a 'Console' tab showing log messages:

```
[info] [15:10:58] Compiling contract
[info] [15:11:10] Compile contract success
[info] [15:11:28] Compiling contract
[error] [15:11:29] Compile contract failed: ["fs://b8fe649d-dfa6-4b82-9e76-8278b24d6 IntegerCalc.sol:27:5: ParserError: Expected ';' but got '\n' \n ^\n"]
```

The 'Output' section includes dropdowns for 'All', 'Compile', 'Deploy', and 'Interaction'. At the bottom, there's a footer with version information (2.0.0) and a welcome message.

# Compiler (Solidity)

01

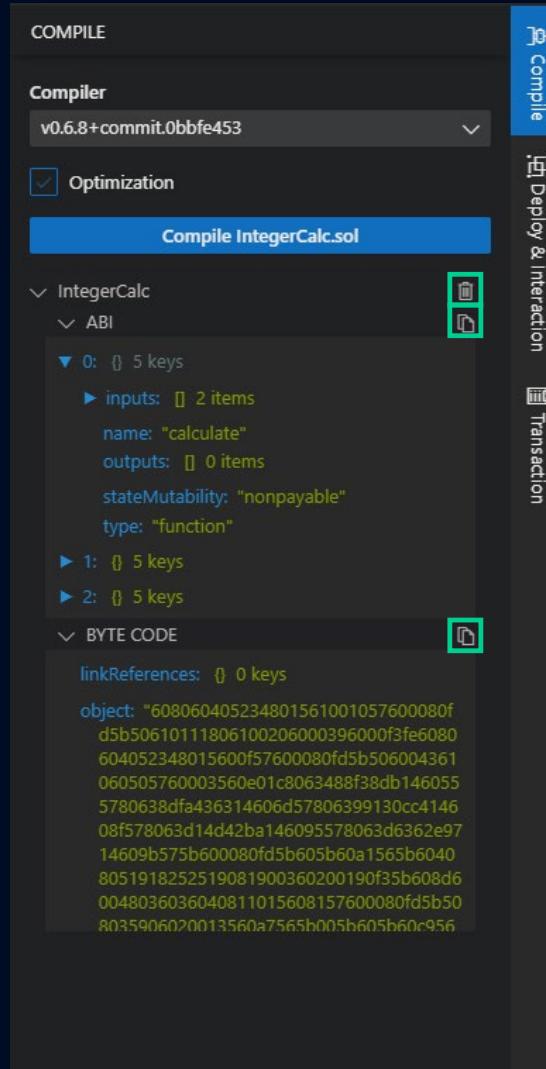
Multi-version compiler

02

Contract Info:

