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

**Overview**

The avalanche.py script is a FastAPI application that provides a RESTful API for interacting with the DSCEngine smart contract on the Avalanche Fuji testnet (chain ID 43113). The contract manages a decentralized stablecoin (DSC) protocol, allowing users to deposit collateral (WAVAX or WETH), mint DSC, redeem collateral, burn DSC, liquidate undercollateralized positions, and receive DSC from Sepolia (chain ID 11155111) via the DSCTransferHub contract. 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 cross-chain operations), 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

* **Avalanche Fuji Node**: Access to a Fuji RPC endpoint (e.g., https://api.avax-test.network/ext/bc/C/rpc).
* **Environment Variables**: Create a .env file in the project root:
* AVALANCHE\_RPC\_URL=https://api.avax-test.network/ext/bc/C/rpc
* PRIVATE\_KEY=0xYOUR\_PRIVATE\_KEY
* AVALANCHE\_DSCEngine\_ADDRESS=0x...
* AVALANCHE\_WAVAX\_ADDRESS=0x...
* AVALANCHE\_WETH\_ADDRESS=0x...
* AVALANCHE\_DSC\_ADDRESS=0x...
* AVALANCHE\_DSC\_TRANSFER\_HUB\_ADDRESS=0x...
  + AVALANCHE\_RPC\_URL: Fuji node endpoint.
  + PRIVATE\_KEY: Avalanche private key (with 0x prefix) for signing transactions.
  + AVALANCHE\_DSCEngine\_ADDRESS: DSCEngine contract address on Fuji.
  + AVALANCHE\_WAVAX\_ADDRESS: WAVAX token contract address (wrapped AVAX).
  + AVALANCHE\_WETH\_ADDRESS: WETH token contract address (wrapped ETH).
  + AVALANCHE\_DSC\_ADDRESS: DSC token contract address.
  + AVALANCHE\_DSC\_TRANSFER\_HUB\_ADDRESS: DSCTransferHub contract address for cross-chain DSC transfers.

**Running the API**

1. Save the script as avalanche.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 avalanche:app --host 0.0.0.0 --port 8000 --reload
3. Access the API at http://localhost:8000. Use /docs for the Swagger UI.

**Avoiding Common Errors**

* **AttributeError: type object 'Web3' has no attribute 'eth'**:
  + Ensure no local file named web3.py or Web3.py exists in the project directory:
  + ls | grep -i web3
  + mv web3.py web3\_backup.py # Rename if found
  + Verify web3.py version:
  + pip install web3>=6.0.0
  + Use w3.eth after instantiating w3 = Web3(Web3.HTTPProvider(...)).
* **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
* **Nonce Too Low**:
  + The build\_tx function retries up to 3 times for "nonce too low" errors, common in Avalanche due to network congestion.

**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 or DSCTransferHub 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.
  + Use AVALANCHE\_DSC\_TRANSFER\_HUB\_ADDRESS as spender\_address for /receive-dsc.

**2. Deposit AVAX to WAVAX (/deposit-avax-to-wavax)**

* **Method**: POST
* **Description**: Sends AVAX to the WAVAX contract to mint WAVAX, triggering the receive() function.
* **Request Body**:
* {
* "amount": 0.1
* }
* **Response**:
* {
* "tx\_hash": "0x...",
* "avax\_balance\_before": 1.5,
* "avax\_balance\_after": 1.39,
* "wavax\_balance\_before": 0.0,
* "wavax\_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 AVAX balance, invalid amount
  + 500: Transaction revert or server error
* **Notes**:
  + Checks AVAX balance before sending.
  + Parses Deposit and Transfer events to verify WAVAX minting.
  + Uses gas limit of 100,000 and doubled gas price (w3.eth.gas\_price \* 2).

