CHENNAI
INSTITUTE OF TECHNOLOGY
(Autonomous)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E – CSE / IT

Regulation: 2022

III Year / VI Semester

CS2V27/ IT2V27 / AD2V27 / CY2V27 / CB2V27/ AM2V27 / - BLOCKCHAIN TECHNOLOGY AND CLOUD COMPUTING LAB

- LAB MANUAL

BLOCKCHAIN TECHNOLOGY AND CLOUD COMPUTING LAB

Ex. No	DATE	TITLE	CO	PO	Page No	MARKS	SIGN
1.A)		Simulate a basic blockchain network to understand transactions, blocks, and the structure of peer-to-peer systems	C2	PO1, PO2, PO3, PO4, PO5, PO8, PO9, PO10			
1.B)		Implementing on-demand resource provisioning in cloud computing.	C2	PO1, PO2, PO3, PO4, PO5, PO8, PO9, PO10			
2.A)		Analyze the Bitcoin mining process, types of wallets, and Ethereum Virtual Machine (EVM) while exploring consensus mechanisms and smart contracts' impact on cryptocurrency.	C1, C2	PO1, PO2, PO3, PO4, PO5, PO8, PO9, PO10			
2.B)		Understanding cloud basics such as service-oriented architecture (SOA) and virtualization.	C4	PO1, PO2, PO3, PO4, PO5, PO8, PO9, PO10			
3.A)		Develop smart contracts in Solidity programming language for Ethereum, comprehend Hyperledger Fabric and Hyperledger Composer	C2	PO1, PO2, PO3, PO4, PO5, PO8, PO9, PO10			
3.B)		Design layered cloud architecture considering cloud services and service models.	C1, C3	PO1, PO2, PO3, PO4, PO5, PO8, PO9, PO10			
4.A)		Implement Solidity programming language features like variables, functions, and data structures, alongside understanding Ethereum wallet and smart contracts structure	C1	PO1, PO2, PO3, PO4, PO5, PO8, PO9, PO10			
4.B)		Ensure cloud security through identity & access management (IAM) and compliance with security standards.	C4, C5	PO1, PO2, PO3, PO4, PO5, PO8, PO9, PO10, PO 11, PO12			
5.		Explore blockchain applications like IoT integration and medical record management,analyze alternative cryptocurrencies (Alt Coins) and their significance, and investigate	C3, C4 , C5	PO1, PO2, PO3, PO4, PO5, PO8, PO9, PO10			

1.A) Simulate a basic blockchain network to understand transactions, blocks, and the structure of peer-to-peer systems

Aim:

To simulate a basic blockchain network to understand transactions, blocks, and the structure of peer-to-peer systems.

Procedure:

1. Install Python and Required Libraries
   1. Ensure Python is installed on your system (Python 3.x recommended).
   2. Install Flask for the blockchain API:

      pip install Flask
2. Create a Blockchain Class
   1. Define a Block class with attributes like index, timestamp, transactions, previous hash, and hash.
   2. Implement a method to compute the hash of each block.
3. Implement a Transaction System
   1. Create a function to add transactions.
   2. Store transactions in a list until they are added to a new block.
4. Create a Proof-of-Work Mechanism
   1. Implement a simple Proof-of-Work (PoW) algorithm.
   2. The algorithm should involve solving a mathematical problem to add a new block.
5. Build a Flask API to Simulate Peer-to-Peer Network
   1. Implement API endpoints to:
      - Add transactions
      - View the blockchain
6. Test the Blockchain System
   1. Run the Flask server and test the endpoints using curl or Postman.
   2. Verify transaction propagation and block creation.
7. Add a Transaction (POST request)

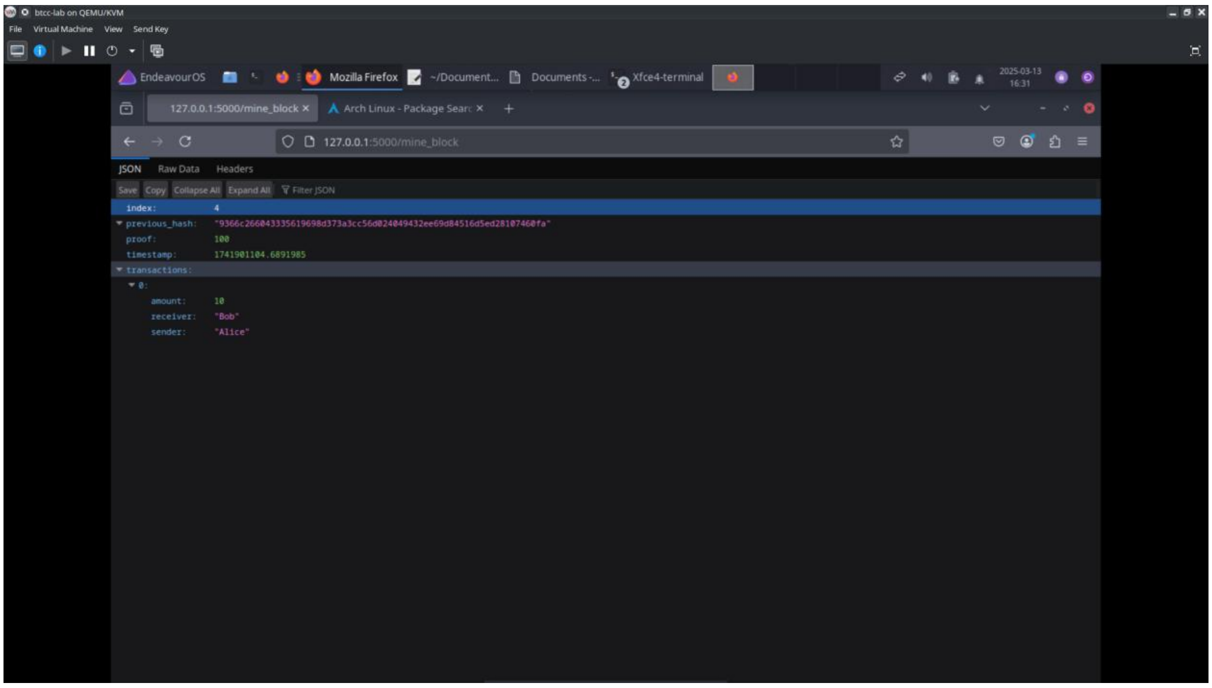
   Use Postman or cURL to send a transaction.

```
curl -X POST http://127.0.0.1:5000/add_transaction \
-H "Content-Type: application/json" \
-d '{"sender": "Alice", "receiver": "Bob", "amount": 10}'
```

Program:

```
import hashlib
import json
from time import time
from flask import Flask, jsonify, request
class Blockchain:
  def __init__(self):
    self.chain = []
    self.transactions = []
    self.create_block(proof=1, previous_hash='0')
  def create_block(self, proof, previous_hash):
    block = {
      'index': len(self.chain) + 1,
      'timestamp': time(),
      'transactions': self.transactions,
      'proof': proof,
      'previous_hash': previous_hash
    }
    self.transactions = []
    self.chain.append(block)
    return block
  def add_transaction(self, sender, receiver, amount):
    self.transactions.append({'sender': sender, 'receiver': receiver, 'amount': amount})
    return self.last_block['index'] + 1
  @property
  def last_block(self):
    return self.chain[-1]

app = Flask(__name__)
bc = Blockchain()
@app.route('/mine_block', methods=['GET'])
def mine_block():
  previous_block = bc.last_block
  previous_hash = hashlib.sha256(json.dumps(previous_block, sort_keys=True).encode()).hexdigest()
  block = bc.create_block(proof=100, previous_hash=previous_hash)
  return jsonify(block), 200
@app.route('/add_transaction', methods=['POST'])
def add_transaction():
  data = request.get_json()
  index = bc.add_transaction(data['sender'], data['receiver'], data['amount'])
  return jsonify({'message': f'Transaction added to Block {index}'}), 201
if __name__ == '__main__':
  app.run(debug=True)
```

Output:


btcc-lab on QEMU/KVM
File Virtual Machine View Send Key
EndeavourOS
Mozilla Firefox
~/Document...
Documents -...
Xfce4-terminal
127.0.0.1:5000/mine_block
Arch Linux - Package Searc
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON

```
127.0.0.1:5000/mine_block

index:          4
previous_hash:  "9366c266043335619698d373a3cc56d024049432ee69d84516d5ed28107460fa"
proof:          100
timestamp:      1741901104.6891985
transactions:
  0:
    amount:     10
    receiver:   "Bob"
    sender:     "Alice"
```

Result:

Simulatation of a basic blockchain network to understand transactions, blocks, and the structure of peer-to-peer systems is completed successfully.

1.B) Implementing on-demand resource provisioning in cloud computing.

