

LAPORAN PRAKTIKUM
BLOCKCHAIN
“Smart Contract”



Oleh:
Mauliza Aprilia
NIM 22346014

Dosen Pengampu:
Ade Kurniawan, S.Pd., M.Pd.T

PROGRAM STUDI INFORMATIKA
DEPARTEMEN TEKNIK ELEKTRONIKA
FAKULTAS TEKNIK
UNIVERSITAS NEGERI PADANG
2025

1. Solidity

Solidity adalah bahasa pemrograman utama untuk menulis *smart contract* di Ethereum. Bahasa ini memiliki sintaks yang mirip dengan JavaScript dan mendukung tipe data statis, pustaka, serta tipe data yang dapat didefinisikan pengguna. Solidity dirancang untuk dikompilasi ke dalam *bytecode* yang dapat dijalankan pada Ethereum Virtual Machine (EVM).

Karena Solidity digunakan di lingkungan terisolasi (*sandboxed*), kontrak pintar yang berjalan di EVM tidak dapat mengakses jaringan, sistem file, atau proses lain secara langsung. Ini berarti interaksi dengan dunia luar memerlukan bantuan layanan seperti *Oracle*, yang berfungsi sebagai perantara untuk memasukkan data eksternal ke dalam blockchain.

Sebagai bahasa yang *statically typed*, Solidity memiliki aturan ketat dalam penulisan kode, seperti:

- Semua fungsi harus dideklarasikan dengan jelas.
- Tidak boleh ada objek bernilai null.
- Operator dan ekspresi harus sesuai dengan tipe data yang digunakan.

Selain Solidity, ada bahasa lain untuk menulis smart contract, seperti Serpent, Vyper, dan LLL, tetapi Solidity tetap menjadi yang paling populer dan banyak digunakan di komunitas Ethereum.

2. Smart Contract

Smart contract adalah program yang berjalan secara otomatis saat kondisi tertentu terpenuhi. Konsep ini memungkinkan transaksi yang transparan dan dapat diverifikasi tanpa perlu pihak ketiga sebagai perantara.

Smart contract di Ethereum umumnya ditulis menggunakan Solidity dan bekerja berdasarkan logika *IF-THIS-THEN-THAT* (IFTTT), yang berarti kode hanya akan dieksekusi ketika kondisi tertentu terpenuhi.

Karena kontrak pintar berjalan di dalam EVM, mereka tidak dapat langsung mengakses jaringan eksternal, sistem file, atau sumber daya lain di luar blockchain. Namun, jika diperlukan, kontrak pintar dapat memperoleh data eksternal melalui layanan *Oracle*, yang mengambil informasi dari luar blockchain dan mengirimkannya ke kontrak pintar.

Dalam implementasinya, smart contract dapat dijalankan pada dua jenis sistem:

1. **Ethereum Virtual Machine (EVM)** – lingkungan eksekusi untuk kontrak pintar Ethereum.
2. **Hyperledger Fabric** – platform *blockchain* yang lebih fleksibel untuk bisnis dan perusahaan.

Sebagian besar pembahasan dan contoh dalam dokumen ini menggunakan Ethereum sebagai platform utama.

3. GAS

GAS adalah unit yang digunakan untuk mengukur biaya eksekusi transaksi di Ethereum. Setiap operasi yang dilakukan di EVM memiliki harga GAS tertentu, yang mencerminkan jumlah sumber daya komputasi yang diperlukan untuk menyelesaikan transaksi.

Biaya GAS bergantung pada kompleksitas transaksi: semakin kompleks operasinya, semakin banyak GAS yang dibutuhkan. GAS dihitung dalam satuan terkecil yang disebut wei, dengan konversi sebagai berikut:

- $1 \text{ ETH} = 10^{18} \text{ wei} = 10^9 \text{ gwei}$

Komponen GAS:

1. **Harga GAS (Gas Price):** biaya per unit GAS yang ditentukan oleh pengguna. Semakin tinggi harga GAS yang ditetapkan, semakin cepat transaksi diproses oleh penambang.
2. **Batas GAS (Gas Limit):** batas maksimum GAS yang pengguna bersedia bayarkan untuk suatu transaksi. Jika transaksi menggunakan lebih sedikit GAS dari batas yang ditentukan, sisa dana akan dikembalikan ke pengguna.
3. **Biaya GAS Aktual:** jumlah total GAS yang digunakan oleh transaksi, dikalikan dengan harga GAS.

