**DSCEngine API Documentation (evm.py)**

**Overview**

The evm.py script is a FastAPI application that provides a RESTful API for interacting with the DSCEngine smart contract on the Ethereum Sepolia testnet (chain ID 11155111). The contract manages a decentralized stablecoin (DSC) protocol, allowing users to deposit collateral (WETH or WAVAX), mint DSC, redeem collateral, burn DSC, liquidate undercollateralized positions, and query account information. The API uses web3.py to interact with the blockchain and relies on environment variables for configuration.

This document is intended for:

* **Frontend Engineers**: To integrate the API into user interfaces, understand endpoints, inputs, outputs, and error handling.
* **AI Engineers**: To understand contract interactions, extend functionality (e.g., for Avalanche Fuji), and debug issues like transaction reverts or library errors.

**Setup**

**Prerequisites**

* **Python**: 3.8+
* **Dependencies**: Install via pip:
* pip install fastapi uvicorn web3 python-dotenv pydantic

Ensure web3.py version is ≥6.0.0 to avoid issues like AttributeError: type object 'Web3' has no attribute 'eth'.

pip show web3

* **Sepolia Node**: Access to a Sepolia RPC endpoint (e.g., Infura, Alchemy).
* **Environment Variables**: Create a .env file in the project root:
* SEPOLIA\_RPC\_URL=https://sepolia.infura.io/v3/YOUR\_PROJECT\_ID
* PRIVATE\_KEY=0xYOUR\_PRIVATE\_KEY
* SEPOLIA\_DSCEngine\_ADDRESS=0x...
* SEPOLIA\_WAVAX\_ADDRESS=0x...
* SEPOLIA\_WETH\_ADDRESS=0x...
* SEPOLIA\_DSC\_ADDRESS=0x...
  + SEPOLIA\_RPC\_URL: Sepolia node endpoint.
  + PRIVATE\_KEY: Ethereum private key (with 0x prefix) for signing transactions.
  + SEPOLIA\_DSCEngine\_ADDRESS: DSCEngine contract address on Sepolia.
  + SEPOLIA\_WAVAX\_ADDRESS: WAVAX token contract address (wrapped AVAX).
  + SEPOLIA\_WETH\_ADDRESS: WETH token contract address (wrapped ETH).
  + SEPOLIA\_DSC\_ADDRESS: DSC token contract address.

**Running the API**

1. Save the script as evm.py in your project directory (e.g., /mnt/d/chainlink-CrossChain-Stablecoin/Base-Hackathon-Stable-Token/dsc-api).
2. Ensure the ABI file exists at:
3. ../dsc-foundry-StableToken/out/DSCEngine.sol/DSCEngine.json

This file contains the DSCEngine contract ABI, generated by Foundry.

1. Start the FastAPI server:
2. uvicorn evm:app --host 0.0.0.0 --port 8000 --reload
3. Access the API at http://localhost:8000. Use /docs for the Swagger UI.

* **Missing ABI File**:
  + Ensure the DSCEngine.json file is in the correct path. If missing, compile the contract using Foundry:
  + cd dsc-foundry-StableToken
  + forge build

**Endpoints**

**1. Approve Tokens (/approve-tokens)**

* **Method**: POST
* **Description**: Approves a spender to spend WAVAX, WETH, or DSC tokens on behalf of the user.
* **Request Body**:
* {
* "token\_address": "0x... (WAVAX, WETH, or DSC address)",
* "spender\_address": "0x... (e.g., DSCEngine address)",
* "amount": 1.0
* }
* **Response**:
  + Success (approval needed):
  + {"WAVAX\_approval": "0x... (tx hash)"}
  + Success (no approval needed):
  + {"message": "No approval needed for WAVAX; sufficient allowance already set"}
  + Error:
  + {"detail": "Invalid token address: 0x..."}
* **Status Codes**:
  + 200: Success
  + 400: Invalid input (e.g., invalid address, non-positive amount)
  + 500: Transaction failure or server error
* **Notes**:
  + Validates token address against WAVAX, WETH, or DSC.
  + Checks existing allowance to avoid redundant approvals.

**2. Deposit ETH to WETH (/deposit-eth-to-weth)**

* **Method**: POST
* **Description**: Sends ETH to the WETH contract to mint WETH, triggering the receive() function.
* **Request Body**:
* {
* "amount": 0.1
* }
* **Response**:
* {
* "tx\_hash": "0x...",
* "eth\_balance\_before": 1.5,
* "eth\_balance\_after": 1.39,
* "weth\_balance\_before": 0.0,
* "weth\_balance\_after": 0.1,
* "events": {
* "deposit\_events": [{"address": "0x...", "amount": 0.1}],
* "transfer\_events": [{"from": "0x...", "to": "0x...", "amount": 0.1}]
* },
* "tx\_data": "0x"
* }
* **Status Codes**:
  + 200: Success
  + 400: Insufficient ETH balance, invalid amount
  + 500: Transaction revert or server error
* **Notes**:
  + Checks ETH balance before sending.
  + Parses Deposit and Transfer events to verify WETH minting.
  + Uses gas limit of 100,000 and dynamic gas pricing.

