

AWS EDUSLG Delivery Data Team - Intern - 2022

AWS Managed BlockChain Implementation

Interns Name:

1. Michael Wong
2. Elizabeth Kelly

Overview

The purpose of this documentation is to provide an outlined process that encompasses all of the necessary steps to recreate what Michael and Elizabeth created during the summer of 2022. This documentation is divided into different sections to outline two phases of the project: 'Learning and Code Development' and 'Data Ingestion Pipeline'. Each section includes architecture diagrams and notes for recommended ways to change the project.

Code Commit Links:

https : <https://git-codecommit.us-east-1.amazonaws.com/v1/repos/blockchain-project>

https (grc) : codecommit::us-east-1://blockchain-project

Materials

AWS SERVICES USED

- Amazon Managed Blockchain (Ethereum)
- Athena
- Cloud9
- CodeCommit
- Glue
- Identity Access Management
- Key Management Service
- Quicksight
- Secrets Manager
- Simple Storage Service

APIS, PACKAGES, DEPENDENCIES

- Boto3 - <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>
- Etherscan API - <https://docs.etherscan.io/>
- Matplotlib - <https://matplotlib.org/>
- Moralis REST API - <https://docs.moralis.io/moralis-dapp/web3-api/moralis-web3-api-rest>
- OpenSea - <https://docs.opensea.io/reference/api-overview>

- Python 3.7 -
- Python Requests - <https://requests.readthedocs.io/en/latest/>
- Web3 py - <https://web3py.readthedocs.io/en/stable/#>

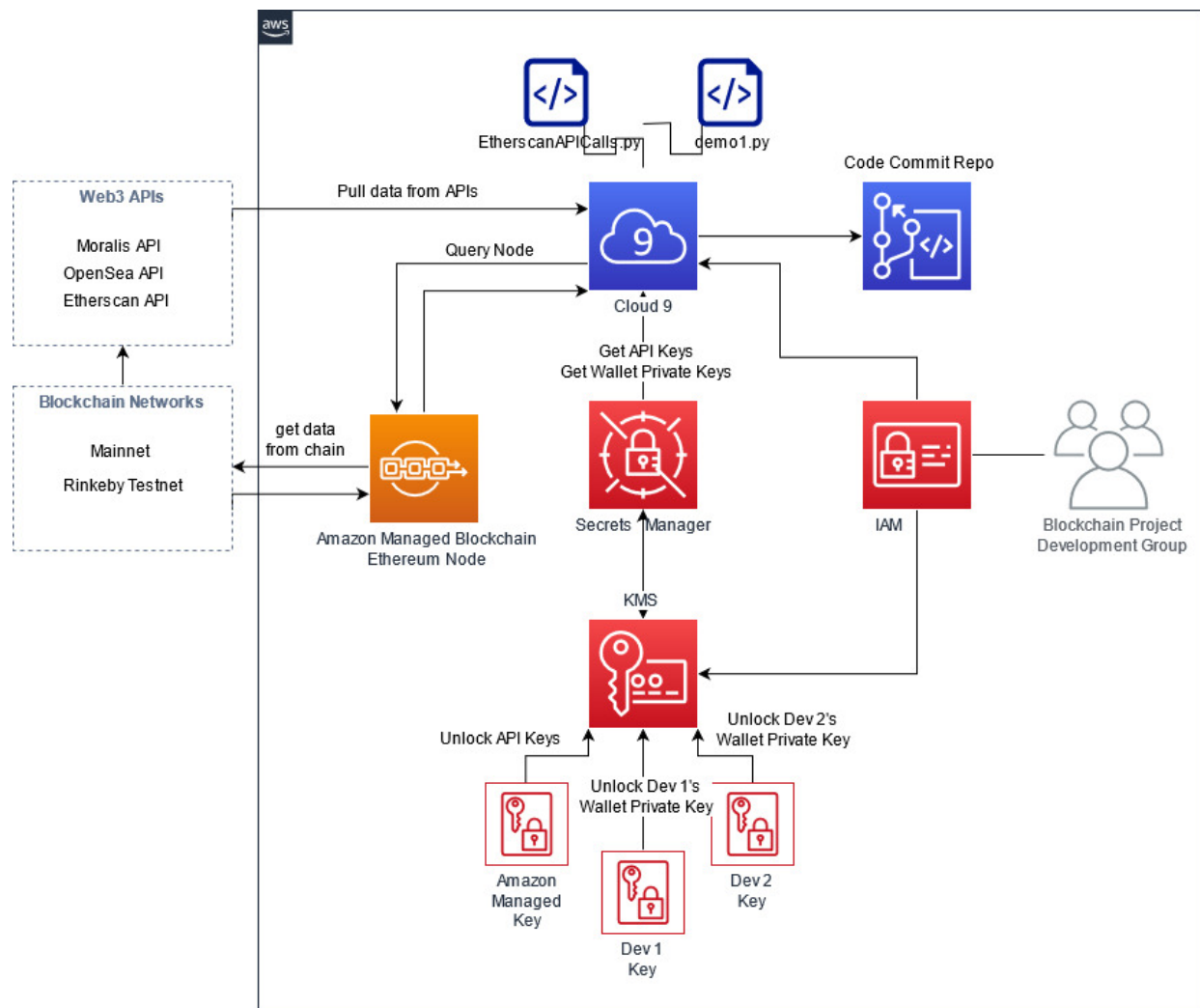
IAM GROUPS, USERS, ROLES, AND PERMISSIONS