Rumus biaya transaksi:

$$\text{Biaya Transaksi} = \text{Gas yang Digunakan} \times \text{Harga GAS}$$

Jika transaksi melebihi batas GAS yang ditetapkan, transaksi akan gagal, tetapi pengguna tetap harus membayar biaya untuk sumber daya yang telah digunakan oleh jaringan.

3.1 Mengapa GAS Diperlukan?

Ada tiga alasan utama mengapa Ethereum menggunakan sistem GAS:

1. **Alasan Finansial:** GAS memberikan insentif bagi penambang untuk menjalankan transaksi dan smart contract dengan menggunakan sumber daya mereka. Semakin kompleks transaksi, semakin tinggi biaya GAS yang harus dibayar pengguna.
2. **Alasan Teoretis:** GAS membantu mengurangi potensi tindakan jahat di dalam jaringan. Setiap transaksi membutuhkan biaya GAS, sehingga aktor jahat tidak akan bisa melakukan serangan seperti spam transaksi tanpa biaya.
3. **Alasan Komputasi:** GAS membantu menyelesaikan *Halting Problem* yang dijelaskan oleh Alan Turing. Dalam komputasi, *Halting Problem* menyatakan bahwa tidak mungkin untuk mengetahui apakah suatu program akan berhenti atau berjalan selamanya. Dengan batas GAS, setiap transaksi memiliki jumlah eksekusi yang terbatas, sehingga tidak mungkin ada program yang berjalan tanpa batas.

4. Ether (ETH)

Ether (ETH) adalah mata uang kripto utama dalam ekosistem Ethereum. ETH digunakan untuk berbagai tujuan, termasuk:

- Membayar biaya transaksi dan eksekusi smart contract.
- Berfungsi sebagai unit nilai dalam aplikasi *decentralized finance (DeFi)*.
- Digunakan dalam mekanisme *staking* dalam Ethereum 2.0 (Proof of Stake).

Di Ethereum, biaya transaksi dihitung dalam ETH dengan rumus:

$$\text{Biaya Transaksi} = \text{GAS yang Digunakan} \times \text{Harga GAS}$$

Jika pengguna menetapkan harga GAS terlalu rendah, transaksi mereka bisa tertunda atau bahkan tidak diproses oleh penambang.

5. Account di Ethereum

Setiap akun di Ethereum memiliki alamat unik dan dapat menyimpan saldo ETH. Ada dua jenis akun dalam Ethereum:

1. Externally Owned Account (EOA)

- Dikendalikan oleh kunci privat.
- Digunakan oleh pengguna untuk menyimpan dan mentransfer ETH.
- Hanya akun jenis ini yang dapat memulai transaksi di Ethereum.

2. Contract Account (Akun Kontrak)

- Dikendalikan oleh kode dalam smart contract.
- Dapat menerima dan mengirim ETH, tetapi tidak dapat memulai transaksi sendiri.
- Memiliki penyimpanan sendiri untuk menyimpan data.

Struktur dasar akun Ethereum:

- **EOA** = Saldo ETH + Tidak memiliki kode + Tidak memiliki penyimpanan.
- **Akun Kontrak** = Saldo ETH + Kode smart contract + Penyimpanan data.

Hanya akun EOA yang dapat memulai transaksi, tetapi akun kontrak dapat merespons transaksi dan menjalankan kode yang telah ditentukan sebelumnya.