Aim:

To implement on-demand resource provisioning in cloud computing

Procedure:

1. Sign in to AWS Console
   - Navigate to AWS Console.
   - Sign in with your AWS credentials.
2. Launch an EC2 Instance
   - Go to the EC2 Dashboard.
   - Click on "Launch Instance."
   - Choose an Amazon Machine Image (AMI) (e.g., Ubuntu 20.04 LTS).
   - Select an instance type (e.g., t2.micro for free-tier users).
   - Configure instance settings:
     - Number of instances: 1
     - Auto-assign Public IP: Enable
   - Add storage (default 8GB is sufficient).
   - Configure security groups:
     - Allow SSH (port 22)
     - Allow HTTP (port 80)
   - Assign a key pair for secure access.
3. Access the EC2 Instance
   - Download the private key (.pem file).
   - Use SSH to connect:

```
chmod 400 your-key.pem
ssh -i your-key.pem ubuntu@your-ec2-public-ip
```

4. Deploy a Sample Web Application
   - Install Apache Web Server:

```
sudo apt update
sudo apt install apache2 -y
```

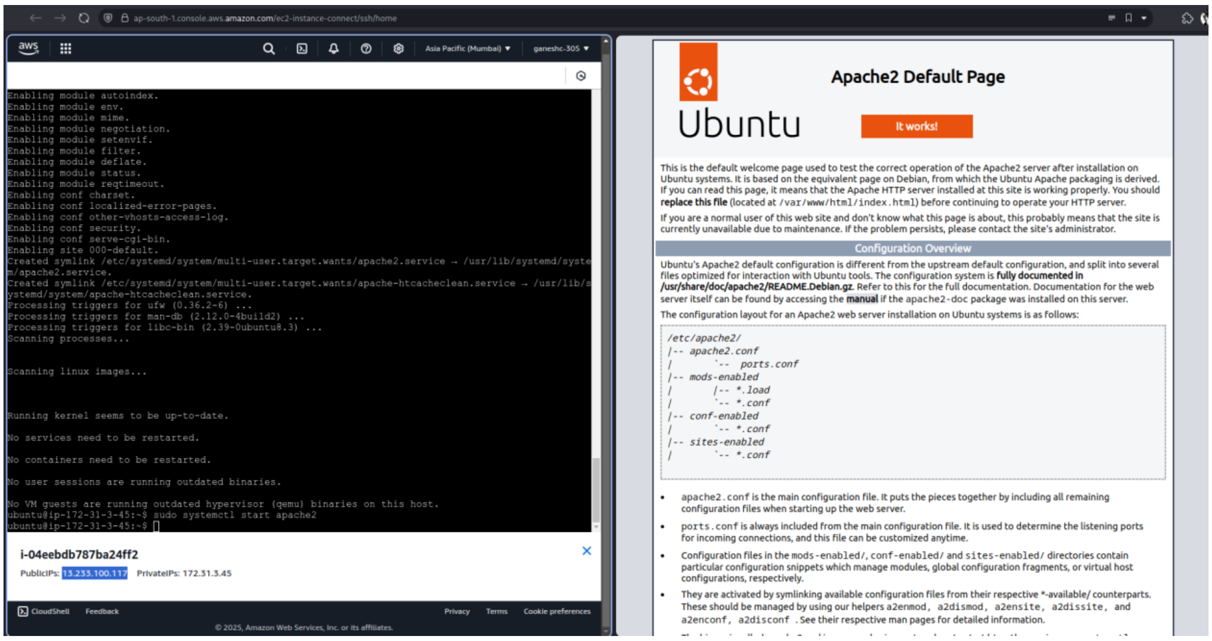
   - Start the server:

```
sudo systemctl start apache2
```

   - Verify the web server by accessing http://your-ec2-public-ip/ in a browser.

5. Auto Scaling Configuration
   - ➢ Set up an Auto Scaling Group in AWS to scale resources dynamically.
6. Terminate Instance
   - ➢ Navigate to EC2 Dashboard and terminate the instance to avoid charges.

Output:


ap-south-1.console.aws.amazon.com/ec2-instance-connect/ssh/home
aws
Asia Pacific (Mumbai)
ganeshc-305

```
Enabling module autoindex.
Enabling module env.
Enabling module mime.
Enabling module negotiation.
Enabling module setenvif.
Enabling module filter.
Enabling module deflate.
Enabling module status.
Enabling module reqtimeout.
Enabling conf charset.
Enabling conf localized-error-pages.
Enabling conf other-vhosts-access-log.
Enabling conf security.
Enabling conf serve-cgi-bin.
Enabling site 000-default.
Created symlink /etc/systemd/system/multi-user.target.wants/apache2.service → /usr/lib/systemd/syste
m/apache2.service.
Created symlink /etc/systemd/system/multi-user.target.wants/apache-htcacheclean.service → /usr/lib/s
ystemd/system/apache-htcacheclean.service.
Processing triggers for ufw (0.36.2-6) ...
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.3) ...
Scanning processes...


Scanning linux images...



Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-3-45:~$ sudo systemctl start apache2
ubuntu@ip-172-31-3-45:~$ 
```

i-04eebdb787ba24ff2

PublicIPs: 13.233.100.117 PrivateIPs: 172.31.3.45

Apache2 Default Page

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at /var/www/html/index.html) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- apache2.conf is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- ports.conf is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the mods-enabled/, conf-enabled/ and sites-enabled/ directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective *-available/ counterparts. These should be managed by using our helpers a2enmod, a2dismod, a2ensite, a2dissite, and a2enconf, a2disconf . See their respective man pages for detailed information.

Result:

Implementation of on-demand resource provisioning in cloud computing is completed successfully.

2. A) Analyze the Bitcoin mining process, types of wallets, and Ethereum Virtual Machine (EVM) while exploring consensus mechanisms and smart contracts' impact on cryptocurrency.

Aim :

To analyze the Bitcoin mining process, types of wallets, and Ethereum Virtual Machine (EVM) while exploring consensus mechanisms and smart contracts' impact on cryptocurrency.

Theory:

Bitcoin Mining Process:

Bitcoin mining involves solving cryptographic puzzles to validate transactions and add blocks to the Bitcoin blockchain.

Proof-of-Work (PoW) is the consensus mechanism used, requiring miners to find a nonce that produces a hash below a set target.

Types of Wallets:

- Hot Wallets: Software wallets (e.g., mobile, desktop, web wallets).
- Cold Wallets: Hardware wallets and paper wallets.
- Multi-Signature Wallets: Require multiple signatures for transactions.

Ethereum Virtual Machine (EVM):

- Executes smart contracts and transactions on Ethereum.
- Gas is the measure of computational effort required for execution.
- Turing Completeness allows the EVM to handle any computation.

Consensus Mechanisms:

- Proof of Work (PoW): Bitcoin's method for validating transactions.
- Proof of Stake (PoS): Ethereum 2.0's energy-efficient mechanism.
- Delegated Proof of Stake (DPoS): Used by platforms like EOS.
- Proof of Authority (PoA): Used by networks like VeChain.

Smart Contracts:

- Self-executing contracts with terms written into code.
- Enable automation and decentralization in blockchain applications.

Procedure:

Bitcoin Mining Process (using regtest mode):

1. Start the Bitcoin daemon with the following command

```
bitcoind -regtest -fallbackfee=0.0001 -daemon
```

2. Create a new wallet:

```
bitcoin-cli -regtest createwallet "testwallet"
```

3. Load the wallet:

```
bitcoin-cli -regtest loadwallet "testwallet"
```

4. Generate 101 blocks (mining):

```
bitcoin-cli -regtest -generate 101
```

5. Check wallet balance:

```
bitcoin-cli -regtest getbalance
```

6. Get a new Bitcoin address:

```
bitcoin-cli -regtest getnewaddress
```

7. Send some Bitcoin to another address:

```
bitcoin-cli -regtest sendtoaddress "your_new_address" 10
```

8. Check the raw memory pool:

```
bitcoin-cli -regtest getrawmempool
```

9. Check balance again:

```
bitcoin-cli -regtest getbalance
```

10. Get blockchain info:

```
bitcoin-cli -regtest getblockchaininfo
```

11. List all transactions:

```
bitcoin-cli -regtest listtransactions
```

12. Retrieve block information using block hash:

```
bitcoin-cli -regtest getblock "blockhash"
```

13. Stop the Bitcoin daemon:

```
bitcoin-cli -regtest stop
```

2. Solidity Program

1. Open Remix IDE: Navigate to Remix IDE.

2. Create New File: In the file explorer, click on the "Create New File" button and name it TokenTransfer.sol.

3. Write Solidity Code: Write the Solidity code into the file.

4. Select Compiler Version: In the "Solidity Compiler" tab, select version 0.8.0.

5. Compile the Contract: Click on the "Compile TokenTransfer.sol" button.

6. Deploy Contract:

Go to the "Deploy & Run Transactions" tab.

Select "Injected Web3" (connect to your MetaMask).

Click "Deploy".