**3. Withdraw AVAX and Burn WAVAX (/withdraw-avax-and-burn-wavax)**

* **Method**: POST
* **Description**: Burns WAVAX to withdraw AVAX by calling withdrawEthAndBurnWETH.
* **Request Body**:
* {
* "amount": 0.1
* }
* **Response**:
* {
* "tx\_hash": "0x...",
* "avax\_balance\_before": 1.39,
* "avax\_balance\_after": 1.49,
* "wavax\_balance\_before": 0.1,
* "wavax\_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 WAVAX or AVAX (for gas), insufficient contract AVAX
  + 500: Transaction revert or server error
* **Notes**:
  + Checks WAVAX balance and contract AVAX balance.
  + Parses BurnedAndWithdrawn and Transfer events.
  + Uses gas limit of 300,000 and doubled gas price.

**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. Receive DSC (/receive-dsc)**

* **Method**: POST
* **Description**: Receives DSC from Sepolia (chain ID 11155111) via the DSCTransferHub contract.
* **Request Body**:
* {
* "amount": 50.0,
* "source\_chain\_id": 11155111
* }
* **Response**:
* {"tx\_hash": "0x..."}
* **Status Codes**:
  + 200: Success
  + 400: Invalid amount, incorrect source chain ID, insufficient allowance
  + 500: Transaction revert or server error
* **Notes**:
  + Requires prior approval for DSC to AVALANCHE\_DSC\_TRANSFER\_HUB\_ADDRESS.
  + Validates source\_chain\_id as 11155111 (Sepolia).

**12. 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

**13. 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

**14. 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

**15. 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

**16. 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 depositAvaxToWavax(amount) {

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

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

depositAvaxToWavax(0.1)

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

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

**Error Handling**

* **400**: Validate inputs (addresses, amounts, source\_chain\_id) before sending requests.
* **500**: Display transaction revert reasons to users (e.g., “Insufficient allowance”).
* **Timeouts**: Handle long transaction times (timeout: 300 seconds for Fuji).

**UI Considerations**

* **Form Validation**: Ensure addresses are valid (use ethers.js or web3.js for validation).
* **Loading States**: Show spinners for transaction endpoints, as Fuji transactions may take longer.
* **Balance Display**: Use /account-information, /collateral-balance, and /usd-value to show user balances and health factor.
* **Cross-Chain UI**: Display a confirmation for /receive-dsc with Sepolia chain ID (11155111).

**AI Engineer Notes**

**Contract Interactions**

* **Web3 Setup**:
  + Uses web3.py with Web3.HTTPProvider for Avalanche Fuji.
  + Instantiates w3 before contract calls to avoid AttributeError:
  + w3 = Web3(Web3.HTTPProvider(AVALANCHE\_RPC\_URL))
  + account = w3.eth.account.from\_key(PRIVATE\_KEY)
* **Transaction Building (build\_tx)**:
  + Handles direct transfers (e.g., AVAX to WAVAX) and contract calls.
  + Uses doubled gas price (w3.eth.gas\_price \* 2) and 100,000 gas for transfers, 3,000,000 for contract calls.
  + Retries up to 3 times for "nonce too low" errors.
  + Waits for transaction receipt (300-second timeout) and checks for reverts.
  + Example usage:
  + tx\_hash, receipt = build\_tx(
  + to\_address=wavax\_address,
  + value=w3.to\_wei(0.1, "ether"),
  + data="0x"
  + )
* **Event Parsing**:
  + Parses Deposit, Transfer, and BurnedAndWithdrawn events for WAVAX operations.
  + Example for /deposit-avax-to-wavax:
  + deposit\_logs = wavax\_contract.events.Deposit().process\_receipt(receipt, errors=DISCARD)
* **ABI**:
  + DSCEngine.json is loaded from a Foundry output file.
  + WAVAX/WETH ABI includes depositEthAndMintWAVAX, withdrawEthAndBurnWETH, and events.
  + DSC ABI is minimal (approve, balanceOf, allowance).
  + DSCTransferHub ABI includes receiveDsc for cross-chain transfers.