6. Transaction (Transaksi) di Ethereum

Transaksi adalah mekanisme utama untuk mengubah status dalam blockchain Ethereum. Transaksi bisa berupa:

1. **Transfer ETH antar akun.**
2. **Pemanggilan fungsi dalam smart contract.**

Setiap transaksi berisi informasi berikut:

- **Alamat Pengirim:** Akun yang mengirim transaksi.
- **Alamat Tujuan:** Akun penerima atau alamat kontrak pintar yang akan dipanggil.
- **Data Transaksi:** Informasi tambahan yang diperlukan untuk eksekusi smart contract.
- **Batas GAS:** Jumlah maksimum GAS yang disediakan oleh pengguna.
- **Harga GAS:** Biaya per unit GAS yang bersedia dibayar pengguna.
- **Nonce:** Nomor unik untuk mencegah transaksi ganda.

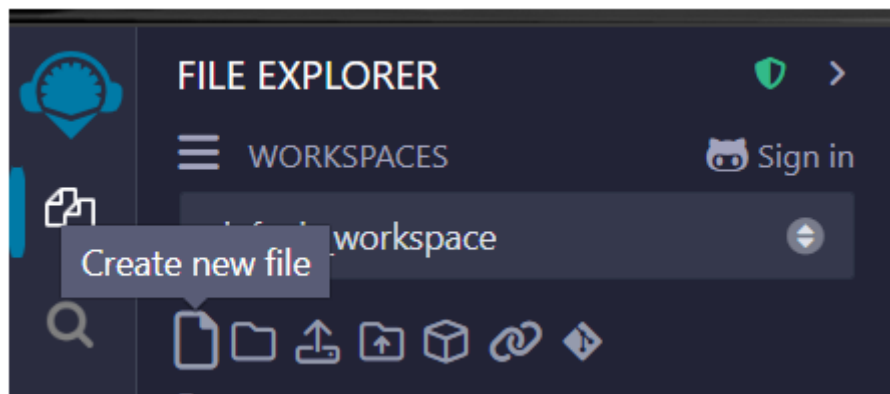
Jika transaksi dikirim ke akun kontrak, maka kode dalam kontrak tersebut akan dieksekusi, dan hasilnya akan disimpan dalam blockchain Ethereum. Karena semua transaksi terekam secara permanen di blockchain, tidak mungkin untuk membatalkan atau mengubah transaksi setelah dikonfirmasi oleh jaringan.

Kesimpulan

Ethereum adalah platform yang kuat untuk menjalankan smart contract, dengan Solidity sebagai bahasa pemrograman utamanya. Konsep GAS memastikan bahwa sumber daya komputasi digunakan secara efisien dan aman, serta memberi insentif bagi penambang untuk memproses transaksi. ETH berfungsi sebagai mata uang utama dalam jaringan Ethereum, sementara sistem akun dan transaksi memastikan keamanan dan transparansi dalam eksekusi kontrak pintar.

LANGKAH LANGKAH PRAKTIKUM

1. Buka Remix IDE.
2. Buat file SimpleContract.sol



3. Salin code berikut:

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.18;

contract MergedContract {
    string public message;
    uint256 public value;

    constructor(string memory _message, uint256 _value) {
        message = _message;
        value = _value;
    }
}
```