7. Interact with Contract:

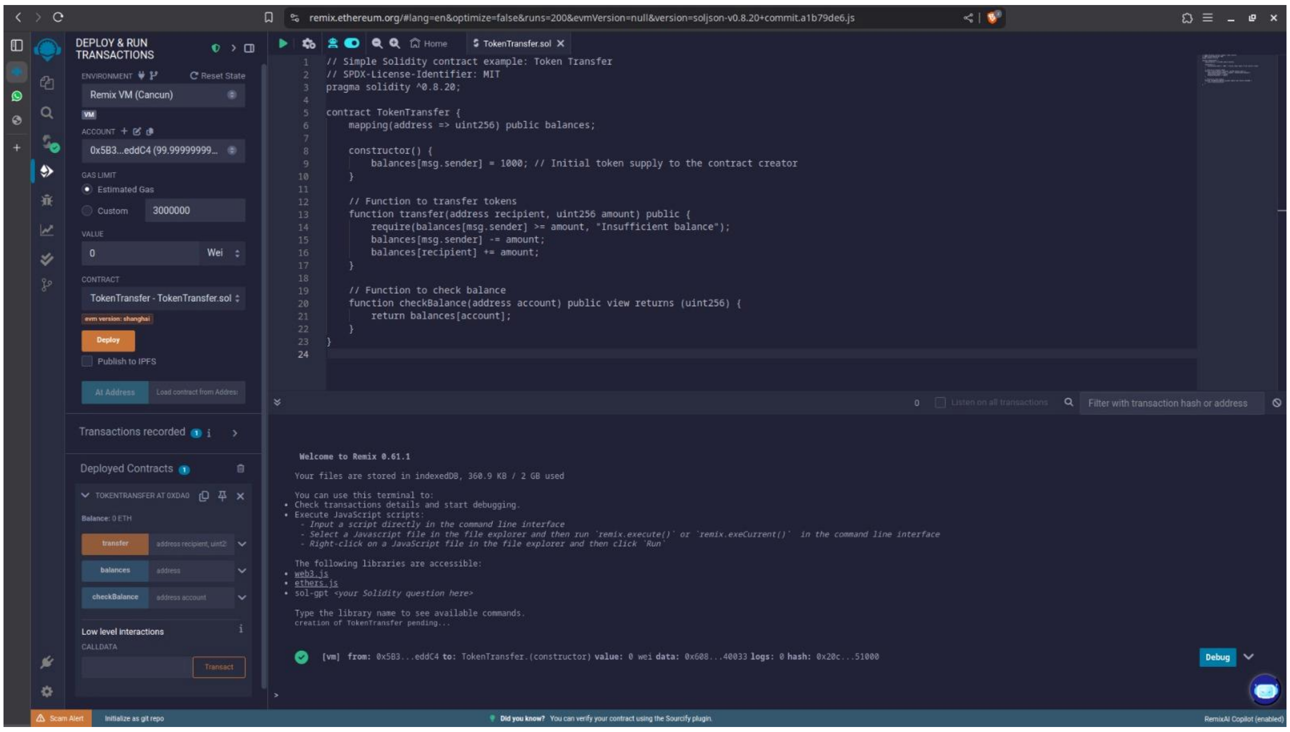
After deployment, the contract's functions will appear.

Use transfer(address, amount) to send tokens and checkBalance(address) to view balances.

Program:

```
pragma solidity ^0.8.0;
contract TokenTransfer {
    mapping(address => uint256) public balances;
    constructor() {
        balances[msg.sender] = 1000;
    }
    function transfer(address recipient, uint256 amount) public {
        require(balances[msg.sender] >= amount, "Insufficient balance");
        balances[msg.sender] -= amount;
        balances[recipient] += amount;
    }
    function checkBalance(address account) public view returns (uint256) {
        return balances[account];
    }
}
```

Output:



```
// Simple Solidity contract example: Token Transfer
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract TokenTransfer {
    mapping(address => uint256) public balances;

    constructor() {
        balances[msg.sender] = 1000; // Initial token supply to the contract creator
    }

    // Function to transfer tokens
    function transfer(address recipient, uint256 amount) public {
        require(balances[msg.sender] >= amount, "Insufficient balance");
        balances[msg.sender] -= amount;
        balances[recipient] += amount;
    }

    // Function to check balance
    function checkBalance(address account) public view returns (uint256) {
        return balances[account];
    }
}
```

```
 Welcome to Remix 0.61.1

Your files are stored in indexedDB, 360.9 KB / 2 GB used

You can use this terminal to:
• Check transactions details and start debugging.
• Execute JavaScript scripts:
  - Input a script directly in the command line interface
  - Select a Javascript file in the file explorer and then run `remix.execute()` or `remix.exeCurrent()`  in the command line interface
  - Right-click on a JavaScript file in the file explorer and then click `Run`

The following libraries are accessible:
• web3.js
• ethers.js
• sol-gpt <your Solidity question here>

Type the library name to see available commands.
creation of TokenTransfer pending...

✓ [vm] from: 0x5B3...eddC4 to: TokenTransfer.(constructor) value: 0 wei data: 0x608...40033 logs: 0 hash: 0x20c...51000
```

Result:

Analyzing of the Bitcoin mining process, types of wallets, and Ethereum Virtual Machine (EVM) while exploring consensus mechanisms and smart contracts' impact on cryptocurrency is successfully completed.

2. B) Understanding cloud basics such as service-oriented architecture (SOA) and virtualization.

Aim:

To understand the fundamental concepts of Cloud Computing, specifically Service-Oriented Architecture (SOA) and Virtualization, and to implement a basic cloud service using a virtualized environment.

Theory:

1. Cloud Computing Overview:

Cloud computing provides on-demand access to computing resources such as servers, storage, and applications over the internet.

The three primary cloud service models are:

IaaS (Infrastructure as a Service): Provides virtualized computing resources (e.g., AWS EC2, Google Compute Engine).

PaaS (Platform as a Service): Provides a development environment with pre-configured tools (e.g., Google App Engine, AWS Lambda).

SaaS (Software as a Service): Provides software applications over the internet (e.g., Gmail, Dropbox).

2. Service-Oriented Architecture (SOA) in Cloud Computing:

SOA in cloud computing helps in designing cloud services as independent, loosely coupled components.

Each service is self-contained and communicates using standard protocols (HTTP, SOAP, REST).

Cloud providers use SOA to offer scalable and modular services (e.g., AWS Lambda, Azure Functions).

3. Virtualization in Cloud Computing:

Definition: Virtualization enables multiple virtual instances of OS, storage, or networks on a single physical machine.

- Server Virtualization (e.g., VMware, Hyper-V)
- Storage Virtualization (e.g., Amazon S3)
- Network Virtualization (e.g., SDN in Google Cloud)

Procedure:

1. Service-Oriented Architecture (SOA) in Cloud Computing

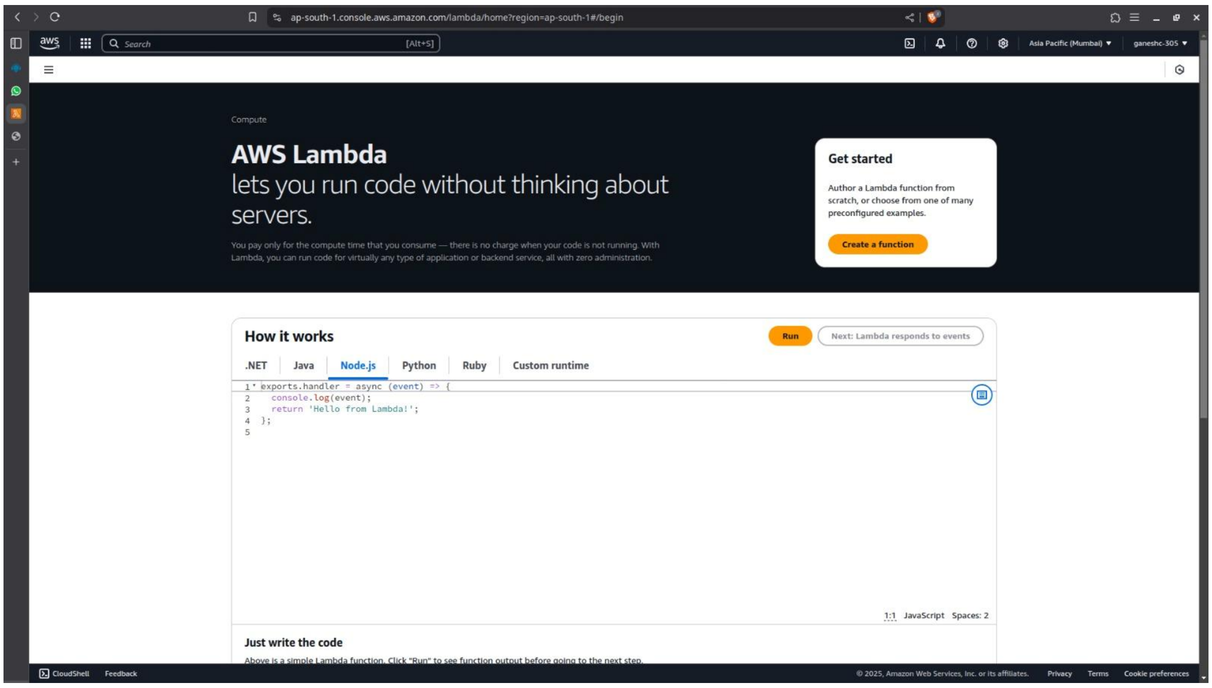
- Log in to a cloud platform like AWS.
- Identify a cloud service that follows SOA principles (e.g., AWS Lambda).
- Observe how cloud services are modular and interact through APIs.

2. Virtualization in Cloud Computing

- Open a virtualization platform (such as AWS EC2).
- Create a new virtual machine (VM) and install an OS (e.g., Amazon Linux on AWS EC2).
- Configure network settings and access the VM via SSH (if cloud-based).