1.ABI

2.Byte Code

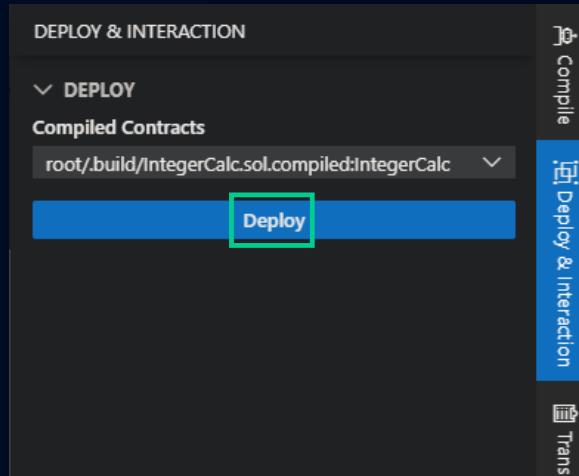


# Deployment & Interaction

01

## Deployment

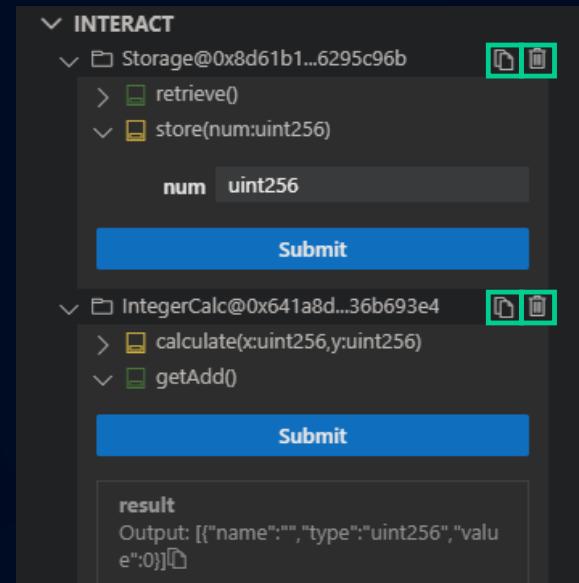
1. deploy to designated blockchain networks
2. Importing external deployed contracts.



02

## Interaction

1. Invoke functions with ABI and address
2. View the Output



# Plugin System

01

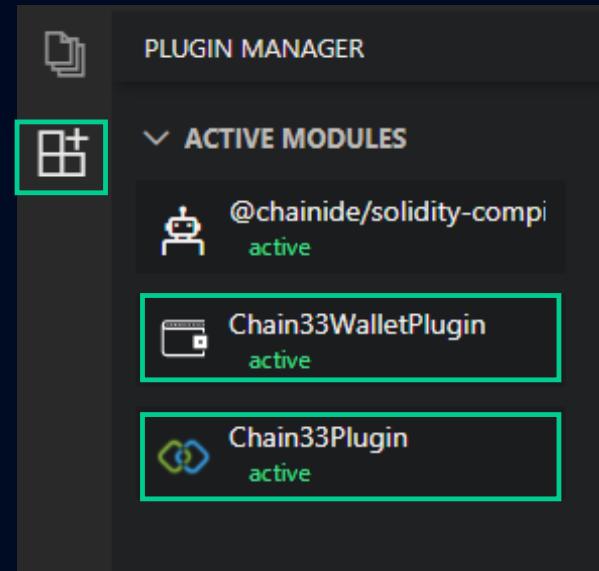
Official and community plugins

02

Different plugins for different blockchains and use-cases

03

Open-source plugin templates.



# PART 03

## Services and Ecosystems



# Templates Market

## Templates



Storage

V1.0.0

This template constructs a skeleton contract that is used to persist values on the blockchain.



ERC20 Showcase

V1.0.0

ERC-20 has emerged as technical standard used for all smart contracts on the Ethereum blockchain for token implementation.



Solidity Samples

V1.0.0

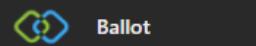
This is the developer list helping developers to get started with Ethereum blockchain.



Ownership

V1.0.0

This template constructs a skeleton contract that models the ownership on the blockchain.



Ballot

V1.0.0

This template constructs a skeleton contract that has "voting with delegation" feature



ERC721 Showcase

V1.0.0

ERC-721 is a free, open standard that describes how to build non-fungible or unique tokens on the Ethereum blockchain.

Provide templates such as Storage, ERC20 Showcase, ERC721, voting system, etc.,

# NFT Game Tutorial

The screenshot shows a web browser window with the following details:

- EXPLORER** sidebar: Shows a project structure under 'UNTITLED2': common, meta-transactions, Creature.sol, ERC721-Tutorial.md, ERC721Tradable.sol, ERC721部署教程.md, README.md, and index.html.
- untitled2** tab: Shows 'Preview ERC721-Tutorial...' and 'ERC721Tradable.sol'.
- Content Area:**
  - ERC-721 Deployment Tutorial**
  - What is ERC-721?**

ERC-721 is a token standard on EVM. This token standard can generate a non-homogeneous token (NFT) and can provide operational support for tracking and transferring NFTs (the token standard of NFT also includes ERC-1155, etc.). This token standard can support the needs of personal transactions and usage and can support application scenarios where NFTs are entrusted to third-party organizations for operation. We have considered the diversity of NFT storage assets and can be detailed by adding different parameters describe and serve as a search index.
  - Prerequisite**

We have created the basic ERC-721 template for the users, the HTML files needed for deployment and upload, but before starting, please download the MetaMask.
  - 1. MetaMask**