**3. Withdraw ETH and Burn WETH (/withdraw-eth-and-burn-weth)**

* **Method**: POST
* **Description**: Burns WETH to withdraw ETH by calling withdrawEthAndBurnWETH.
* **Request Body**:
* {
* "amount": 0.1
* }
* **Response**:
* {
* "tx\_hash": "0x...",
* "eth\_balance\_before": 1.39,
* "eth\_balance\_after": 1.49,
* "weth\_balance\_before": 0.1,
* "weth\_balance\_after": 0.0,
* "events": {
* "burned\_and\_withdrawn\_events": [{"user": "0x...", "amount": 0.1, "status": "success"}],
* "transfer\_events": [{"from": "0x...", "to": "0x...", "amount": 0.1}]
* }
* }
* **Status Codes**:
  + 200: Success
  + 400: Insufficient WETH or ETH (for gas), insufficient contract ETH
  + 500: Transaction revert or server error
* **Notes**:
  + Checks WETH balance and contract ETH balance.
  + Parses BurnedAndWithdrawn and Transfer events.
  + Uses gas limit of 300,000.

**4. Deposit Collateral (/deposit-collateral)**

* **Method**: POST
* **Description**: Deposits WAVAX or WETH as collateral into DSCEngine.
* **Request Body**:
* {
* "token\_address": "0x... (WAVAX or WETH)",
* "amount": 0.5
* }
* **Response**:
* {"tx\_hash": "0x..."}
* **Status Codes**:
  + 200: Success
  + 400: Invalid token address, non-positive amount, insufficient allowance
  + 500: Transaction revert or server error
* **Notes**:
  + Requires prior approval (/approve-tokens).

**5. Mint DSC (/mint-dsc)**

* **Method**: POST
* **Description**: Mints DSC based on deposited collateral.
* **Request Body**:
* {
* "amount": 100.0
* }
* **Response**:
* {"tx\_hash": "0x..."}
* **Status Codes**:
  + 200: Success
  + 400: Non-positive amount
  + 500: Transaction revert (e.g., insufficient collateral)
* **Notes**:
  + Ensure sufficient collateral and health factor.

**6. Redeem Collateral (/redeem-collateral)**

* **Method**: POST
* **Description**: Redeems WAVAX or WETH collateral from DSCEngine.
* **Request Body**:
* {
* "token\_address": "0x... (WAVAX or WETH)",
* "amount": 0.5
* }
* **Response**:
* {"tx\_hash": "0x..."}
* **Status Codes**:
  + 200: Success
  + 400: Invalid token address, non-positive amount
  + 500: Transaction revert (e.g., insufficient collateral)
* **Notes**:
  + Checks health factor to avoid liquidation.

**7. Burn DSC (/burn-dsc)**

* **Method**: POST
* **Description**: Burns DSC to reduce debt.
* **Request Body**:
* {
* "amount": 50.0
* }
* **Response**:
* {"tx\_hash": "0x..."}
* **Status Codes**:
  + 200: Success
  + 400: Non-positive amount, insufficient allowance
  + 500: Transaction revert
* **Notes**:
  + Requires prior approval for DSC.

**8. Deposit Collateral and Mint DSC (/deposit-collateral-and-mint-dsc)**

* **Method**: POST
* **Description**: Combines collateral deposit and DSC minting.
* **Request Body**:
* {
* "token\_address": "0x... (WAVAX or WETH)",
* "amount": 0.5,
* "amount\_dsc\_to\_mint": 100.0
* }
* **Response**:
* {"tx\_hash": "0x..."}
* **Status Codes**:
  + 200: Success
  + 400: Invalid token address, non-positive amounts, insufficient allowance
  + 500: Transaction revert

**9. Redeem Collateral for DSC (/redeem-collateral-for-dsc)**

* **Method**: POST
* **Description**: Redeems collateral by burning DSC.
* **Request Body**:
* {
* "token\_address": "0x... (WAVAX or WETH)",
* "amount": 0.5,
* "amount\_dsc\_to\_burn": 100.0
* }
* **Response**:
* {"tx\_hash": "0x..."}
* **Status Codes**:
  + 200: Success
  + 400: Invalid token address, non-positive amounts, insufficient allowance
  + 500: Transaction revert