- ➢ Install and host a simple web service inside the virtual machine.
- ➢ Verify that the virtualized environment is working by accessing the hosted service from another system.

Output:



ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/begin
aws
Search [Alt+S]
Asia Pacific (Mumbai) ▾
ganeshc-305 ▾

Compute

AWS Lambda
lets you run code without thinking about servers.

You pay only for the compute time that you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.

Get started

Author a Lambda function from scratch, or choose from one of many preconfigured examples.

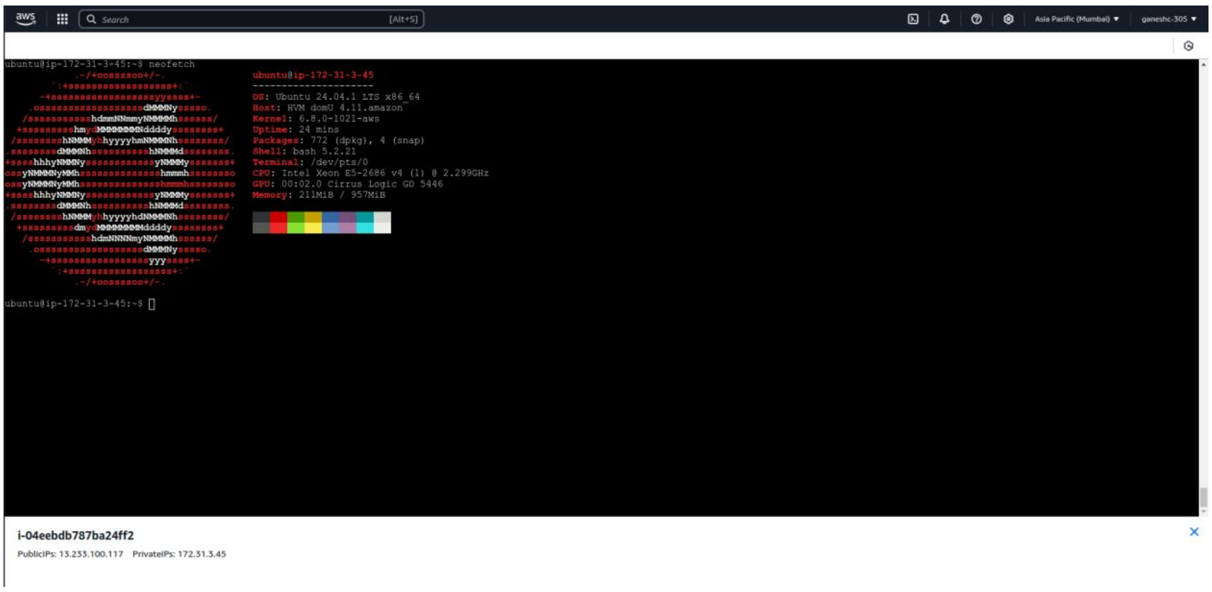
Create a function

How it works

```
exports.handler = async (event) => {
  console.log(event);
  return 'Hello from Lambda!';
};
```

Just write the code

Above is a simple Lambda function. Click "Run" to see function output before going to the next step.



```
ubuntu@ip-172-31-3-45:~$ neofetch
            .-/+oossssoo+/-.               ubuntu@ip-172-31-3-45
        `:+ssssssssssssssssss+:`           ---------------------
      -+ssssssssssssssssssyyssss+-         OS: Ubuntu 24.04.1 LTS x86_64
    .ossssssssssssssssssdMMMNysssso.       Host: HVM domU 4.11.amazon
   /ssssssssssshdmmNNmmyNMMMMhssssss/      Kernel: 6.8.0-1021-aws
  +ssssssssshmydMMMMMMMNddddyssssssss+     Uptime: 24 mins
 /sssssssshNMMMyhhyyyyhmNMMMNhssssssss/    Packages: 772 (dpkg), 4 (snap)
.ssssssssdMMMNhsssssssssshNMMMdssssssss.   Shell: bash 5.2.21
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   Terminal: /dev/pts/0
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   CPU: Intel Xeon E5-2686 v4 (1) @ 2.299GHz
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   GPU: 00:02.0 Cirrus Logic GD 5446
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   Memory: 211MiB / 957MiB
.ssssssssdMMMNhsssssssssshNMMMdssssssss.
 /sssssssshNMMMyhhyyyyhdNMMMNhssssssss/
  +sssssssssdmydMMMMMMMMddddyssssssss+
   /ssssssssssshdmNNNNmyNMMMMhssssss/
    .ossssssssssssssssssdMMMNysssso.
      -+sssssssssssssssssyyyssss+-
        `:+ssssssssssssssssss+:`
            .-/+oossssoo+/-.

ubuntu@ip-172-31-3-45:~$
```

i-04eebdb787ba24ff2

PublicIPs: 13.233.100.117 PrivateIPs: 172.31.3.45

Result:

Understanding cloud basics such as service-oriented architecture (SOA) and virtualization is executed and completed successfully.

3. A) Develop smart contracts in Solidity programming language for Ethereum, comprehend Hyperledger Fabric and Hyperledger Composer

Aim:

To develop smart contracts in Solidity programming language for Ethereum, comprehend Hyperledger Fabric and Hyperledger Composer

Procedure:

Step 1: Open Remix IDE

- Open Remix IDE by visiting https://remix.ethereum.org.
- Click on "File Explorer" and then "New File".
- Name the file AdvancedTransactionManager.sol.

Step 2: Write the Solidity Smart Contract

- Copy and paste the provided Solidity code into the newly created file.

Step 3: Compile the Smart Contract

- Click on the Solidity Compiler tab in Remix IDE.
- Select 0.8.20 as the compiler version.
- Click Compile AdvancedTransactionManager.sol.

Step 4: Deploy the Smart Contract

- Navigate to the Deploy & Run Transactions tab.
- Select "JavaScript VM (London)" as the environment.
- Click Deploy.
- The deployed contract will appear under the Deployed Contracts section.

Step 5: Interacting with the Contract

A. Deposit ETH:

Enter an amount in the "Value" field (e.g., 1 ETH).

Click on the receive function.

B. Transfer ETH:

Enter a recipient address and amount in transferETH(address, amount).

Click transact.

C. Withdraw ETH (Only Owner):

Enter an amount in withdraw(amount).

Click transact.

D. View Contract Balance:

Click getContractBalance().

E. Check Transaction History:

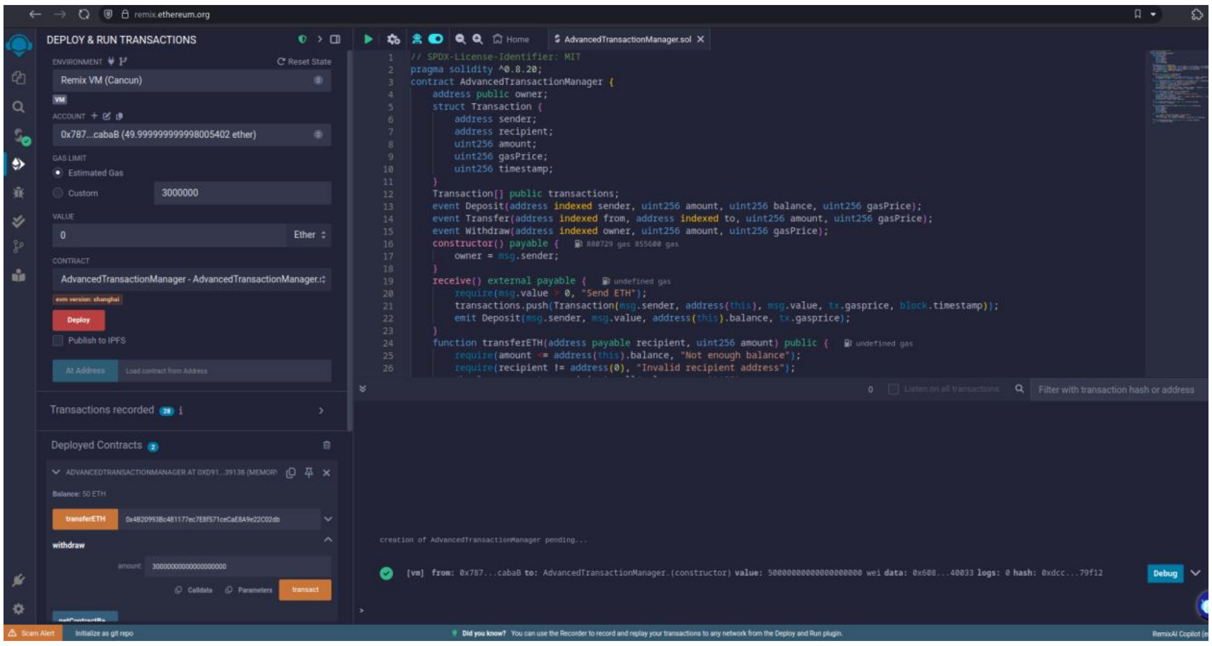
Click getTransactionCount() to check the number of transactions.

Use getTransaction(index) to view specific transaction details.

