CESS Network

The Decentralized Data Infrastructure

Episode 8: DApp Development using Ink! Smart Contract

https://www.cess.network

Course Logistics

Course Website: https://course.cess.network/

Episode 1	CESS Network Introduction
Episode 2	CESS Architecture & Key Technologies
Episode 3	CESS Ecosystem and Applications
Episode 4	CESS Nodes & CESS Account Setup
Episode 5	Demo: Running a Consensus Node
Episode 6	Demo: Running a Storage Node
Episode 7	CESS DeOSS and DeOSS REST API
Episode 8	dApp Development using ink! Smart Contract
Episode 9	dApp Development using Solidity Smart Contract
Episode 10	Building Custom Pallet

Why Choose Ink! Smart Contract?

CESS

Access to Polkadot Ecosystem

CESS developers can leverage the extensive network and resources of the Polkadot ecosystem.

WebAssembly (WASM) Compatibility

Ink! Compiles to WASM, which is supported by various blockchains, allowing cross-chain deployment

Enhanced Security

Developers benefit from the security features of Rust, minimizing the risk of smart contract vulnerabilities.

Developer-Friendly Environment

With robust tooling, documentation and support Ink! Attracts wide pool of developers.

Performance and Efficiency

Ensures high performance and efficient execution of smart contracts on CESS

Advanced Features

Offers upgradable contracts and modular design for flexible and maintainable DApps on CESS

Ink! Smart Contract Dev Prerequisites

- Install Rust & Cargo

```
curl https://sh.rustup.rs -sSf | sh
```

- Install Ink! CLI

```
rustup component add rust-src
cargo install --force --locked cargo-contract
```

Flipper Smart Contract Walkthrough

A simple Flipper smart contract that flips a boolean value from true to false and vice versa

- Create a Smart Contract

```
cargo contract new flipper
cd flipper
```

```
flipper
  └─ lib.rs                <-- Contract Source Code
  └─ Cargo.toml            <-- Rust Dependencies and ink! Configuration
  └─ .gitignore
```

Lib.rs Structure

```rust
#![cfg_attr(not(feature = "std"), no_std, no_main)]

#[ink::contract]
pub mod flipper {
...
}
```

```rust
#[ink(storage)]
pub struct Flipper {
    value: bool,
}
```

Macros

cfg_attr: Rust attribute that conditionally configures attributes indicating the project to be *no_std* or no standard library with *no_main* for non-main function execution.

ink::contract: Marks the beginning of the Ink smart contract module making flipper module as the ink smart contract

ink(storage): Specifies the storage layout of the contract.

Flipper Implementation

```rust
impl Flipper {
    /// Creates a new flipper smart contract initialized with the given value.
    #[ink(constructor)]
    pub fn new(init_value: bool) -> Self {
        Self { value: init_value }
    }

    /// Creates a new flipper smart contract initialized to `false`.
    #[ink(constructor)]
    pub fn new_default() -> Self {
        Self::new(Default::default())
    }

    /// Flips the current value of the Flipper's boolean.
    #[ink(message)]
    pub fn flip(&mut self) {
        self.value = !self.value;
    }

    /// Returns the current value of the Flipper's boolean.
    #[ink(message)]
    pub fn get(&self) -> bool {
        self.value
    }
}
```

Flipper Implementation

Contains functions that are specific to Flipper contract.

new and new_default are constructors that initialize the contract marked with #[ink(constructor)]

#[ink(message)]: defines the external interface for the smart contract which represents an action that external entities can invoke.

- flip(): Flips the boolean value
- get(): Retrieves the current boolean value

Test Module

```rust
#[cfg(test)]
mod tests {
  use super::*;

  #[ink::test]
  fn default_works() {
    let flipper = Flipper::default();
    assert_eq!(flipper.get(), false);
  }

  #[ink::test]
  fn it_works() {
    let mut flipper = Flipper::new(false);
    assert_eq!(flipper.get(), false);
    flipper.flip();
    assert_eq!(flipper.get(), true);
  }
}
```

#[ink(test)]: Macro indicates that tests are only compiled when running tests.