OTHER NEEDS

- Meta Mask Browser Extension
- Rinkeby Wallet
- Ethereum
- Rinkeby Etherscan Website - <https://rinkeby.etherscan.io/>
- Mainnet Etherscan Website - <https://etherscan.io/>
- OpenSea NFT Rankings - <https://opensea.io/rankings>

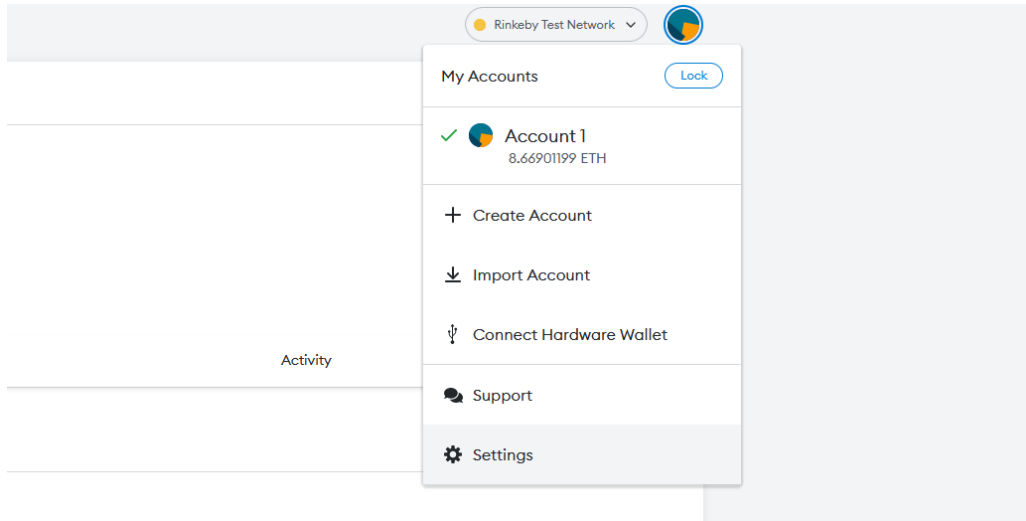
Stages

LEARNING AND CODE DEVELOPMENT

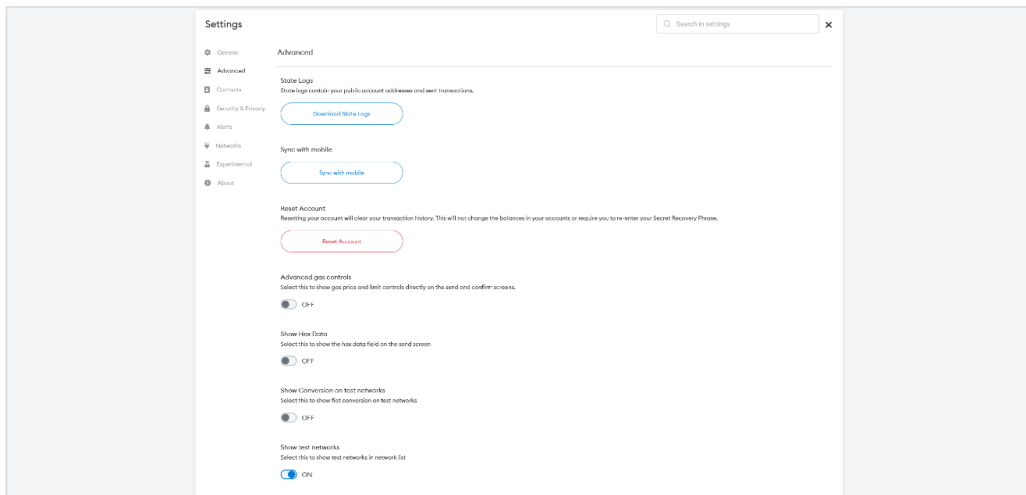


Getting Test Wallets and Ethereum

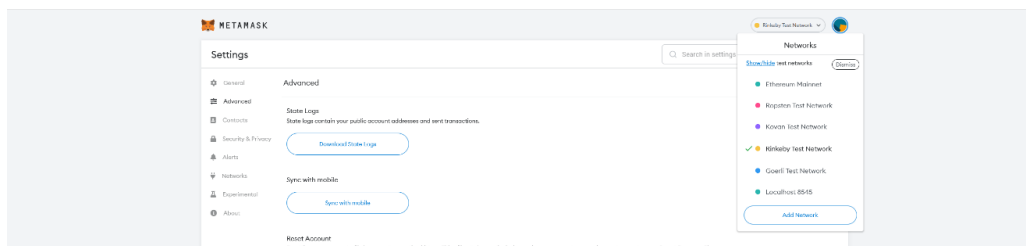
- For creating your test wallet, go to metamask.io
 - Install the plugin for your browser.
 - Follow the instructions for your wallet password.