Program:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
contract AdvancedTransactionManager {
    address public owner;
    struct Transaction {
        address sender;
        address recipient;
        uint256 amount;
        uint256 gasPrice;
        uint256 timestamp;
    }
    Transaction[] public transactions;
    event Deposit(address indexed sender, uint256 amount, uint256 balance, uint256 gasPrice);
    event Transfer(address indexed from, address indexed to, uint256 amount, uint256 gasPrice);
    event Withdraw(address indexed owner, uint256 amount, uint256 gasPrice);
    constructor() payable {
        owner = msg.sender;
    }
    receive() external payable {
        require(msg.value > 0, "Send ETH");
        transactions.push(Transaction(msg.sender, address(this), msg.value, tx.gasprice, block.timestamp));
        emit Deposit(msg.sender, msg.value, address(this).balance, tx.gasprice);
    }
    function transferETH(address payable recipient, uint256 amount) public {
        require(amount <= address(this).balance, "Not enough balance");
        require(recipient != address(0), "Invalid recipient address");
        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Transfer failed");
        transactions.push(Transaction(msg.sender, recipient, amount, tx.gasprice, block.timestamp));
        emit Transfer(msg.sender, recipient, amount, tx.gasprice);
    }
    function withdraw(uint256 amount) public {
        require(msg.sender == owner, "Not the owner");
        require(amount <= address(this).balance, "Insufficient balance");
        uint256 gasPrice = tx.gasprice;
        transactions.push(Transaction(msg.sender, msg.sender, amount, gasPrice, block.timestamp));
        payable(msg.sender).transfer(amount);
        emit Withdraw(msg.sender, amount, gasPrice);
    }
    function getContractBalance() public view returns (uint256) {
        return address(this).balance;
    }
    function getTransaction(uint256 index) public view returns (
        address sender,
        address recipient,
        uint256 amount,
        uint256 gasPrice,
        uint256 timestamp
    ) {
        require(index < transactions.length, "Invalid index");
        Transaction memory txn = transactions[index];
        return (txn.sender, txn.recipient, txn.amount, txn.gasPrice, txn.timestamp);
    }
    function getTransactionCount() public view returns (uint256) {
        return transactions.length;
    } }
```

Output:



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
contract AdvancedTransactionManager {
    address public owner;
    struct Transaction {
        address sender;
        address recipient;
        uint256 amount;
        uint256 gasPrice;
        uint256 timestamp;
    }
    Transaction[] public transactions;
    event Deposit(address indexed sender, uint256 amount, uint256 balance, uint256 gasPrice);
    event Transfer(address indexed from, address indexed to, uint256 amount, uint256 gasPrice);
    event Withdraw(address indexed owner, uint256 amount, uint256 gasPrice);
    constructor() payable {    880729 gas 855600 gas
        owner = msg.sender;
    }
    receive() external payable {    undefined gas
        require(msg.value > 0, "Send ETH");
        transactions.push(Transaction(msg.sender, address(this), msg.value, tx.gasprice, block.timestamp));
        emit Deposit(msg.sender, msg.value, address(this).balance, tx.gasprice);
    }
    function transferETH(address payable recipient, uint256 amount) public {    undefined gas
        require(amount <= address(this).balance, "Not enough balance");
        require(recipient != address(0), "Invalid recipient address");
```

```
creation of AdvancedTransactionManager pending...

[vm] from: 0x787...cabaB to: AdvancedTransactionManager.(constructor) value: 5000000000000000000 wei data: 0x608...40033 logs: 0 hash: 0xdcc...79f12
```

ENVIRONMENT: Remix VM (Cancun)

ACCOUNT: 0x787...cabaB (49.999999999998005402 ether)

GAS LIMIT: [✓ Estimated Gas] Custom 3000000

VALUE: 0 Ether

CONTRACT: AdvancedTransactionManager - AdvancedTransactionManager.s

evm version: shanghai

Transactions recorded 28

Deployed Contracts 2

ADVANCEDTRANSACTIONMANAGER AT 0XD91...39138 (MEMORY)

Balance: 50 ETH

transferETH 0x4820993Bc481177ec7E8f571ceCaE8A9e22C02db

withdraw

amount: 3000000000000000000

Did you know? You can use the Recorder to record and replay your transactions to any network from the Deploy and Run plugin.

remix.ethereum.org
DEPLOY & RUN TRANSACTIONS
Reset State
Deploy
Publish to IPFS
At Address
Load contract from Address
Calldata
Parameters
transact
Home
AdvancedTransactionManager.sol
Listen on all transactions
Filter with transaction hash or address
Debug
Scam Alert
Initialize as git repo
RemixAI Copilot

Result:

Developing a smart contracts in Solidity programming language for Ethereum, comprehend Hyperledger Fabric and Hyperledger Composer is successfully completed

3.B) Design layered cloud architecture considering cloud services and service models.

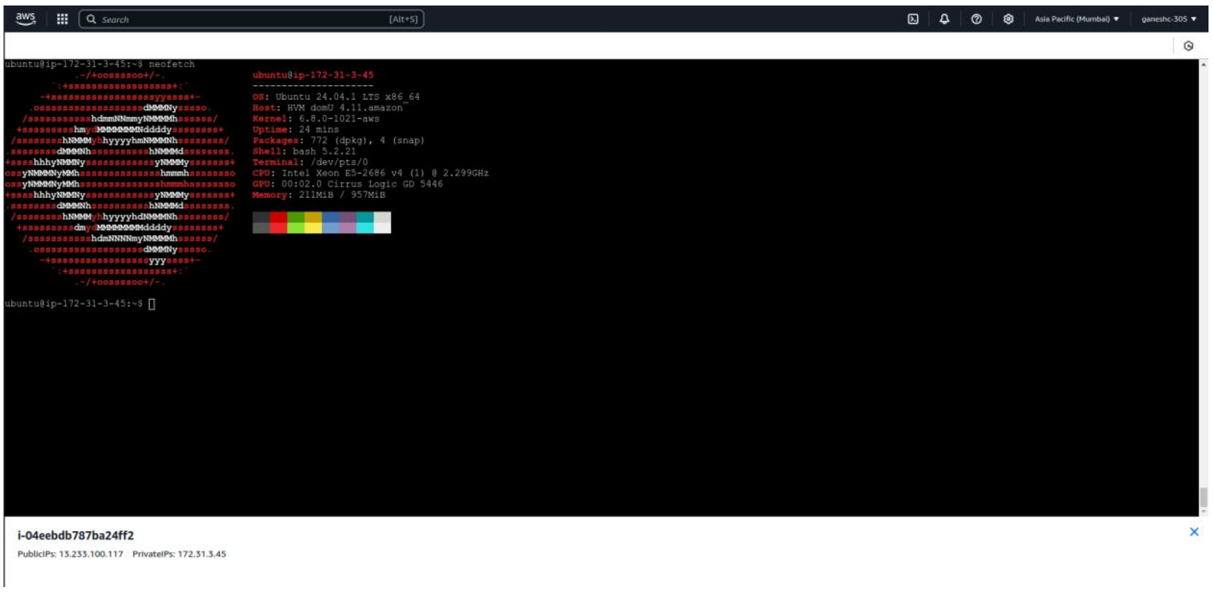
Aim:

To design layered cloud architecture considering cloud services and service models.

Procedure:

1. Identify the Layers in Cloud Architecture:
    - Infrastructure Layer (IaaS): Virtual machines, storage, networking.
    - Platform Layer (PaaS): Database services, development frameworks.
    - Application Layer (SaaS): End-user applications like web apps, email services.
2. Setting Up AWS Infrastructure (IaaS):
    - Log in to AWS Console (https://aws.amazon.com/console/).
    - Navigate to EC2 Dashboard and launch a new instance.
    - Choose an appropriate Amazon Machine Image (AMI) (e.g., Ubuntu 22.04 LTS).
    - Select an instance type (e.g., t2.micro for free tier eligibility).
    - Configure instance settings, security groups, and assign a key pair.
    - Start the instance and connect via SSH.
3. Deploying a Platform Layer Service (PaaS):
    - Navigate to AWS RDS (Relational Database Service).
    - Click on "Create Database" and select MySQL as the engine.
    - Configure settings, storage, and authentication credentials.
    - Launch the database and connect it to an EC2 instance.
4. Configuring Application Services (SaaS):
    - Navigate to AWS S3 to create a new storage bucket.
    - Upload a sample static website and enable public access.
    - Use AWS CloudFront to distribute content globally with CDN.
    - Integrate with AWS Lambda for serverless computing tasks.
5. Testing and Monitoring:
    - Use AWS CloudWatch to monitor EC2 instance performance.
    - Check RDS database connectivity using MySQL Workbench.
    - Access the S3-hosted website via a public URL.