If you don't know how to set up the MetaMask environment, you can learn by clicking [here](#). The Ethereum network recommended for this test is [Rinkeby Test Network](#), you can get the gas fee through the process of this website(Open sea testnet only supports Rinkeby Test Network).
- Rinkeby Authenticated Faucet Screenshot:**

A screenshot of the Rinkeby Authenticated Faucet interface. It shows a button labeled 'Give me Ether' with a dropdown menu showing options like '3 Ether / 8 hours', '7.5 Ether / 1 day', and '18.75 Ether / 3 days'. Below the button, there's a section titled 'How does this work?' with instructions for requesting funds via Twitter or Facebook.
- Bottom Navigation:**

Version 2.0.0, Welcome, Output, chainide, 1M9U..n1vp, 0

- 01 ERC721 Introduction
- 02 NFT Tutorial
- 03 Full front and backend tutorial of building a NFT Game.

# MoveCastle Interactive Tutorial

ChainCastle ← Back

## Chapter 1 Introduction

In this chapter, we will learn how to build a castle.

- We will use modules to build a new castle
- Our castle will be stored in the database, i.e., Libra Blockchain

In the following, we will add more functions to our castle, such as enhancing its power, fighting with other castles. But before that, we should first realize the function of creating a castle.

### What are the properties of the castle?

Each castle has its name, serial number, level, economic and military powers. The serial number determines the appearance of the castle and the race of creatures in the castle. The level determines the value of the castle. The economic and military powers affect the performance of the castle when interacting with other castles.

#### How does the serial number affect the castle?

The appearance of a castle and its race are decided by an 8-digit integer, which is named as the serial number, such as 83451029.

Each digit of the serial number corresponds to a property of the castle. In particular,

1	2	3	4	5	6	7	8
size	style	color	logo style	logo color	logo position	race	special ID

The name of each newly built castle is input by the user; the serial number is randomly generated according to the depth of the current block; the level is initialized as 1; the initial economic and military powers are determined by the race digit in the serial number. The corresponding relationships are defined as follows:

Race digit	0, 5	1, 6	2, 7	3, 8	4, 9
Economic power	6	8	3	7	4
Military power	6	3	8	4	7

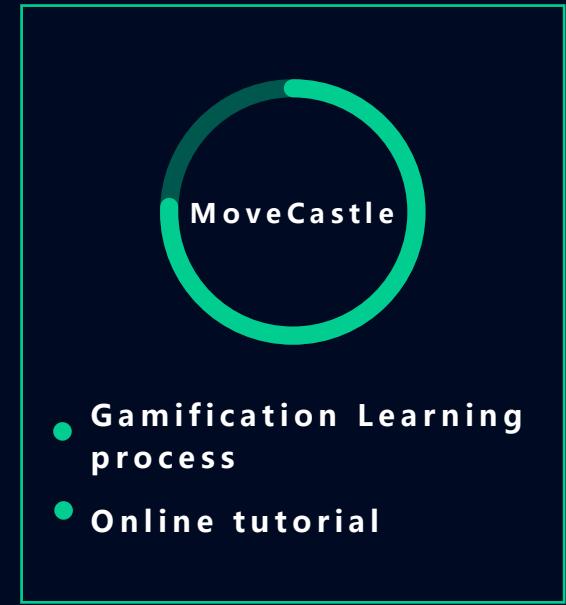
**Practical Exercise**

1. Please drag the slider on the right side of the page to explore the appearance of the castle corresponding to a specific serial number.

With the above basic knowledge, you can click the Next Chapter button to continue learning Move.