**Cross-Chain Considerations**

* **Receiving DSC**:
  + The /receive-dsc endpoint calls receiveDsc on the DSCTransferHub contract, expecting source\_chain\_id as 11155111 (Sepolia).
  + Requires DSC approval to AVALANCHE\_DSC\_TRANSFER\_HUB\_ADDRESS.
  + Example cross-chain flow:
    1. Approve DSC on Sepolia (evm.py /approve-tokens).
    2. Send DSC via Sepolia’s transfer hub (not implemented in evm.py).
    3. Call /receive-dsc on Fuji to receive DSC.
* **Chainlink CCIP**:
  + Assumes DSCTransferHub uses Chainlink CCIP for cross-chain transfers (e.g., destination\_chain\_selector = 16015286601757825753 for Sepolia).
  + Verify CCIP configuration in DSCTransferHub contract.

**Debugging Tips**

* **Transaction Reverts**:
  + Check revert reason using w3.eth.call:
  + try:
  + w3.eth.call(txn)
  + except Exception as e:
  + print(f"Revert reason: {str(e)}")
  + Common issues: Insufficient allowance, low health factor, or invalid source\_chain\_id.
* **Contract Verification**:
  + Verify contract deployment:
  + code = w3.eth.get\_code(contract\_address)
  + assert len(code) > 0, "Contract not deployed"
* **Snowtrace**:
  + Check transactions on https://testnet.snowtrace.io/tx/<tx\_hash>.
* **Gas Issues**:
  + Fuji requires higher gas prices (e.g., ≥25 nAVAX):
  + gas\_price = max(w3.eth.gas\_price \* 2, w3.to\_wei("25", "gwei"))
* **Owner Check**:
  + If DSCEngine or DSCTransferHub has owner restrictions:
  + owner = contract.functions.owner().call()
  + assert owner.lower() == public\_address.lower(), "Not contract owner"

**Testing**

**Setup**

1. Fund the account with Fuji AVAX:
2. open https://faucet.avax.network/
3. Backup the script:
4. cp avalanche.py avalanche.py.bak

**Test Endpoint**

curl -X POST http://localhost:8000/deposit-avax-to-wavax -H 'Content-Type: application/json' -d '{"amount": 0.01}'

Expected response:

{

"tx\_hash": "0x...",

"avax\_balance\_before": 1.0,

"avax\_balance\_after": 0.989,

"wavax\_balance\_before": 0.0,

"wavax\_balance\_after": 0.01,

"events": {...},

"tx\_data": "0x"

}

**Verify on Snowtrace**

open https://testnet.snowtrace.io/tx/0x...

**Check Logs**

tail -f uvicorn.log

**Security Considerations**

* **Private Key**: Store securely in .env, never hardcode.
* **Gas Limits**: Adjust if transactions fail due to gas (e.g., increase to 150,000 for transfers).
* **Rate Limiting**: Add middleware to prevent abuse:
* from fastapi.middleware.cors import CORSMiddleware
* app.add\_middleware(CORSMiddleware, allow\_origins=["\*"], allow\_methods=["\*"], allow\_headers=["\*"])
* **Input Validation**: All endpoints validate addresses, amounts, and source\_chain\_id.
* **Cross-Chain Security**: Ensure DSCTransferHub is audited for CCIP integration.

**Cross-Chain Notes**

* The API supports receiving DSC from Sepolia via /receive-dsc, assuming Chainlink CCIP integration in DSCTransferHub.
* For sending DSC to Sepolia, implement a corresponding endpoint in evm.py (e.g., /send-dsc).
* Verify chain selectors (e.g., Sepolia: 16015286601757825753) in DSCTransferHub contract.

This documentation provides a complete guide for using and extending the DSCEngine API on Avalanche Fuji. For further assistance, share error messages, transaction hashes, or contract details.