Output:



```
ubuntu@ip-172-31-3-45:~$ neofetch
            .-/+oossssoo+/-.               ubuntu@ip-172-31-3-45
        `:+ssssssssssssssssss+:`           ---------------------
      -+ssssssssssssssssssyyssss+-         OS: Ubuntu 24.04.1 LTS x86_64
    .ossssssssssssssssssdMMMNysssso.       Host: HVM domU 4.11.amazon
   /ssssssssssshdmmNNmmyNMMMMhssssss/      Kernel: 6.8.0-1021-aws
  +ssssssssshmydMMMMMMMNddddyssssssss+     Uptime: 24 mins
 /sssssssshNMMMyhhyyyyhmNMMMNhssssssss/    Packages: 772 (dpkg), 4 (snap)
.ssssssssdMMMNhsssssssssshNMMMdssssssss.   Shell: bash 5.2.21
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   Terminal: /dev/pts/0
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   CPU: Intel Xeon E5-2686 v4 (1) @ 2.299GHz
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   GPU: 00:02.0 Cirrus Logic GD 5446
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   Memory: 211MiB / 957MiB
.ssssssssdMMMNhsssssssssshNMMMdssssssss.
 /sssssssshNMMMyhhyyyyhdNMMMNhssssssss/
  +sssssssssdmydMMMMMMMMddddyssssssss+
   /ssssssssssshdmNNNNmyNMMMMhssssss/
    .ossssssssssssssssssdMMMNysssso.
      -+sssssssssssssssssyyyssss+-
        `:+ssssssssssssssssss+:`
            .-/+oossssoo+/-.

ubuntu@ip-172-31-3-45:~$ 
```

i-04eebdb787ba24ff2

PublicIPs: 13.233.100.117 PrivateIPs: 172.31.3.45

Result:

Designing a layered cloud architecture considering cloud services and service models is successfully completed.

4.A) Implement Solidity programming language features like variables, functions, and data structures, alongside understanding Ethereum wallet and smart contracts structure

Aim:

To implement Solidity programming language features like variables, functions, and data structures, alongside understanding Ethereum wallet and smart contracts structure

Procedure:

Step 1: Install MetaMask

1. Open the Google Chrome or Mozilla Firefox browser.
2. Search for MetaMask extension and install it.
3. Open the extension and create a new Ethereum wallet.
4. Securely store the Seed Phrase as it is crucial for wallet recovery.
5. Switch the network to Localhost 8545 to connect to Ganache.

Step 2: Install Ganache

1. Download and install Ganache from Truffle Suite.
2. Open Ganache and create a new workspace.
3. Copy the RPC Server URL (default: http://127.0.0.1:7545).
4. Note down the private keys of the pre-funded test accounts.

Step 3: Connect MetaMask to Ganache

1. Open MetaMask → Click on Network Selector → Add Network Manually.
2. Enter:

    Network Name: Ganache

    New RPC URL: http://127.0.0.1:7545

    Chain ID: 1337

    Currency Symbol: ETH

3. Click Save and switch to the Ganache network.
4. Import an account from Ganache using the private key.

Step 4: Open Remix IDE

1. Visit Remix IDE (https://remix.ethereum.org/).
2. Create a new Solidity file (contract.sol).
3. Write a basic smart contract implementing variables, functions, and data structures.

Step 5: Compile the Contract

1. In Remix IDE, go to the Solidity Compiler tab.
2. Select the Solidity version 0.8.0+.
3. Click Compile contract.sol.

Step 6: Deploy the Contract Locally

1. Go to the Deploy & Run Transactions tab.
2. Select Environment: Web3 Provider.
3. Enter http://127.0.0.1:7545 (Ganache RPC URL) and connect.
4. Click Deploy.
5. Wait for confirmation.

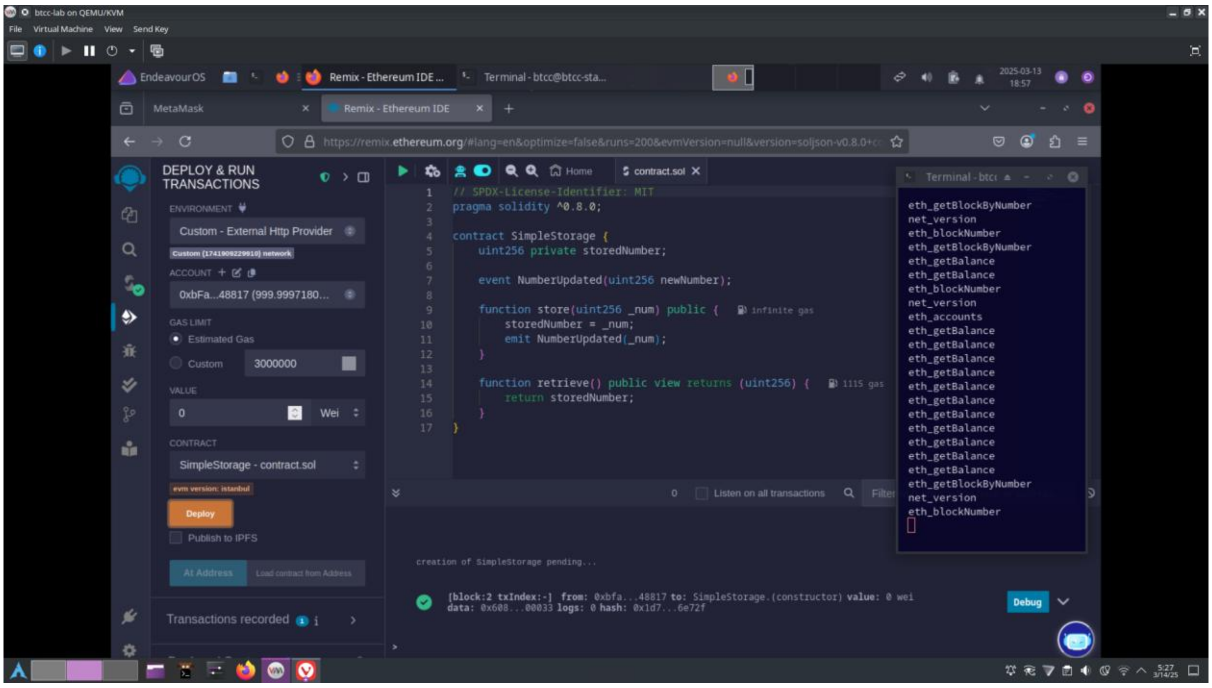
Step 7: Interact with the Contract

1. In Remix, under the deployed contract section:
   - Call store(25) to store a number.
   - Call retrieve() to get the stored number.
2. Observe the transaction details in Remix.

Program:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract SimpleStorage {
   uint256 private storedNumber;
   event NumberUpdated(uint256 newNumber);
   function store(uint256 _num) public {
      storedNumber = _num;
      emit NumberUpdated(_num);
   }
   function retrieve() public view returns (uint256) {
     return storedNumber;
   }
}
```

Output:



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Custom - External Http Provider

ACCOUNT: 0xbFa...48817 (999.9997180...)

GAS LIMIT: Estimated Gas / Custom 3000000

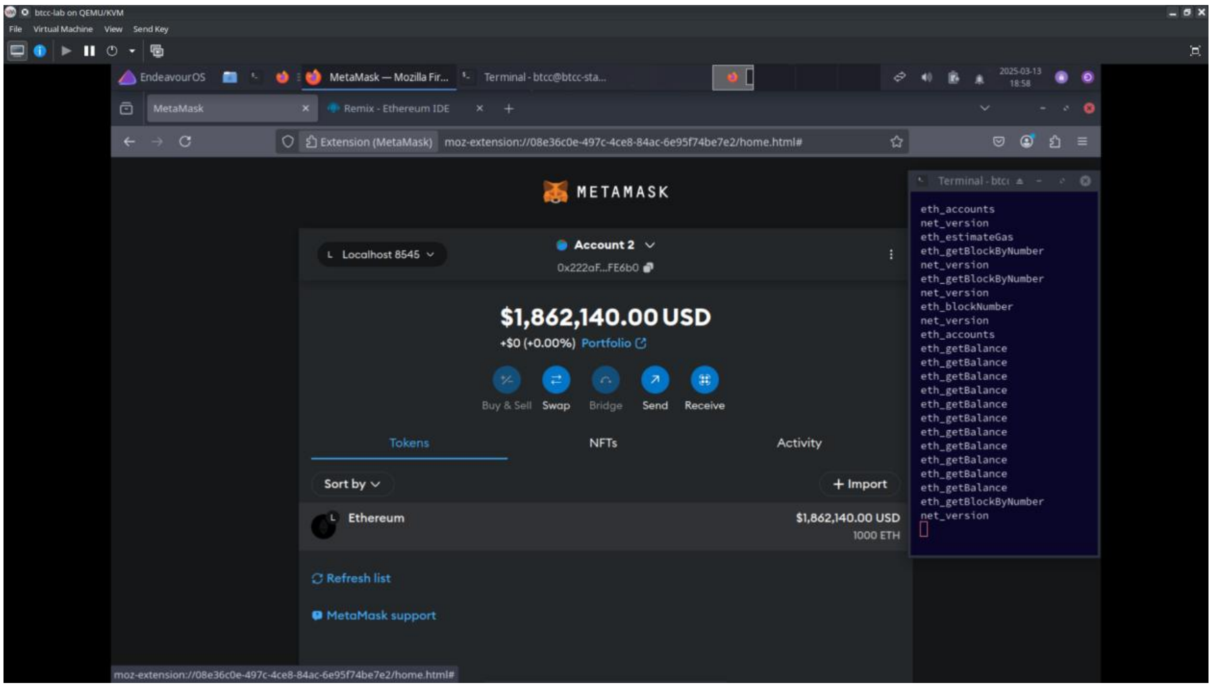
VALUE: 0 Wei

CONTRACT: SimpleStorage - contract.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleStorage {
    uint256 private storedNumber;

    event NumberUpdated(uint256 newNumber);

    function store(uint256 _num) public {    infinite gas
        storedNumber = _num;
        emit NumberUpdated(_num);
    }

    function retrieve() public view returns (uint256) {    1115 gas
        return storedNumber;
    }
}
```

```
creation of SimpleStorage pending...

[block:2 txIndex:-] from: 0xbfa...48817 to: SimpleStorage.(constructor) value: 0 wei
data: 0x608...00033 logs: 0 hash: 0x1d7...6e72f
```

```
eth_getBlockByNumber
net_version
eth_blockNumber
eth_getBlockByNumber
eth_getBalance
eth_getBalance
eth_blockNumber
net_version
eth_accounts
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBlockByNumber
net_version
eth_blockNumber
```



METAMASK

Localhost 8545 — Account 2 — 0x222aF...FE6b0

$1,862,140.00 USD

+$0 (+0.00%) Portfolio

Ethereum — $1,862,140.00 USD — 1000 ETH

```
eth_accounts
net_version
eth_estimateGas
eth_getBlockByNumber
net_version
eth_getBlockByNumber
net_version
eth_blockNumber
net_version
eth_accounts
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBalance
eth_getBlockByNumber
net_version
```

Result:

Implementing Solidity programming language features like variables, functions, and data structures, alongside understanding Ethereum wallet and smart contracts structure is successfuly completed.

