Homework 3: Bitcoin Data Ingestion

1. Build and Run Bitcoind Using Docker

1.1 Write a Dockerfile to Build Bitcoind

```
FROM ubuntu:22.04

ENV BITCOIN_VERSION=28.1
ENV BITCOIN_DATA=/data

RUN apt-get update && apt-get install -y \
    curl \
    libssl3 \
    libevent-2.1-7 \
    libzmq5 \
    python3 \
    python3-pip \
    && ln -s /usr/bin/python3 /usr/bin/python \
    && rm -rf /var/lib/apt/lists/*

RUN pip3 install python-bitcoinrpc fastapi[standard]

RUN set -ex \
    && BITCOIN_TAR="bitcoin-${BITCOIN_VERSION}-x86_64-linux-gnu.tar.gz" \
    && curl -fSLO "https://bitcoincore.org/bin/bitcoin-core-${BITCOIN_VERSION}/${BITCOIN_TAR}" \
    && tar -xzf "${BITCOIN_TAR}" -C /usr/local --strip-components=1 \
    && rm "${BITCOIN_TAR}"

RUN mkdir -p "${BITCOIN_DATA}"
COPY bitcoin.conf "${BITCOIN_DATA}/"

EXPOSE 8332 8333
```

1.2 Write a Docker-Compose File

```
version: '3.8'

services:
  bitcoind:
    build: .
    ports:
      - "8333:8333"  # P2P network
      - "8332:8332"  # RPC interface
    volumes:
      - ./bitcoin_data:/data
    command: ["bitcoind", "-datadir=/data"]
```

1.3 Write a bitcon.conf File

```
# bitcoin.conf
server=1
rest=1
txindex=0
prune=550

rpcauth=admin:bfcb3608ae0e91a0d38404fc329049f7$e7e32504e9d0da7c5351cc9474c0abf93952a7a16c5279ea70f95a16ffef99ac
rpcallowip=127.0.0.1
rpcport=8332

listen=1
port=8333
```

1.4 Data Storage Location Instructions

1. Blockchain data will be stored on the host machine in the following paths (relative to the docker-compose.yml location):
   - `./bitcoin_data/blocks/` (block data)
   - `./bitcoin_data/chainstate/` (chain state)
   - `./bitcoin_data/bitcoin.conf` (configuration file)
2. Other important data locations:
   - `./bitcoin_data/wallets/` (wallet files, if wallets are created)
   - `./bitcoin_data/debug.log` (log file)