LEVEL 1  
Serial Number  
**00000000**  
LITTLE  
Special ID: 0  
Human RACE    6 ECONOMIC    6 MILITARY

Castle Size: 0  
Castle Style: 0  
Castle Color: 0  
Logo Style: 0  
Logo Color: 0  
Logo Position: 0  
Race: 0  
Special ID: 0



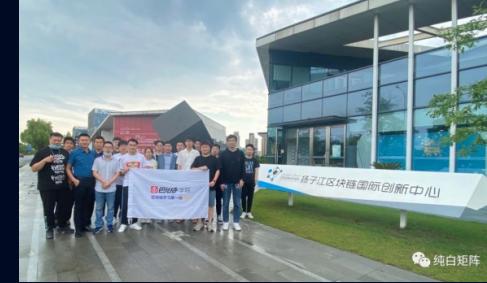
# ChainIDE Forum



- Forum for discussion and
- Weekly online seminars
- Application ideas

# Developer Activities

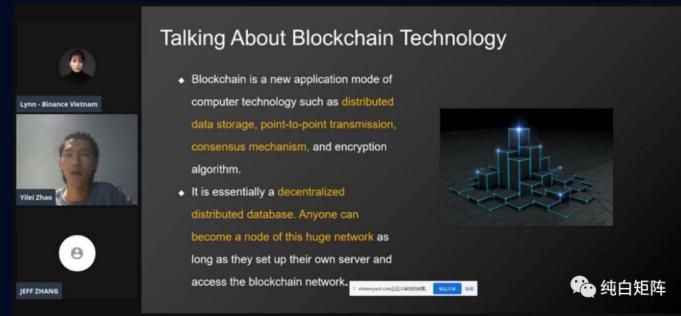
## Offline Activities



## Online classes

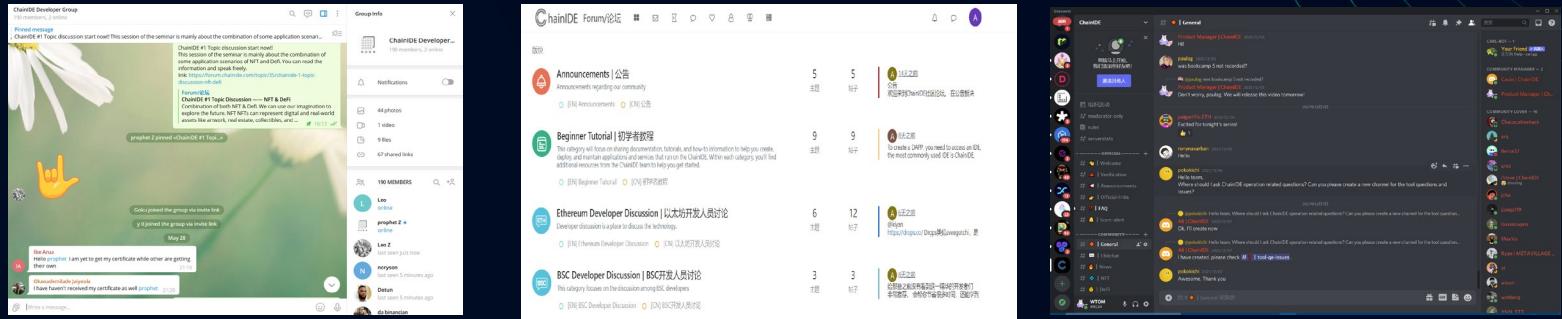


## Hackerthon



# Developer Ecosystem

## Forum & Community



## Partners



## Social Media

Twitter, Medium, Youtube, Tiktok

# Supporting Services/Solutions



**ChainBase** – Managed DApp frontend and backend hosting solution:

- Frontend hosting;
- Backend hosting;
- NFT media hosting and CDN;
- Auth and Identity;