4.B) Ensure cloud security through identity & access management (IAM) and compliance with security standards.

Aim:

To ensure cloud security through identity & access management (IAM) and compliance with security standards.

Procedure:

Step 1: Login to AWS Console

1. Go to https://aws.amazon.com/.
2. Click on Sign in to the Console.
3. Enter your credentials to log in.

Step 2: Navigate to IAM

1. In the AWS Management Console, search for IAM (Identity & Access Management).
2. Click on IAM Dashboard.

Step 3: Create an IAM User

1. Click Users → Add User.
2. Enter the username CloudUser.
3. Select Programmatic access and AWS Management Console access.
4. Click Next: Permissions.

Step 4: Assign IAM Permissions

1. Choose Attach existing policies directly.
2. Select AmazonS3FullAccess to grant access to S3 services.
3. Click Next: Review → Create User.
4. Note down the Access Key ID and Secret Access Key.

Step 5: Create an IAM Group and Assign Policies

1. Go to Groups → Create New Group.
2. Name the group Developers.
3. Attach AdministratorAccess policy.
4. Add the CloudUser to this group.

Step 6: Set IAM Compliance Policies

1. Go to Policies.
2. Create a new policy CloudPolicy
3. Attach this policy to enforce MFA.

4. Validate compliance settings.

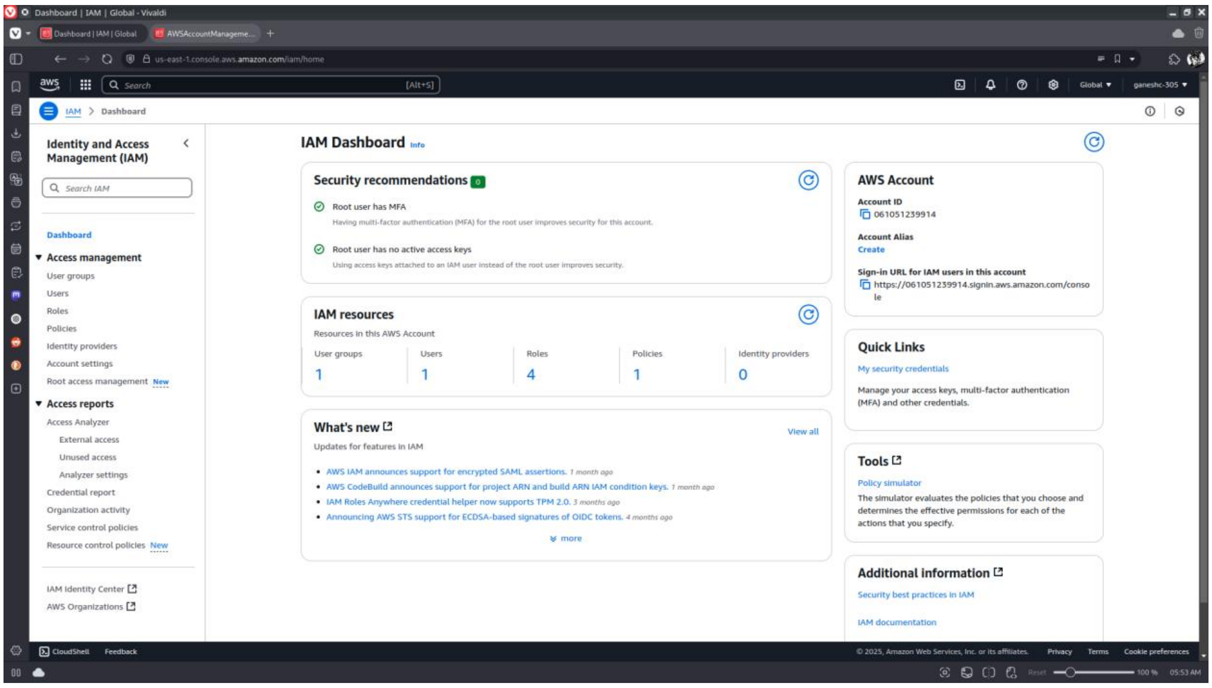
Step 7: Verify IAM Setup

1. Sign in with CloudUser credentials.
2. Try accessing S3 without MFA (should be denied).
3. Enable MFA and retry (should be allowed).