1.5 Usage Steps:

1. Start the container:

```
docker-compose up -d
```

2. Stop the container (data will be automatically preserved):

```
docker-compose down
```

---

2. Run Full Data Sync on Modal

2.1 Bitcoin Data Sync

Below is the Python code to sync Bitcoin data using Chainstack's RPC endpoint. It stored the bitcoin data into a database that is hosted on Modal Volume.

```python
# bitcoin_explorer.py
import modal
from modal import App, Volume, Secret
import os
import requests
import sqlite3
import json
import time
from typing import Dict, Any

app = App(name="chongchen-bitcoin-explorer")  # Use modal.App

# Define the volume and Docker image
volume = Volume.from_name("chongchen-bitcoin-data", create_if_missing=True)
bitcoin_image = modal.Image.debian_slim().pip_install("requests")

class BitcoinRPC:
    """Handles RPC communication with Bitcoin node via Chainstack"""
    def __init__(self):
        self.rpc_username = os.environ["RPC_USERNAME"]
        self.rpc_password = os.environ["RPC_PASSWORD"]
        self.rpc_host = os.environ["RPC_HOST"]
        self.rpc_port = os.environ["RPC_PORT"]
        self.rpc_path = os.environ["RPC_PATH"]
        self.rpc_endpoint = f"https://{self.rpc_host}:{self.rpc_port}{self.rpc_path}"
        self.auth = (self.rpc_username, self.rpc_password)

    def make_rpc_call(self, method: str, params: list) -> Dict[str, Any]:
        """Execute JSON-RPC call"""
        payload = {
            "jsonrpc": "2.0",
            "method": method,
            "params": params,
            "id": 1
        }
        try:
            response = requests.post(
                self.rpc_endpoint,
                auth=self.auth,
                json=payload,
                headers={'Content-Type': 'application/json'},
                timeout=10
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"RPC Error: {e}")
            raise

    def get_block_count(self) -> int:
        """Fetch current blockchain height"""
        resp = self.make_rpc_call("getblockcount", [])
        return resp["result"]

    def get_block_hash(self, height: int) -> str:
        """Get block hash by height"""
        resp = self.make_rpc_call("getblockhash", [height])
        return resp["result"]

    def get_block(self, block_hash: str) -> Dict:
        """Retrieve block data with transactions"""
        resp = self.make_rpc_call("getblock", [block_hash, 2])
        return resp["result"]

def get_db_connection():
    """Connect to SQLite database in Modal Volume"""
    return sqlite3.connect('/data/bitcoin.db')

def init_db():
    """Initialize database schema if not exists"""
    with get_db_connection() as conn:
        conn.execute("""
            CREATE TABLE IF NOT EXISTS block (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                hash VARCHAR(255) NOT NULL,
                confirmations INTEGER NOT NULL,
                height INTEGER NOT NULL,
                version INTEGER NOT NULL,
                versionHex VARCHAR(255) NOT NULL,
                merkleroot VARCHAR(255) NOT NULL,
                time INTEGER NOT NULL,
                mediantime INTEGER NOT NULL,
                nonce INTEGER NOT NULL,
                bits VARCHAR(255) NOT NULL,
                difficulty REAL NOT NULL,
                chainwork VARCHAR(255) NOT NULL,
                nTx INTEGER NOT NULL,
                previousblockhash VARCHAR(255) NOT NULL,
                nextblockhash VARCHAR(255) NOT NULL,
                strippedsize INTEGER NOT NULL,
                size INTEGER NOT NULL,
                weight INTEGER NOT NULL,
                tx JSON NOT NULL
            );
        """)
        conn.commit()

def get_max_height() -> int:
    """Get the highest block height from the database"""
    with get_db_connection() as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT MAX(height) AS max_height FROM block")
        row = cursor.fetchone()
        return row[0] if row[0] is not None else -1

def save_block(block_data: Dict):
    """Save block to database and Volume"""
    # Insert into SQLite
    with get_db_connection() as conn:
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO block (
                hash, confirmations, height, version, versionHex, merkleroot,
                time, mediantime, nonce, bits, difficulty, chainwork, nTx,
                previousblockhash, nextblockhash, strippedsize, size, weight, tx
            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        """, (
            block_data['hash'],
            block_data.get('confirmations', 0),
            block_data['height'],
            block_data['version'],
            block_data['versionHex'],
            block_data['merkleroot'],
            block_data['time'],
            block_data.get('mediantime', block_data['time']),
            block_data['nonce'],
            block_data['bits'],
            block_data['difficulty'],
            block_data['chainwork'],
            block_data['nTx'],
            block_data.get('previousblockhash', ''),
            block_data.get('nextblockhash', ''),
            block_data['strippedsize'],
            block_data['size'],
            block_data['weight'],
            json.dumps(block_data['tx'])
        ))
        conn.commit()

    # Save JSON to Volume
    # block_dir = "/data/blocks"
    # os.makedirs(block_dir, exist_ok=True)
    # with open(f"{block_dir}/block_{block_data['height']}.json", 'w') as f:
    #     json.dump(block_data, f)

@app.function(
    volumes={"/data": volume},
    image=bitcoin_image,
    secrets=[Secret.from_name("chongchen-bitcoin-chainstack")],
    timeout=86400  # Extend timeout for long syncing
)
def sync_blocks():
    """Main function to sync blocks continuously"""
    init_db()
    rpc = BitcoinRPC()
    while True:
        current_height = rpc.get_block_count()
        max_synced = get_max_height()

        if max_synced >= current_height:
            print("All blocks synced. Sleeping for 10 minutes.")
            time.sleep(600)
            continue

        print(f"Syncing blocks {max_synced + 1} to {current_height}")
        for height in range(max_synced + 1, current_height + 1):
            try:
                block_hash = rpc.get_block_hash(height)
                block_data = rpc.get_block(block_hash)
                save_block(block_data)
                print(f"Block {height} synced")
            except Exception as e:
                print(f"Failed to sync block {height}: {e}")
                break  # Retry from current height on next iteration

if __name__ == "__main__":
    with app.run():
        sync_blocks.call()
```