 - Once you have created your account, click on your profile photo on the upper right hand side and navigate to settings



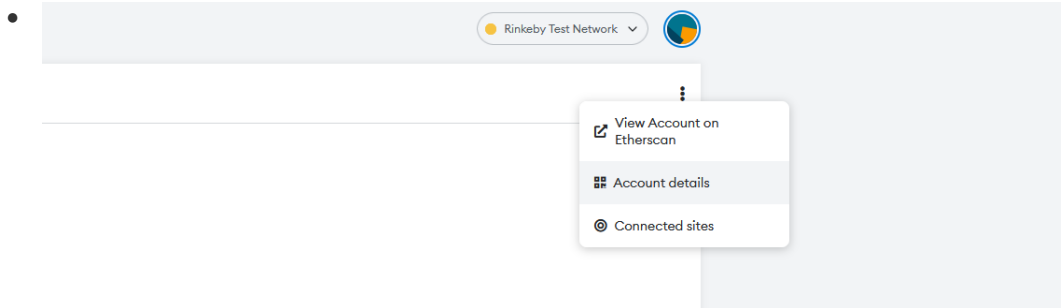
- Go to the "Advanced" section and scroll down to show test networks and click "on".



- Navigate to the upper right hand side and select "Rinkeby Test Network"



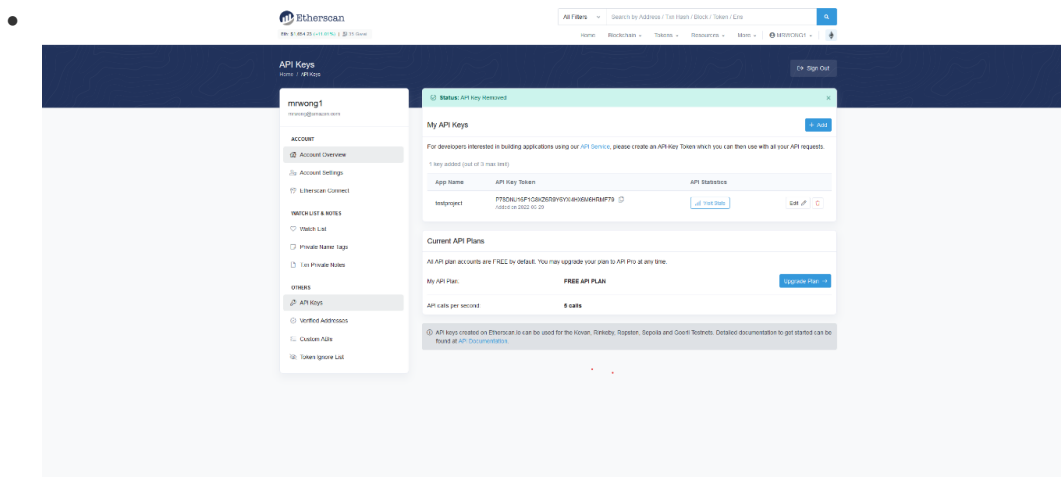
- To view your wallet address, click on the three dots and select "account details"



- Here, you can see your wallet address and also export your wallet's private key.