```

    }

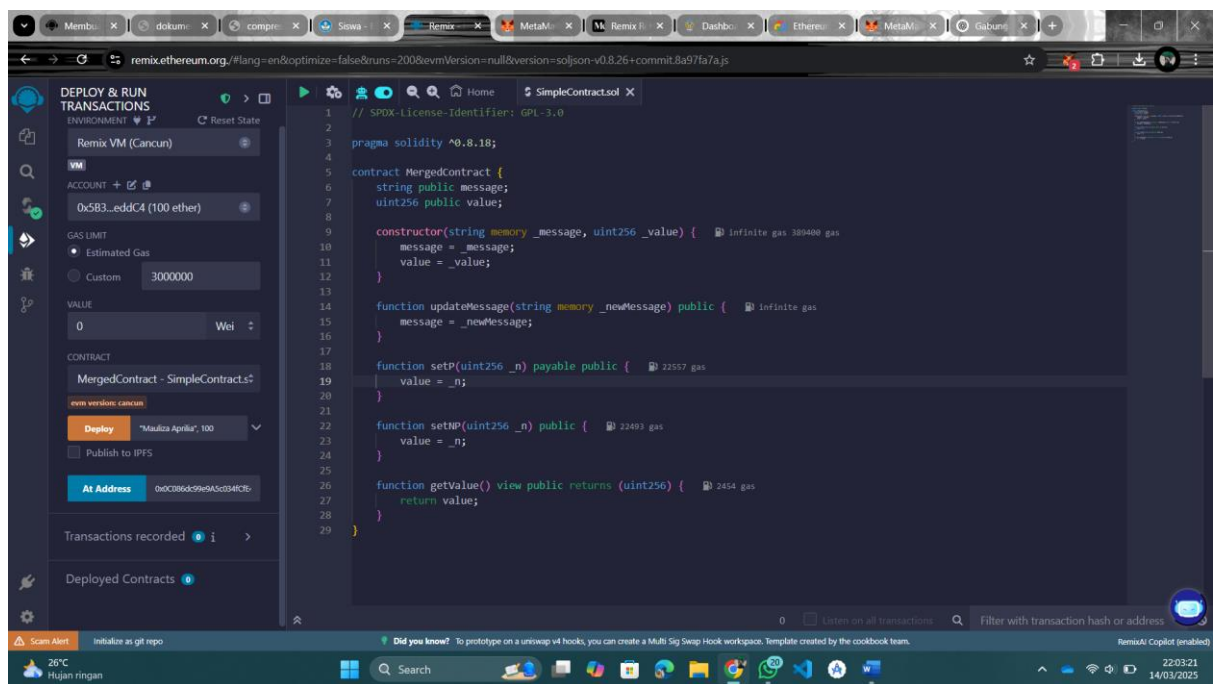
    function updateMessage(string memory _newMessage) public {
        message = _newMessage;
    }

    function setP(uint256 _n) payable public {
        value = _n;
    }

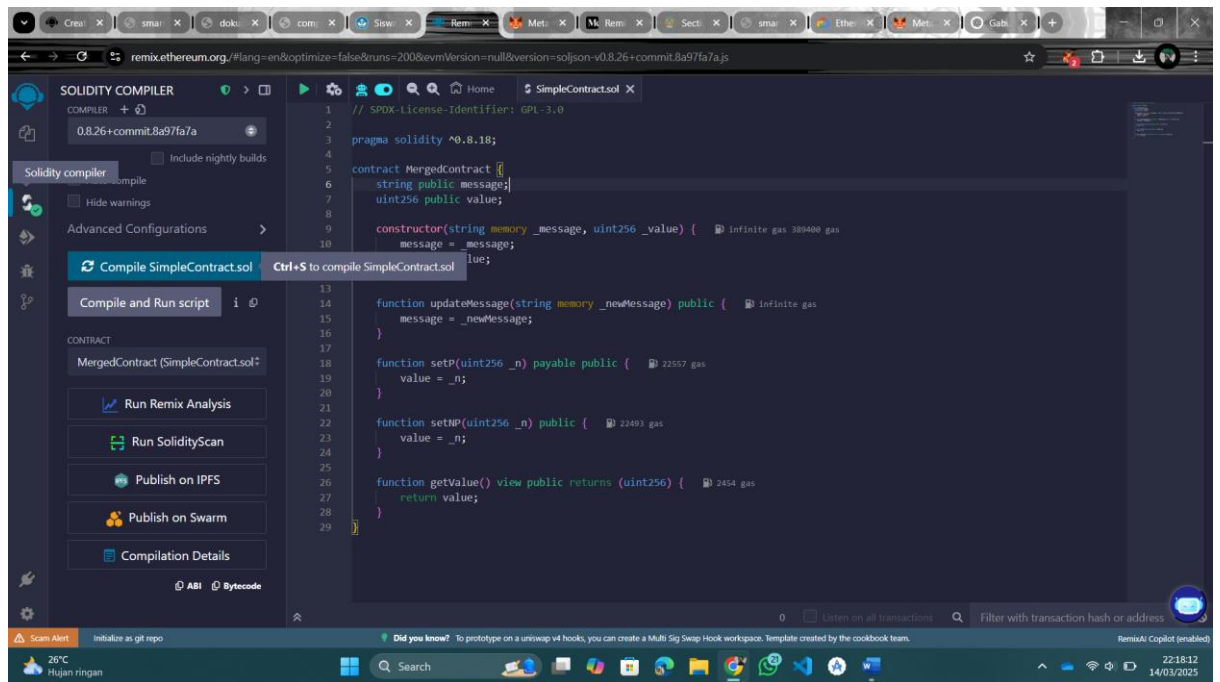
    function setNP(uint256 _n) public {
        value = _n;
    }