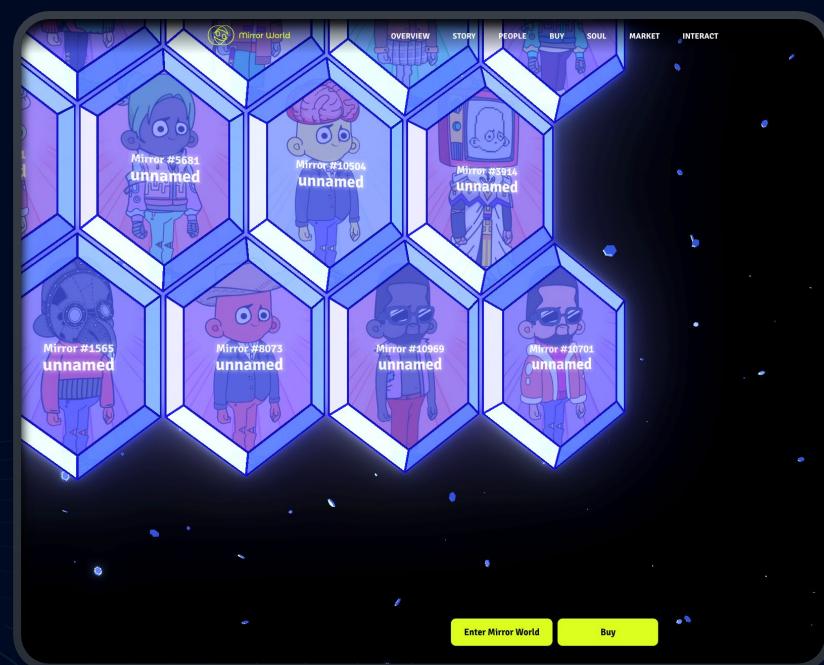


**ChainSync** – Managed data synchronization and enhancement service:

- Sync blockchain events and onchain data to offchain store;
- Enhance synced data;
- Trigger offchain function/service call via onchain events;

# Products built with ChainIDE

- RIVERMEN
- MIRROR WORLD
- MATRIX WORLD



# PART 4

Towards the Future



# Towards Web 3.0 Services

01

Everything on the Cloud

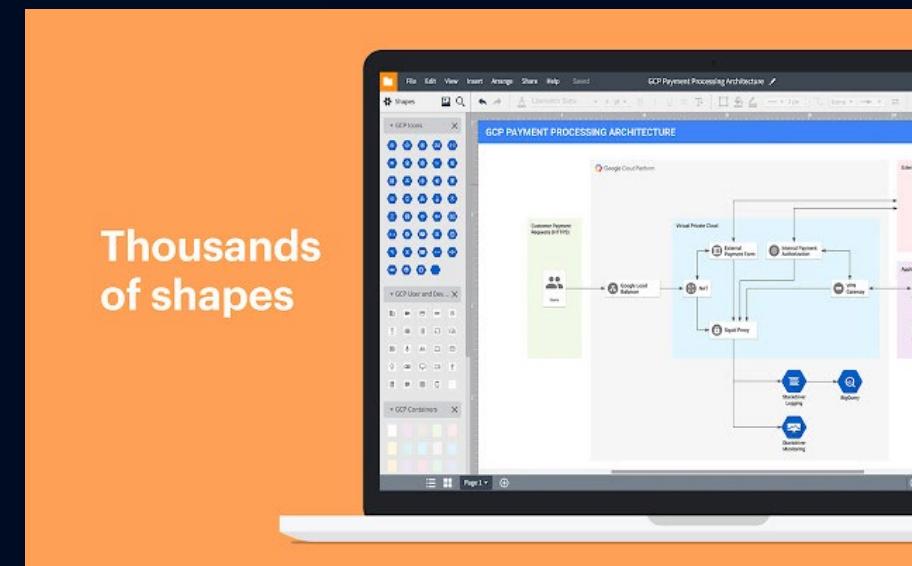
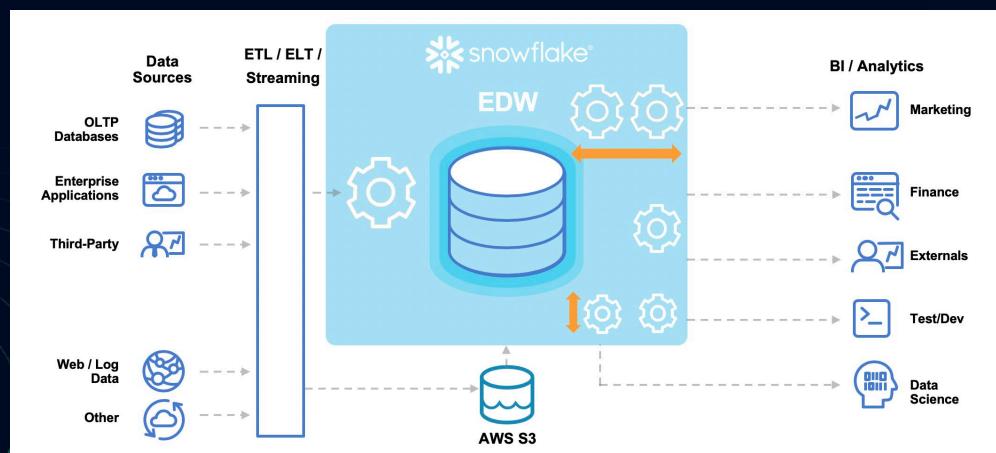
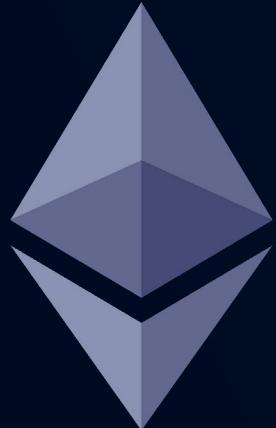
02