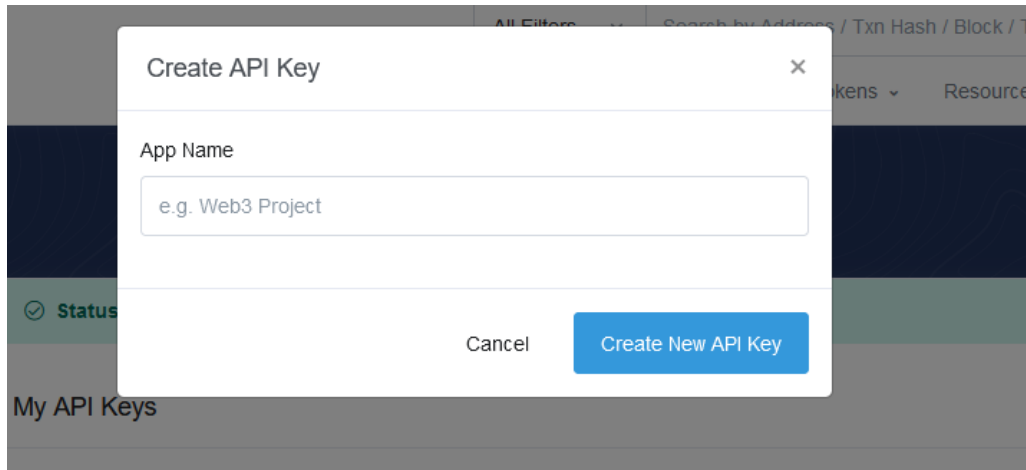
Setting Up Accounts for APIs

- Let's start off by retrieving our etherscan API key.
- Navigate to etherscan.io
- Follow the instructions for creating your account.
- Navigate to your profile and click "my profile."
- Scroll down on the left side and select "API Keys"

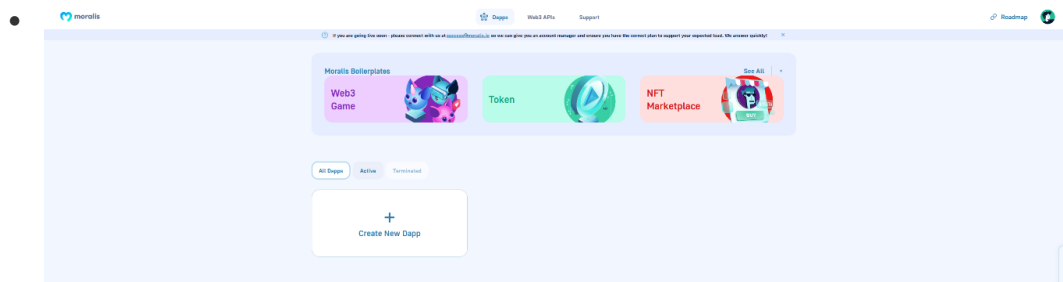


- Now create your API key by clicking "add"

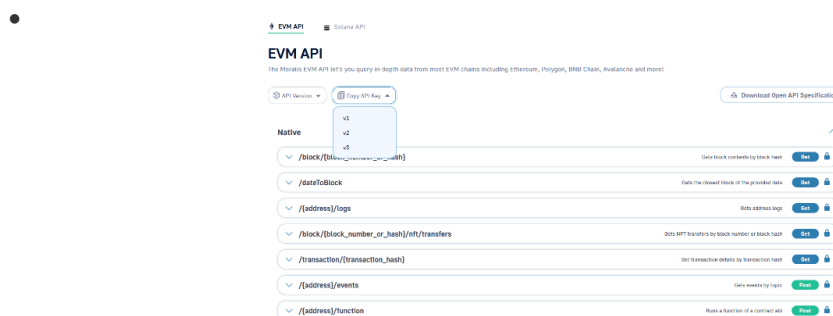
-



- Now, let's get the Moralis API Key.
- Navigate to <https://admin.moralis.io/login>
- Follow the instructions for creating your account.
- Navigate to your "Web 3 APIs" at the top.



- Click on the dropdown "Copy API Key" and select v3.