    function getValue() view public returns (uint256) {
        return value;
    }
}

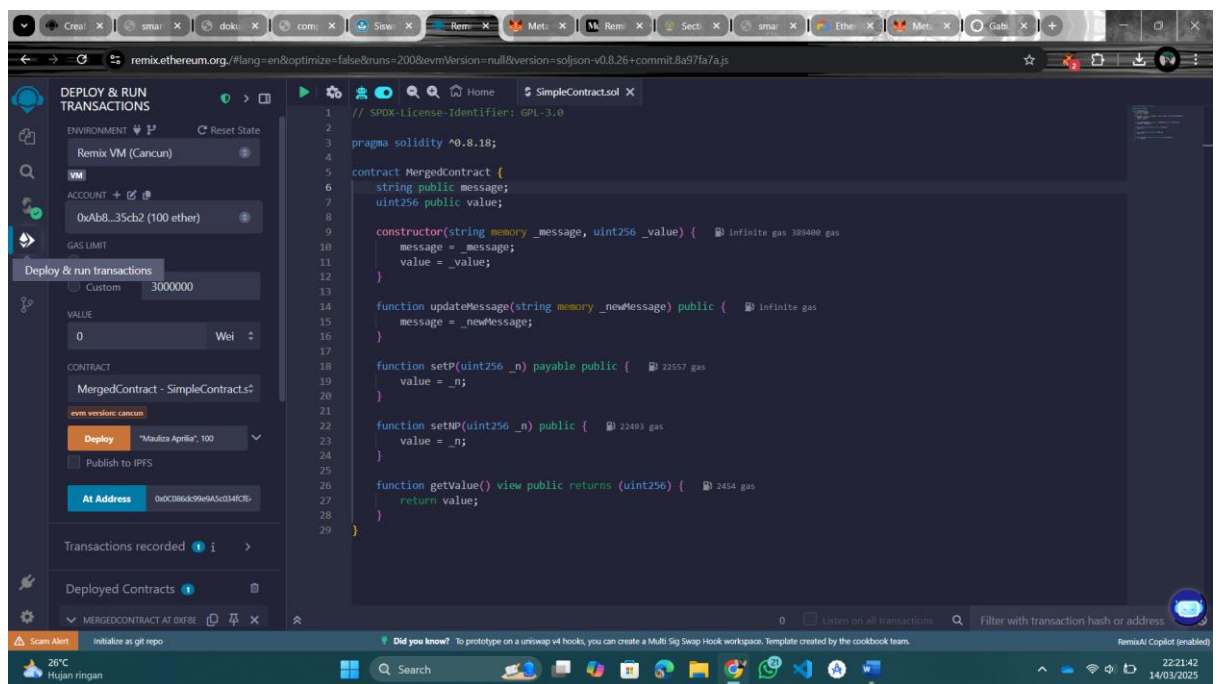
```



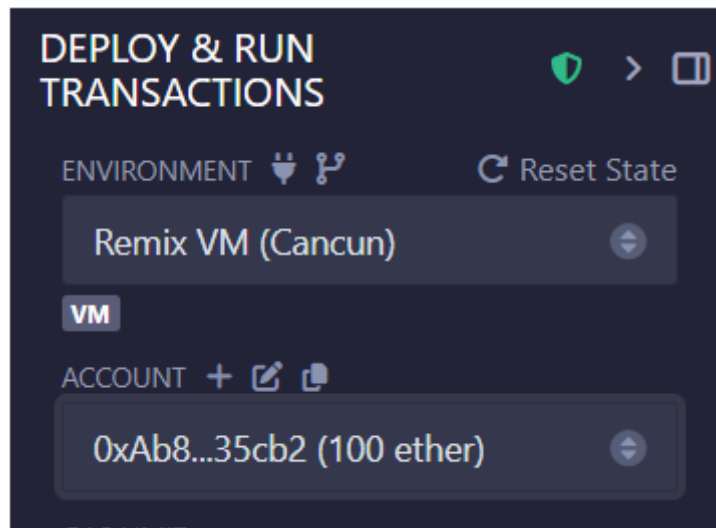
4. Compile file contract



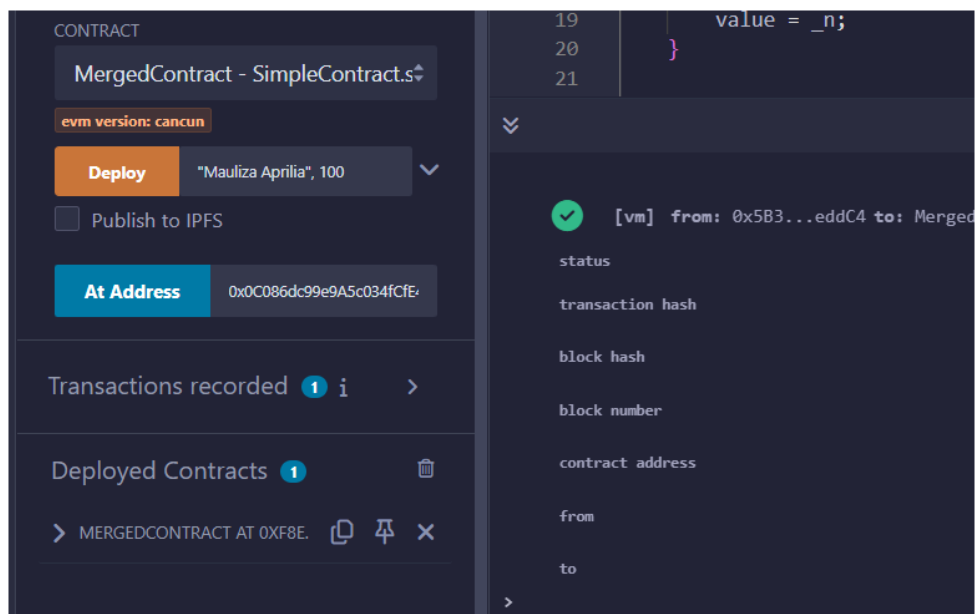
5. Deploy contract



- a. Pilih Environment Remix VM yang paling atas






- b. Deploy sebuah contract






- c. Berinteraksi dengan instance yang diterapkan


DEPLOY & RUN TRANSACTIONS

ENVIRONMENT   Reset State

Remix VM (Cancun) 

VM

ACCOUNT   


0x5B3...eddC4 (99.999999999...) 


GAS LIMIT


☒ Estimated Gas




☐ Custom

VALUE

Wei 

Transactions recorded 1  >

Deployed Contracts 1 

▼ MERGEDCONTRACT AT 0XF8E   

Balance: 0 ETH

setNP uint256_n ▼

setP uint256_n ▼

updateMessage string_newMessage ▼

getValue


0: uint256: 100

message

0: string: Mauliza Aprilia

value

0: uint256: 100

Low level interactions 

remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.26+commit.8a97fa7a.js

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: **cancon**
Deploy "Maulia Aprilla", 100
Publish to IPFS
At Address: 0x0C086d59a9A5c0348C36

Transactions recorded: 1
Deployed Contracts: 1
MERGEDCONTRACT AT 0xF81

Balance: 0 ETH

Contract: **SimpleContract.sol**

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity ^0.8.18;
4
5 contract MergedContract {
6     string public message;
7     uint256 public value;
8
9     constructor(string memory _message, uint256 _value) {
10         message = _message;
11         value = _value;
12     }
13
14     function updateMessage(string memory _newMessage) public {
15         message = _newMessage;
16     }
17 }
```