Verifies the functionality of Flipper contract.

E2E(End to End) Test

```rust
mod e2e_tests {
  use super::*;
  use ink_e2e::build_message;

  type E2EResult<T> = std::result::Result<T, Box<dyn std::error::Error>>;

  #[ink_e2e::test]
  async fn it_works(mut client: ink_e2e::Client<C, E>) -> E2EResult<()> {
    // Test logic goes here...
    Ok(())
  }

  #[ink_e2e::test]
  async fn default_works(mut client: ink_e2e::Client<C, E>) -> E2EResult<()> {
      Ok(())
  }
}
```

#[ink_e2e::test]: Indicates that the test are only compiled when both standard test and specific E2E test features are enabled

Uses Ink! E2e crate to conduct end-to-end testing simulating interactions with the Flipper contract in a controlled environment

CESS

Demo

Deploying Ink! Smart Contract on CESS

Deploying Ink! Smart Contract on CESS

CESS

Step 1: Clone and Build CESS source code

git clone https://github.com/CESSProject/cess.git --branch v0.7.4

cd cess

cargo build

Step 2: Start CESS node in Dev Mode

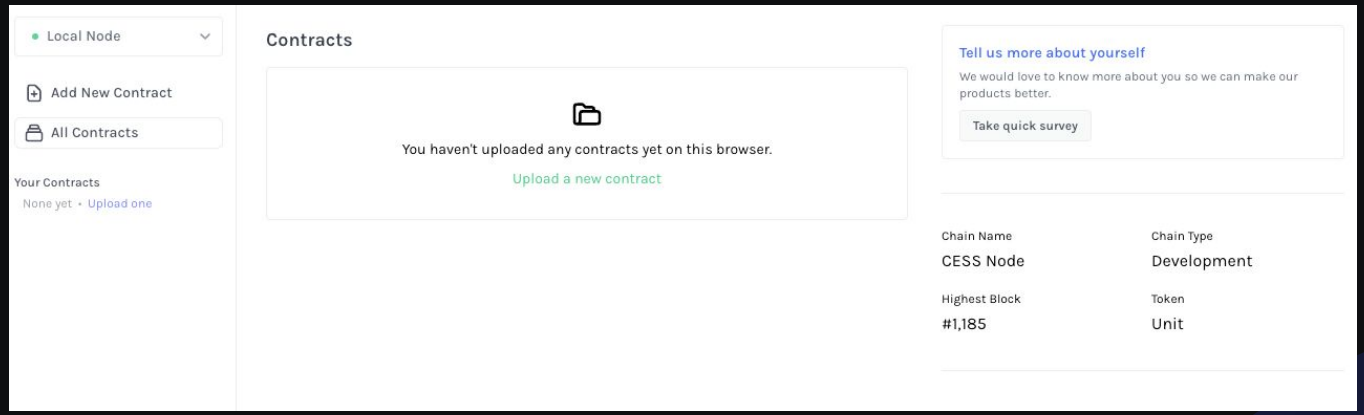
target/debug/cess-node --dev

```
2023-08-23 18:01:10 CESS Node
2023-08-23 18:01:10 ✌️ version 0.6.0-unknown
2023-08-23 18:01:10 ❤️ by CESS LAB, 2017-2023
2023-08-23 18:01:10 📋 Chain specification: Development
2023-08-23 18:01:10 🏷 Node name: alike-stretch-9767
2023-08-23 18:01:10 👤 Role: AUTHORITY
2023-08-23 18:01:10 💾 Database: RocksDb at /tmp/substrateZdfnHg/chains/dev/db/full
2023-08-23 18:01:10 ⛓ Native runtime: cess-node-109 (cess-node-1.tx1.au1)
channel 17: open failed: connect failed: Connection refused
2023-08-23 18:01:13 [0] 💸 generated 1 npos voters, 1 from validators and 0 nominators
2023-08-23 18:01:13 [0] 💸 generated 1 npos targets
2023-08-23 18:01:15 🔨 Initializing Genesis block/state (state: 0xd45c…c2b1, header-hash: 0x126c…4ad9)
2023-08-23 18:01:15 👴 Loading GRANDPA authority set from genesis on what appears to be first startup.
channel 17: open failed: connect failed: Connection refused
2023-08-23 18:01:17 👶 Creating empty RRSC epoch changes on what appears to be first startup.
2023-08-23 18:01:17 Using default protocol ID "sup" because none is configured in the chain specs
2023-08-23 18:01:17 🏷 Local node identity is: 12D3KooWPaspHpeUHLxY4iRhBL3cbMEscfhAFTuSEgPLWaJFpdgt
```