Infra-less

03

Decentralization/Multi-central, more  
interactions and connections

# Infra-less



# Infra-less in ChainIDE -- Sandbox

01

Skip the configuration of  
env, infra, servers.

02

Frontend, backend, and  
contract development in  
one place

Node Runtime

Geth CLI

Truffle CLI

NFT Toolbox

Blockchain Node

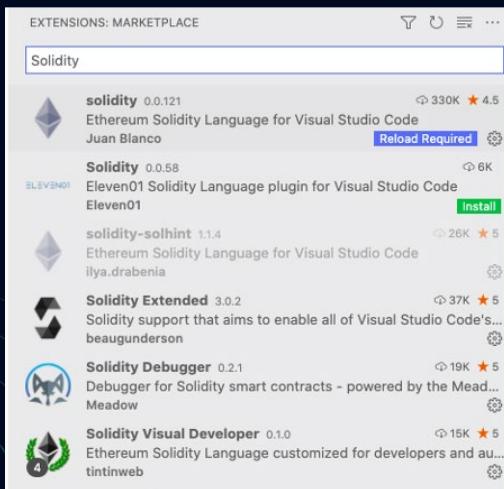
The screenshot shows the ChainIDE interface with the following components:

- EXPLORER:** Shows the project structure with files like package.json, README.md, and .gitignore.
- Code Editor:** Displays the content of README.md, which includes a preview of the page and some shell commands for cloning the repository and building the SDK.
- Terminal:** Shows the output of running `yarn compile:contract`, which successfully compiles four contracts (CellEvolutionNewWorld, SafeMath, and Strings) into artifacts of sizes 15.93, 0.08, and 0.08 KB respectively.
- Deploy & Interaction:** A sidebar with tabs for DEPLOY, INTERACT, and TRANSACTION. The DEPLOY tab is active, showing fields for Gas Limit (3000000), gas price (0 gwei), and a Deploy button.
- Buttons:** Various buttons for actions like 编译合约 (Compile Contract), 发布合约 (Deploy Contract), 打包SDK (Build SDK), and 开发DAPP (Develop DAPP).

# ChainIDE Connection Project (Coming Soon)

## VSCode extension integration

- VSCode plugin compatible
- Compatible with VSCode theme
- First party VSCode plugin Port
- ChainIDE front-end open source project



## Front-end and back-end hosting integration

- Cloudflare, Netlify integration
- Fleek integration
- IPFS integration
- AWS, GCP, Azure integration,
- Infura, Alchemy integration



## NFT Toolbox

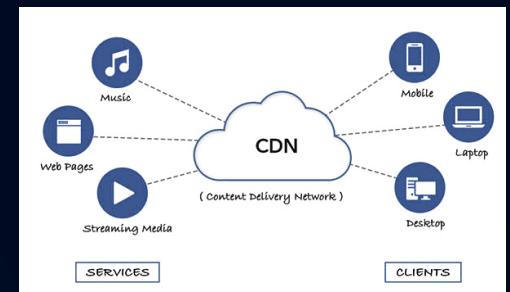
- NFT market integration
- NFT Metadata Backend template
- NFT Contract template
- NFT persistent storage hosting, CDN