**10. Liquidate (/liquidate)**

* **Method**: POST
* **Description**: Liquidates an undercollateralized user’s position.
* **Request Body**:
* {
* "collateral": "0x... (WAVAX or WETH)",
* "user": "0x...",
* "debt\_to\_cover": 50.0
* }
* **Response**:
* {"tx\_hash": "0x..."}
* **Status Codes**:
  + 200: Success
  + 400: Invalid addresses, non-positive debt
  + 500: Transaction revert

**11. Account Information (/account-information)**

* **Method**: POST
* **Description**: Retrieves DSC minted and collateral value for a user.
* **Request Body**:
* {
* "user": "0x..."
* }
* **Response**:
* {
* "total\_dsc\_minted": 100.0,
* "collateral\_value\_in\_usd": 200.0
* }
* **Status Codes**:
  + 200: Success
  + 400: Invalid user address
  + 500: Server error

**12. Collateral Balance (/collateral-balance)**

* **Method**: POST
* **Description**: Gets a user’s collateral balance for a token.
* **Request Body**:
* {
* "user": "0x...",
* "token": "0x... (WAVAX or WETH)"
* }
* **Response**:
* {"balance": 0.5}
* **Status Codes**:
  + 200: Success
  + 400: Invalid addresses
  + 500: Server error

**13. Health Factor (/health-factor)**

* **Method**: POST
* **Description**: Retrieves a user’s health factor.
* **Request Body**:
* {
* "user": "0x..."
* }
* **Response**:
* {"health\_factor": 1.5}
* **Status Codes**:
  + 200: Success
  + 400: Invalid user address
  + 500: Server error

**14. Collateral Tokens (/collateral-tokens)**

* **Method**: GET
* **Description**: Lists supported collateral tokens.
* **Response**:
* {"collateral\_tokens": ["0x... (WAVAX)", "0x... (WETH)"]}
* **Status Codes**:
  + 200: Success
  + 500: Server error

**15. USD Value (/usd-value)**

* **Method**: POST
* **Description**: Gets the USD value of a user’s collateral.
* **Request Body**:
* {
* "user": "0x...",
* "token": "0x... (WAVAX or WETH)",
* "amount": 0.5 // Optional; defaults to user’s collateral balance
* }
* **Response**:
* {"usd\_value": 100.0}
* **Status Codes**:
  + 200: Success
  + 400: Invalid addresses, non-positive amount
  + 500: Server error

**Frontend Integration**

**Example API Call (JavaScript)**

async function depositEthToWeth(amount) {

const response = await fetch('http://localhost:8000/deposit-eth-to-weth', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ amount })

});

const result = await response.json();

if (!response.ok) throw new Error(result.detail);

return result;

}

// Usage

depositEthToWeth(0.1)

.then(result => console.log('Success:', result))

.catch(error => console.error('Error:', error));

**Error Handling**

* **400**: Validate inputs (addresses, amounts) before sending requests.
* **500**: Display transaction revert reasons to users (e.g., “Insufficient allowance”).
* **Timeouts**: Handle long transaction times (default timeout: 120 seconds).

**UI Considerations**

* **Form Validation**: Ensure addresses are valid Ethereum addresses (use ethers.js or web3.js for validation).
* **Loading States**: Show spinners for transaction endpoints, as they may take time.
* **Balance Display**: Use /account-information, /collateral-balance, and /usd-value to show user balances and health factor.

**AI Engineer Notes**

**Contract Interactions**

* **Web3 Setup**:
  + Uses web3.py with Web3.HTTPProvider for Sepolia.
  + Instantiates w3 before contract calls to avoid AttributeError.
  + Example fix for Avalanche:
  + w3 = Web3(Web3.HTTPProvider(AVAX\_RPC\_URL))
  + account = w3.eth.account.from\_key(PRIVATE\_KEY)
* **Transaction Building (build\_tx)**:
  + Handles both direct transfers (e.g., ETH to WETH) and contract calls.
  + Uses dynamic gas pricing (w3.eth.gas\_price) and 100,000 gas for transfers, 3,000,000 for contract calls.
  + Waits for transaction receipt (120-second timeout) and checks for reverts.
  + Example usage:
  + tx\_hash, receipt = build\_tx(
  + to\_address=weth\_address,
  + value=w3.to\_wei(0.1, "ether"),
  + data="0x"
  + )
* **Event Parsing**:
  + Parses Deposit, Transfer, and BurnedAndWithdrawn events for WETH operations.
  + Example for /deposit-eth-to-weth:
  + deposit\_logs = weth\_contract.events.Deposit().process\_receipt(receipt, errors=DISCARD)
* **ABI**:
  + DSCEngine.json is loaded from a Foundry output file.
  + WETH/WAVAX ABI includes depositEthAndMintWETH, withdrawEthAndBurnWETH, and events.
  + DSC ABI is minimal (approve, balanceOf, allowance).