Deploying Ink! Smart Contract on CESS

Step 3: Open Substrate Contracts UI

https://contracts-ui.substrate.io/?rpc=ws://127.0.0.1:9944



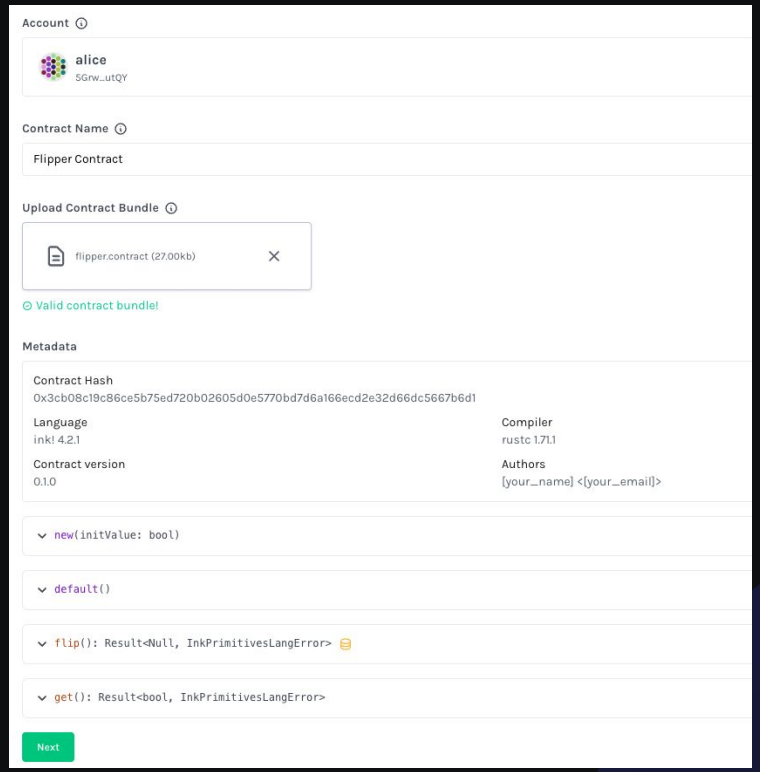
Step 4: Click on Upload a new Contract

Upload *target/ink/flipper.contract file*

Deploying Ink! Smart Contract on CESS

Step 5:

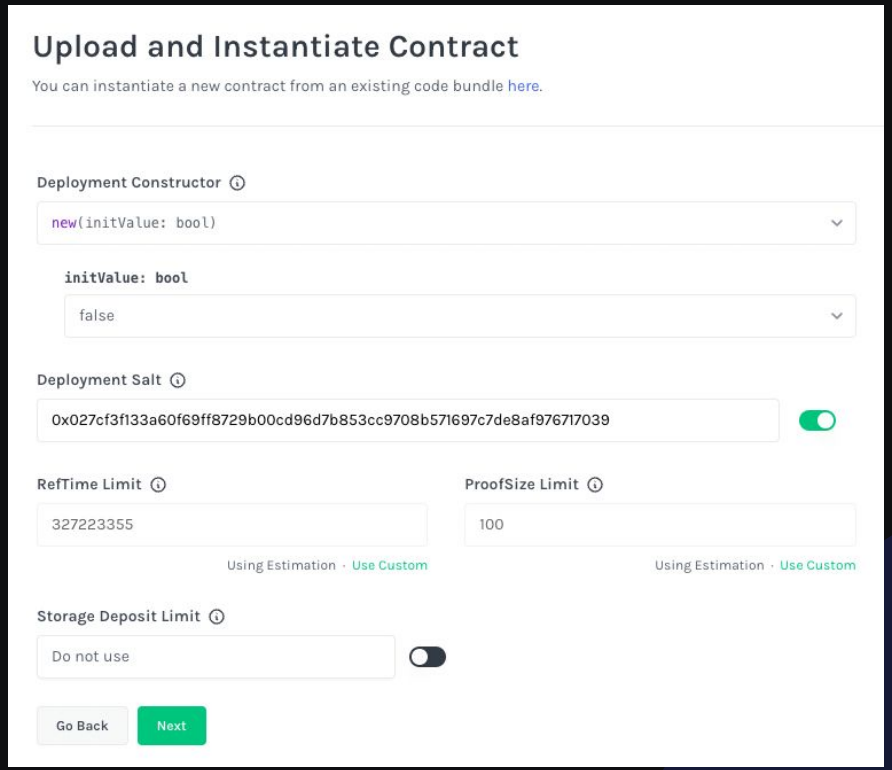
Give a name to your Contract in “Contract Name” field and Click “Next”



Deploying Ink! Smart Contract on CESS

Step 6:

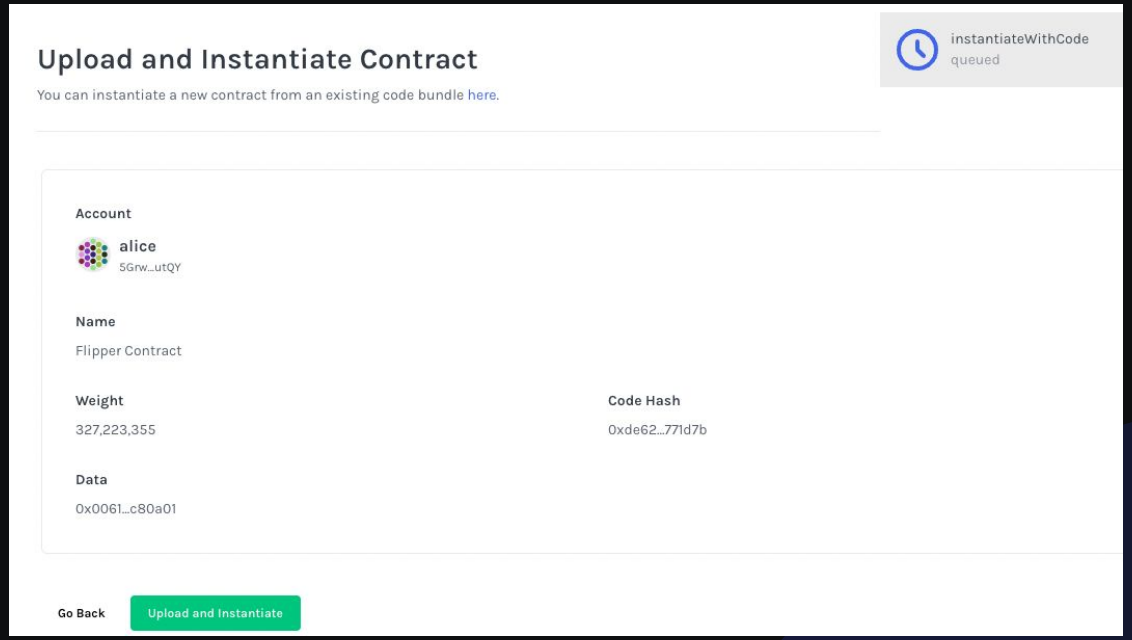
Instantiate Contract with Default Values