OpenSea

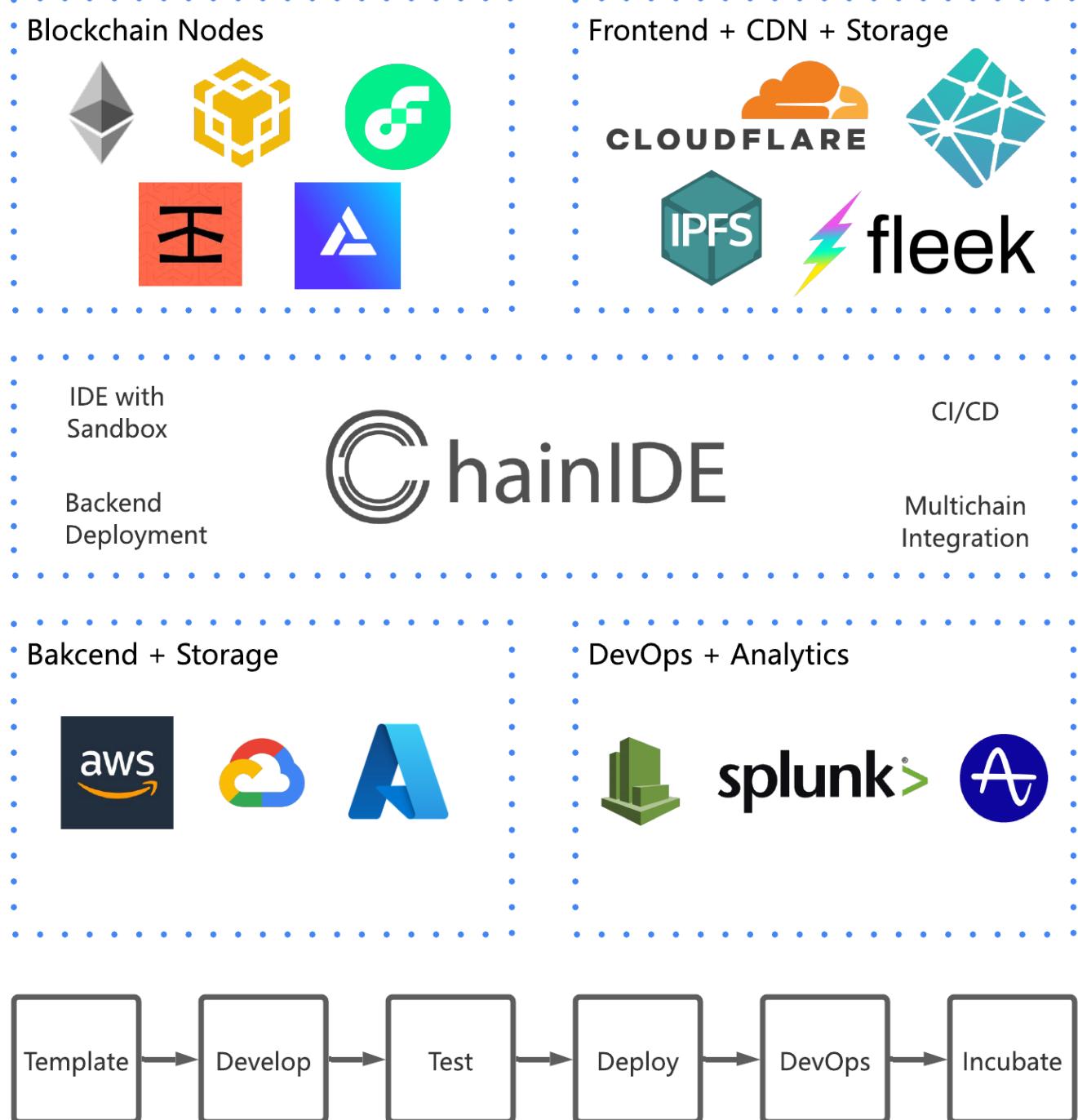


Rarible



# ChainIDE

-- Web 3.0  
Development  
Platform



# ChainIDE DApp Development Showcase

<https://chainide.com/>

# THANKS

ChainIDE team looks forward to your  
use and feedback!