The database that on Modal Volume can access with: https://modal.com/api/volumes/neu-info5100-oak-spr-2025/main/chongchen-bitcoin-data/files/content?path=bitcoin.db

2.2 Run the Modal Function in Detached Mode

```
modal run --detach bitcoin_explorer.py
```

---

3. Monitor Sync Progress

3.1 Check Data Growth in Modal Volume

```
 modal volume ls
modal volume ls chongchen-bitcoin-data
```

3.2 Check Sync Progress via Chainstack RPC Call

```
 # chainstack_rpc.py
import requests
import json
from datetime import datetime
import os
from dotenv import load_dotenv

def get_block_verbose(block_hash, endpoint, username, password):
    """
    Make a getblock RPC call with verbosity=2

    Args:
        block_hash (str): The hash of the block to retrieve
        endpoint (str): The Chainstack endpoint URL
        username (str): Chainstack username
        password (str): Chainstack password
    """

    payload = {
        "jsonrpc": "2.0",
        "method": "getblock",
        "params": [block_hash, 2],
        "id": 1
    }

    try:
        response = requests.post(
            endpoint,
            auth=(username, password),
            json=payload,
            headers={'Content-Type': 'application/json'}
        )

        response.raise_for_status()
        return response.json()

    except requests.exceptions.RequestException as e:
        print(f"Error making RPC call: {e}")
        return None

def save_block_data(block_data, output_dir="block_data"):
    """
    Save block data to a JSON file
    """
    # Create output directory if it doesn't exist
    os.makedirs(output_dir, exist_ok=True)

    # Get block height or hash for filename
    block_info = block_data.get('result', {})
    block_height = block_info.get('height', 'unknown')
    block_hash = block_info.get('hash', 'unknown')

    # Create filename with timestamp
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"block_{block_height}_{block_hash[:8]}_{timestamp}.json"
    filepath = os.path.join(output_dir, filename)

    # Save to file with pretty printing
    with open(filepath, 'w') as f:
        json.dump(block_data, f, indent=2)

    return filepath

def main():
    # Load environment variables from .env file
    load_dotenv()

    # Get Chainstack credentials from environment variables
    ENDPOINT = os.getenv("chainstack_https_endpoint")
    USERNAME = os.getenv("chainstack_username")
    PASSWORD = os.getenv("chainstack_password")

    # Check if environment variables are loaded properly
    if not all([ENDPOINT, USERNAME, PASSWORD]):
        print("Error: Missing environment variables. Make sure .env file exists with required variables.")
        return

    # Example block hash (Bitcoin genesis block)
    BLOCK_HASH = "000000000000000000003fb9a79c6b9c73831537eb31b469ad113d6a99176a97"

    # Make the call
    print("Fetching block data...")
    result = get_block_verbose(BLOCK_HASH, ENDPOINT, USERNAME, PASSWORD)

    if result and 'result' in result:
        # Save to file
        filepath = save_block_data(result)
        print(f"Block data saved to: {filepath}")
    elif result:
        print("Error in response:", result.get('error'))
    else:
        print("Failed to get response")

if __name__ == "__main__":
    main()
```