Deploying Ink! Smart Contract on CESS

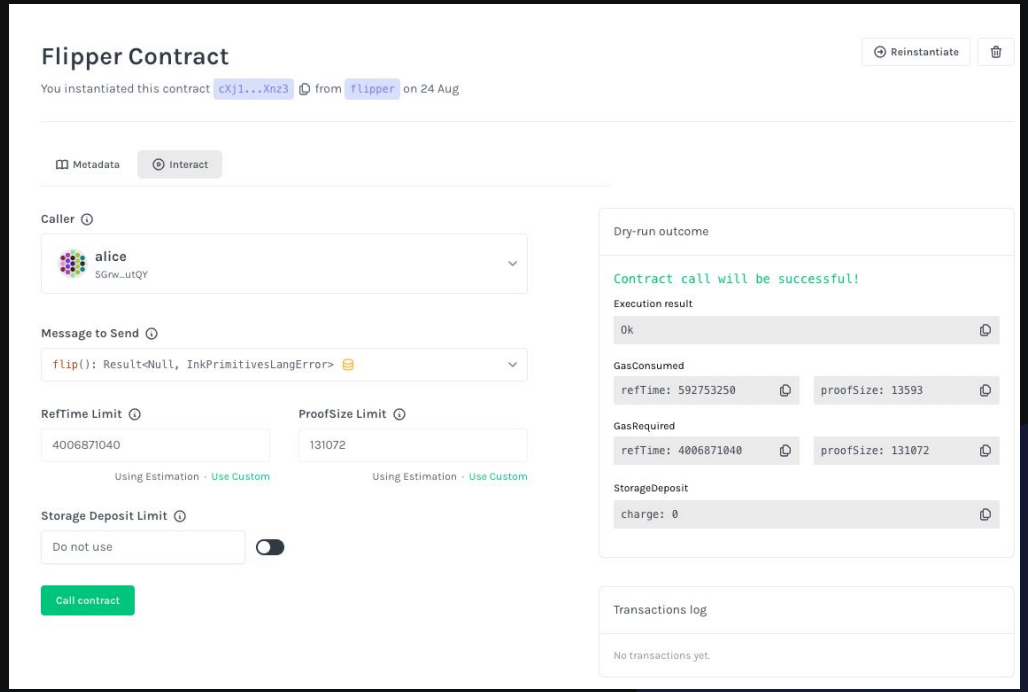
Congratulations!

You have Successfully Instantiated Smart Contract



Interacting With Ink! Smart Contract

- Select caller account
- Select function to execute
- Enter the desired values
- Call Contract



CESS

Demo

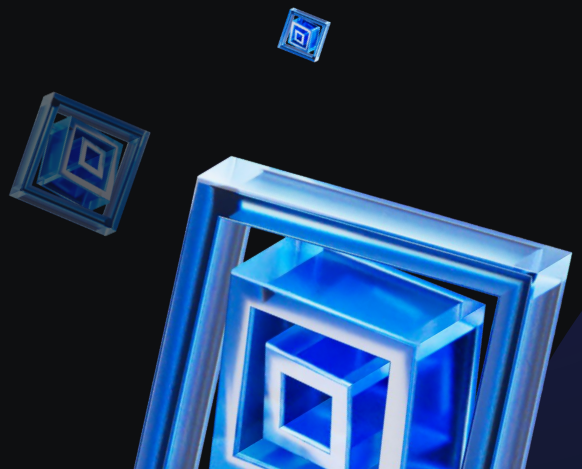
Interacting with Ink! Smart Contract Using

The useink Library

CESS

Demo

NFT Marketplace Example



CESS

Thank you for watching

Please Join Our Community

CESS

CESS Network - Episode 8

DApp Development using Ink!
Smart Contract

https://www.cess.network