Program:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "s3:*",
      "Resource": "*",
      "Condition": {
        "Bool": { "aws:MultiFactorAuthPresent": "false" }
      }
    }
  ]
}
```

Output:



IAM Dashboard Info

Security recommendations 0

Root user has MFA
Having multi-factor authentication (MFA) for the root user improves security for this account.

Root user has no active access keys
Using access keys attached to an IAM user instead of the root user improves security.

IAM resources
Resources in this AWS Account

User groups	Users	Roles	Policies	Identity providers
1	1	4	1	0

What's new
Updates for features in IAM
View all

- AWS IAM announces support for encrypted SAML assertions. 1 month ago
- AWS CodeBuild announces support for project ARN and build ARN IAM condition keys. 1 month ago
- IAM Roles Anywhere credential helper now supports TPM 2.0. 3 months ago
- Announcing AWS STS support for ECDSA-based signatures of OIDC tokens. 4 months ago

more

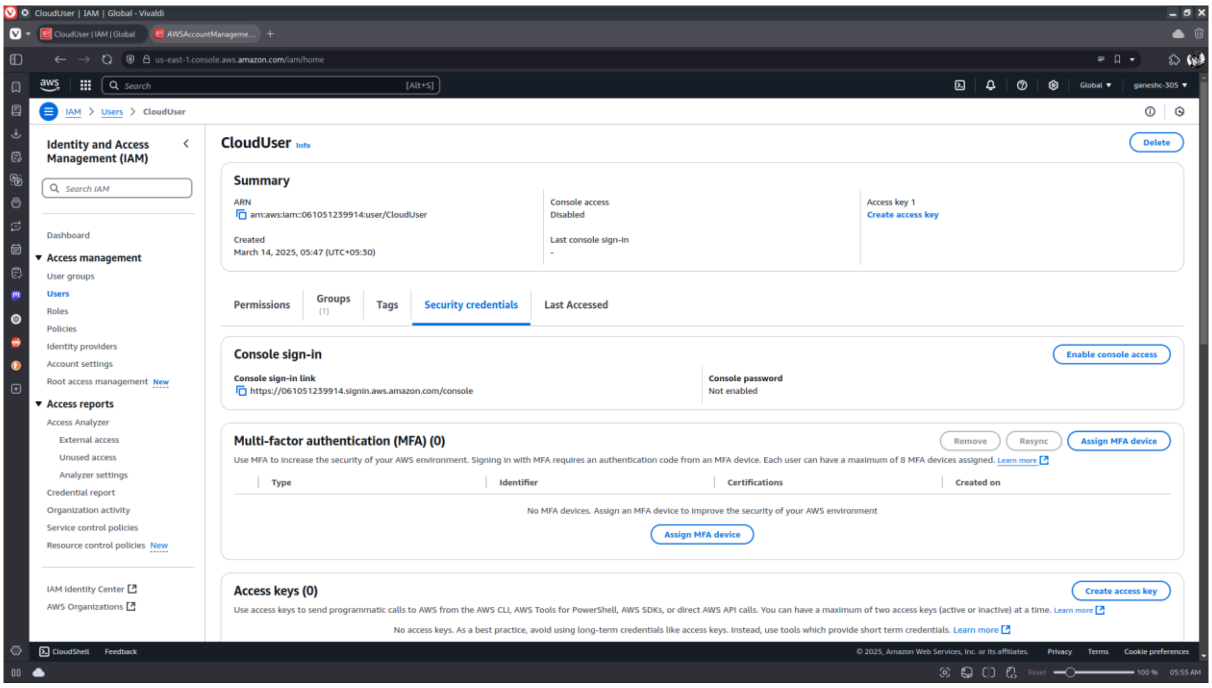
AWS Account
Account ID
061051239914
Account Alias
Create
Sign-in URL for IAM users in this account
https://061051239914.signin.aws.amazon.com/console

Quick Links
My security credentials
Manage your access keys, multi-factor authentication (MFA) and other credentials.

Tools
Policy simulator
The simulator evaluates the policies that you choose and determines the effective permissions for each of the actions that you specify.

Additional information
Security best practices in IAM
IAM documentation

© 2025, Amazon Web Services, Inc. or its affiliates.



CloudUser Info

Summary
ARN
arn:aws:iam::061051239914:user/CloudUser
Created
March 14, 2025, 05:47 (UTC+05:30)
Console access
Disabled
Last console sign-in
-
Access key 1
Create access key

Console sign-in
Console sign-in link
https://061051239914.signin.aws.amazon.com/console
Console password
Not enabled

Multi-factor authentication (MFA) (0)
Use MFA to increase the security of your AWS environment. Signing in with MFA requires an authentication code from an MFA device. Each user can have a maximum of 8 MFA devices assigned. Learn more

Type	Identifier	Certifications	Created on

No MFA devices. Assign an MFA device to improve the security of your AWS environment

Access keys (0)
Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. Learn more

No access keys. As a best practice, avoid using long-term credentials like access keys. Instead, use tools which provide short term credentials. Learn more

© 2025, Amazon Web Services, Inc. or its affiliates.

Result:

Ensuring cloud security through identity & access management (IAM) and compliance with security standards is successfully done.

5. Explore blockchain applications like IoT integration and medical record management,analyze alternative cryptocurrencies (Alt Coins) and their significance, and investigate

Aim:

To explore blockchain applications like IoT integration and medical record management,analyze alternative cryptocurrencies (Alt Coins) and their significance, and investigate

Procedure:

Step 1: Exploring Blockchain Applications

1. IoT Integration:
    - Blockchain enhances IoT security by providing decentralized data storage and eliminating single points of failure.
    - It enables secure device-to-device communication through smart contracts, ensuring data integrity and transparency.
    - Example: Blockchain-based authentication for smart home devices prevents unauthorized access.
2. Medical Record Management:
    - Blockchain ensures secure and tamper-proof storage of medical records.
    - Patients control access to their records using cryptographic keys, ensuring privacy.
    - Example: A hospital using blockchain to manage patient records reduces data breaches and enhances interoperability.

Step 2: Understanding Alternative Cryptocurrencies (Altcoins)

1. Definition and Differences from Bitcoin:
    - Altcoins are cryptocurrencies other than Bitcoin, designed to improve upon Bitcoin's limitations.
    - Examples include Ethereum, Litecoin, and Ripple, which offer unique features such as smart contracts and faster transactions.
2. Real-World Applications of Altcoins:
    - Ethereum (ETH): Enables decentralized applications (DApps) and smart contracts.
    - Litecoin (LTC): Provides faster transactions with lower fees than Bitcoin.
    - Ripple (XRP): Focuses on real-time cross-border payments.

Step 3: Deploying a Smart Contract on Ethereum

Setting Up the Environment

1. Open Remix IDE in a web browser.
2. Click on File Explorer and create a new Solidity file named AltcoinContract.sol.
3. Ensure the Solidity compiler version is set to 0.8.0 or later.

Step 4: Compiling and Deploying the Contract

Click on the Solidity Compiler tab in Remix IDE.

Select the compiler version matching pragma solidity ^0.8.0;.

Click Compile AltcoinContract.sol .

Open the Deploy & Run Transactions tab.

Select JavaScript VM (Cancun) as the environment to use Remix's built-in blockchain simulator.

Click Deploy, specifying the initial token supply (e.g., 1000 tokens).

7. The contract will be deployed in the Remix VM without requiring a real blockchain network.

Step 5: Testing the Contract

1. After deployment, verify the contract functions in Remix.

Click on the name(), symbol(), and totalSupply() functions to confirm the token details.

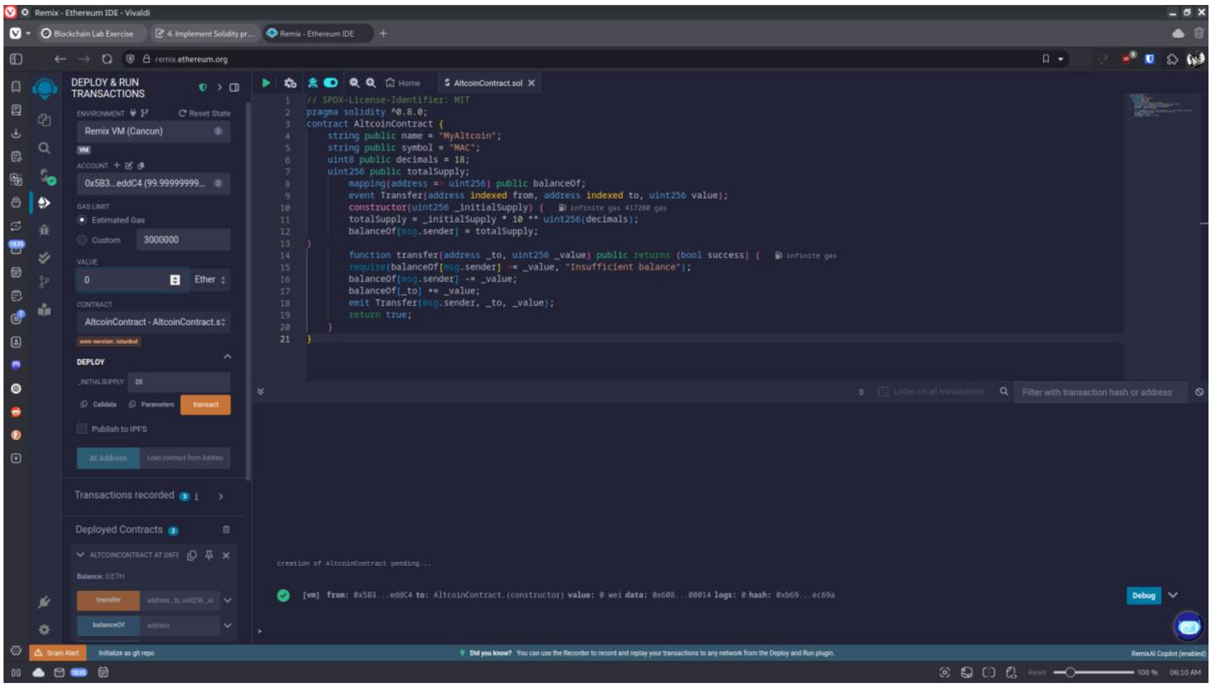
Use the transfer() function to send tokens between accounts and verify balance updates.

Program:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract AltcoinContract {
  string public name = "MyAltcoin";
  string public symbol = "MAC";
  uint8 public decimals = 18;
  uint256 public totalSupply;
    mapping(address => uint256) public balanceOf;
    event Transfer(address indexed from, address indexed to, uint256 value);
    constructor(uint256 _initialSupply) {
    totalSupply = _initialSupply * 10 ** uint256(decimals);
    balanceOf[msg.sender] = totalSupply;
}
    function transfer(address _to, uint256 _value) public returns (bool success) {
    require(balanceOf[msg.sender] >= _value, "Insufficient balance");
    balanceOf[msg.sender] -= _value;
    balanceOf[_to] += _value;
```

```
        emit Transfer(msg.sender, _to, _value);
        return true;
    }
}
```

Output:



ENVIRONMENT: Remix VM (Cancun)
ACCOUNT: 0x5B3...eddC4 (99.99999999...
GAS LIMIT: Estimated Gas; Custom 3000000
VALUE: 0 Ether
CONTRACT: AltcoinContract - AltcoinContract.s
evm version: istanbul
DEPLOY
_INITIALSUPPLY: 20
Calldata Parameters transact
Publish to IPFS
At Address Load contract from Address
Transactions recorded 3
Deployed Contracts 2
ALTCOINCONTRACT AT 0XFE
Balance: 0 ETH
transfer address _to, uint256 _va
balanceOf address

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract AltcoinContract {
    string public name = "MyAltcoin";
    string public symbol = "MAC";
    uint8 public decimals = 18;
    uint256 public totalSupply;
        mapping(address => uint256) public balanceOf;
        event Transfer(address indexed from, address indexed to, uint256 value);
        constructor(uint256 _initialSupply) {    infinite gas 417200 gas
        totalSupply = _initialSupply * 10 ** uint256(decimals);
        balanceOf[msg.sender] = totalSupply;
}
        function transfer(address _to, uint256 _value) public returns (bool success) {    infinite gas
        require(balanceOf[msg.sender] >= _value, "Insufficient balance");
        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;
        emit Transfer(msg.sender, _to, _value);
        return true;
    }
}
```

```
creation of AltcoinContract pending...

[vm] from: 0x5B3...eddC4 to: AltcoinContract.(constructor) value: 0 wei data: 0x608...00014 logs: 0 hash: 0xb69...ec69a
```

Did you know? You can use the Recorder to record and replay your transactions to any network from the Deploy and Run plugin.

Result:

Explore blockchain applications like IoT integration and medical record management,analyze alternative cryptocurrencies (Alt Coins) and their significance, and investigate is successfully completed.