Using the following command to run the script:

```
python chainstack_rpc.py
```

3.3 Check Sync Status via Modal Database

```
# query_modal_db.py
import sqlite3
from modal import App, Volume

app = App("chongchen-bitcoin-rpc")
volume = Volume.from_name("chongchen-bitcoin-data")

@app.function(volumes={"/data": volume})
def query_bitcoin_db():
    """Query the Bitcoin blockchain database stored in Modal Volume."""
    conn = sqlite3.connect("/data/bitcoin.db")
    cursor = conn.cursor()

    # Example 1: Get the latest block information
    cursor.execute("SELECT * FROM block ORDER BY height DESC LIMIT 1;")
    latest_block = cursor.fetchone()
    print("Latest Block:", latest_block)

    # Example 2: Count total number of blocks in the database
    cursor.execute("SELECT COUNT(*) FROM block;")
    total_blocks = cursor.fetchone()[0]
    print("Total Blocks:", total_blocks)

    # Example 3: Get the block with the highest difficulty
    cursor.execute("SELECT * FROM block ORDER BY difficulty DESC LIMIT 1;")
    highest_difficulty_block = cursor.fetchone()
    print("Block with Highest Difficulty:", highest_difficulty_block)

    conn.close()

if __name__ == "__main__":
    query_bitcoin_db.call()
```

The above code shows how to query the SQLite database stored in a Modal Volume. You can run this function using:

```
modal run query_modal_db.py
```

---

4. Implementation Details

Key Features

1. **Full Blockchain Sync**
    - Uses `BitcoinRPC` class to interact with a remote `bitcoind` node via **JSON-RPC**.
    - Fetches block data using `getblockcount`, `getblockhash`, and `getblock` RPC calls.

- Stores blockchain data in SQLite within a **Modal Volume** (`chongchen-bitcoin-data`).
- Periodically checks for new blocks and syncs them.

2. **Persistence Using Modal Volume**

   - Uses Modal Volumes to store data (`/data/bitcoin.db`).
   - Sync resumes from the last stored block height.

3. **Security & Remote Access**

   - Authentication is enforced via environment secrets (`chongchen-bitcoin-chainstack`).
   - JSON-RPC communication is secured with **HTTPS** and **username/password authentication**.
   - RPC calls from a local machine to the `bitcoind` server on Modal are secured using Modal **webhooks** or **tunnels**.

4. **Monitoring Sync Progress**

   - The sync process runs in **detached mode** (`modal run --detach`).
   - Progress verification methods:
     - Using **Modal Volume CLI** (`modal volume ls`)
     - Running `getblockchaininfo` and `getblockcount` RPC calls via chainstack.
     - Running `query_modal_db.py` to query the database for latest block and total blocks count.

Usage Instructions

Running the Sync

Execute the following command to start the sync process in detached mode:

```
modal run --detach bitcoin_explorer.py
```

Checking Sync Progress

Method 1: Inspect Modal Volume

```
modal volume ls chongchen-bitcoin-data
```

Method 2: Run Chainstack RPC Calls

A `getblock` function can be added to invoke chainstack RPC locally:

```
def get_block_verbose(block_hash, endpoint, username, password):
    """
    Make a getblock RPC call with verbosity=2

    Args:
        block_hash (str): The hash of the block to retrieve
        endpoint (str): The Chainstack endpoint URL
        username (str): Chainstack username
        password (str): Chainstack password
    """

    payload = {
        "jsonrpc": "2.0",
        "method": "getblock",
        "params": [block_hash, 2],
        "id": 1
    }

    try:
        response = requests.post(
            endpoint,
            auth=(username, password),
            json=payload,
            headers={'Content-Type': 'application/json'}
        )

        response.raise_for_status()
        return response.json()

    except requests.exceptions.RequestException as e:
        print(f"Error making RPC call: {e}")
        return None
```

Method 3: Query Modal Database

```
modal run query_modal_db.py
```

Expected Sync Duration

- After the initial sync, new blocks are added in real-time.

---

5. Sources and Links

All codes: https://github.com/chongchen1999/INFO7500-cryptocurrency/tree/main/hw3

Bitcoin database on Modal Volume: https://modal.com/api/volumes/neu-info5100-oak-spr-2025/main/chongchen-bitcoin-data/files/content?path=bitcoin.db

---