Decoded input: { "string_message": "Maulia Aprilla", "uint256_value": "100" }

Decoded output: -

Logs: []

Raw logs: []

Low level interactions: CALLDATA

Scam Alert: Initialize as git repo

Did you know? To prototype on a uniswap v4 hook, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team.

RemixAI Copilot (enabled)

22:04:51 14/03/2025

remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.26+commit.8a97fa7a.js

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: **Remix VM (Cancon)**
Reset State
ACCOUNT: 0x5B3...eddC4 (99.99999999...)
GAS LIMIT: Estimated Gas
Custom: 3000000
VALUE: 0 Wei

CONTRACT: **MergedContract - SimpleContract**
evm version: cancon
Deploy "Maulia Aprilla", 100
Publish to IPFS
At Address: 0x0C086d59a9A5c0348C36

Transactions recorded: 1
Deployed Contracts: 1
MERGEDCONTRACT AT 0xF81

Contract: **SimpleContract.sol**

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity ^0.8.18;
4
5 contract MergedContract {
6     string public message;
7     uint256 public value;
8
9     constructor(string memory _message, uint256 _value) {
10         message = _message;
11         value = _value;
12     }
13
14     function updateMessage(string memory _newMessage) public {
15         message = _newMessage;
16     }
17 }
```

Decoded input: { "string_message": "Maulia Aprilla", "uint256_value": "100" }

Decoded output: -

Logs: []

Raw logs: []

Scam Alert: Initialize as git repo

Did you know? To prototype on a uniswap v4 hook, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team.

RemixAI Copilot (enabled)

22:05:03 14/03/2025

remix.ethereum.org/#lang=en&optimize=false&runs=2008&evmVersion=null&version=soljson-v0.8.26+commit.8a97fa7a.js

DEPLOY & RUN TRANSACTIONS

At Address 0x00

Transactions recorded 1

Deployed Contracts 1

MERGEDCONTRACT AT 0x3E

Balance: 0 ETH

setP uint256_n

setP uint256_n

updateMessage string_newMessage

getValue

message

value

Low level interactions

CALLDATA

Transact

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity ^0.8.18;
4
5 contract MergedContract {
6     string public message;
7     uint256 public value;
8
9     constructor(string memory _message, uint256 _value) {
10         message = _message;
11         value = _value;
12     }
13
14     function updateMessage(string memory _newMessage) public {
15         message = _newMessage;
16     }
17 }
```

logs 0 Listen on all transactions Filter with transaction hash or address

raw logs 1

call to MergedContract.getValue

ms: [call] from: 0x58380a6a701c568545dcfc803fc8875f56beddC4 to: MergedContract.getValue() data: 0x209...65255 Debug

call to MergedContract.message

ms: [call] from: 0x58380a6a701c568545dcfc803fc8875f56beddC4 to: MergedContract.message() data: 0xe21...f37ce Debug

call to MergedContract.value

ms: [call] from: 0x58380a6a701c568545dcfc803fc8875f56beddC4 to: MergedContract.value() data: 0x3fa...4f245 Debug

Scan Alert Initialize as git repo Did you know? To prototype on a uniswap v4 hooks, you can create a Multi Sig Swap Hook workspace. Template created by the cookbook team. RemixAI Copilot (enabled)

26°C Hujan ringan Search 22:06:54 14/03/2025