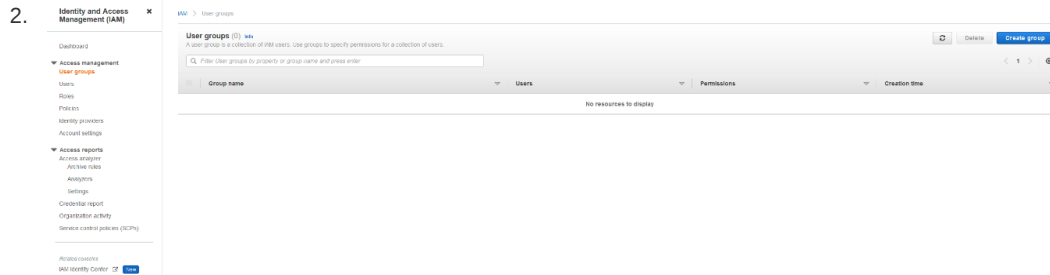


Development Environment Set Up

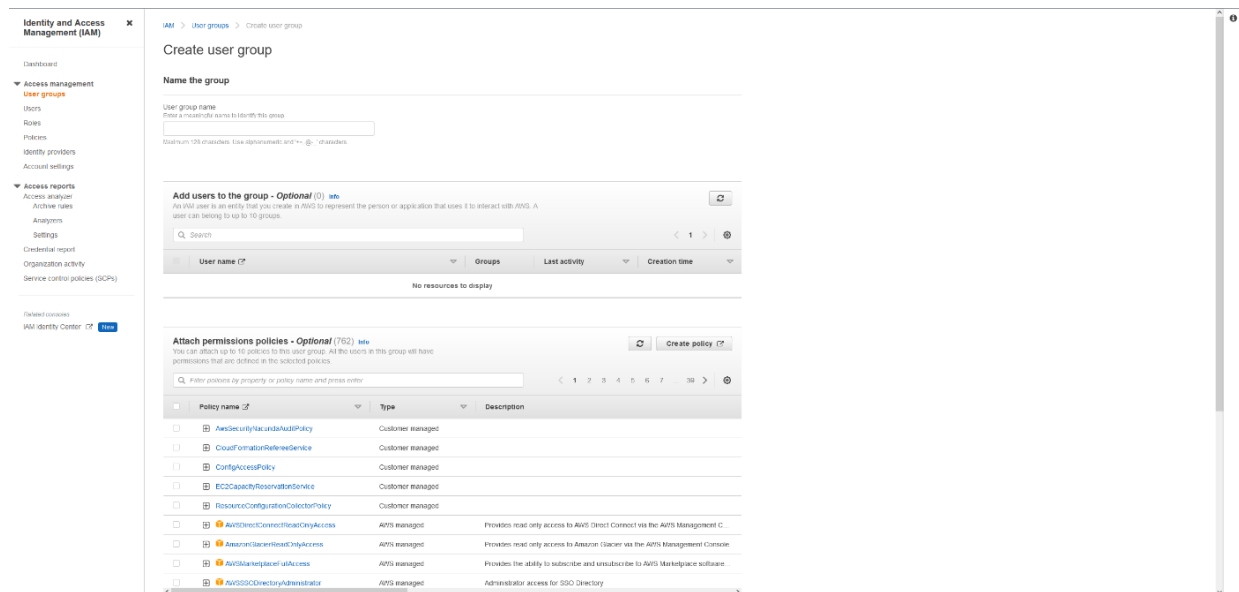
Creating Developer IAM Group

First, we will have to make a developer group that can access the services they need while following the principle of least privilege.

1. First, in your admin account, navigate to the IAM Console, and click 'User Groups' on the side menu. Click 'Create Group'.



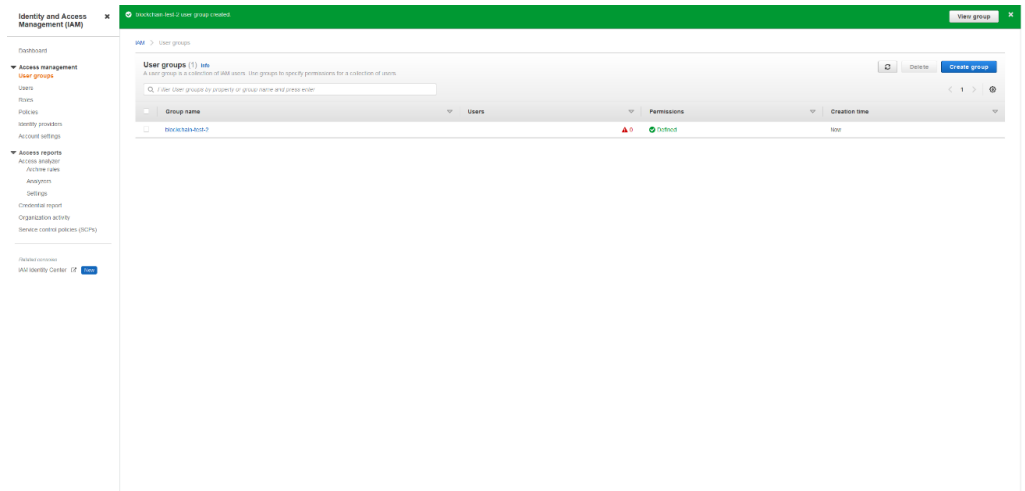
3. Skip add users for now, and attach the following permissions policies:



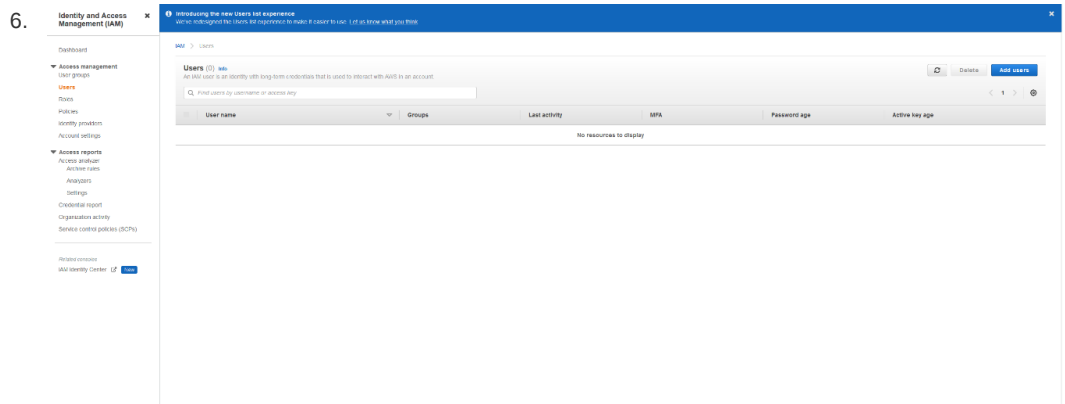
- AWScloud9User
- AmazonManagedBlockchainFullAccess
- AWSCodeCommitFullAccess
- SecretsManagerReadWrite
- AWSKeyManagementServicePowerUser
- AmazonS3FullAccess

3. Click create group.

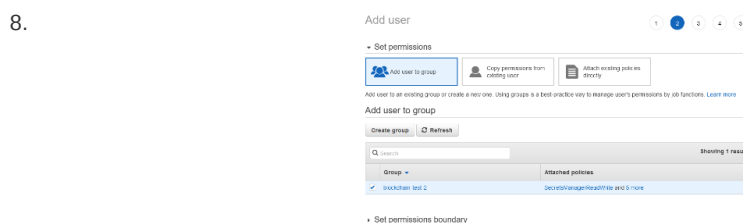
- 4.



- Now create developer users. Navigate to users on the sidebar menu. Click add users. Under Select AWS access type, make sure both Access key - Programmatic access and Password - AWS Management Console access are both selected. Select Password settings as you would prefer. Click Next.



- Under Set Permissions, select Add user to group, and select the group that you created from earlier steps. Click Next.

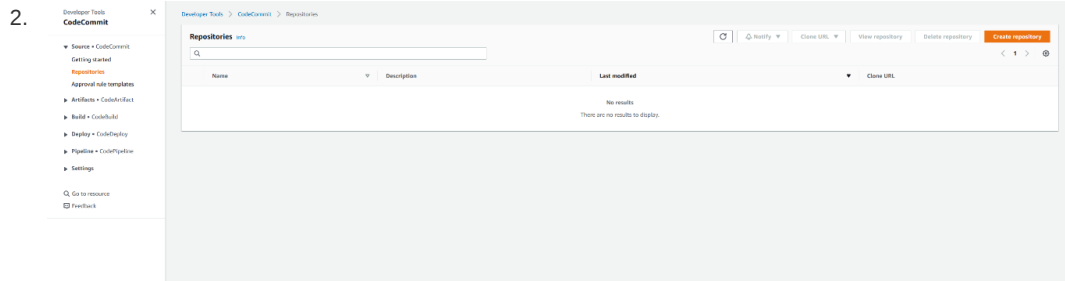


- Add tags as you would prefer. Suggested for keeping track of project resources. Click Next.
- Click Create User, and click download .csv to download your developer account credentials.
- You should now have access the management console through your developer account.

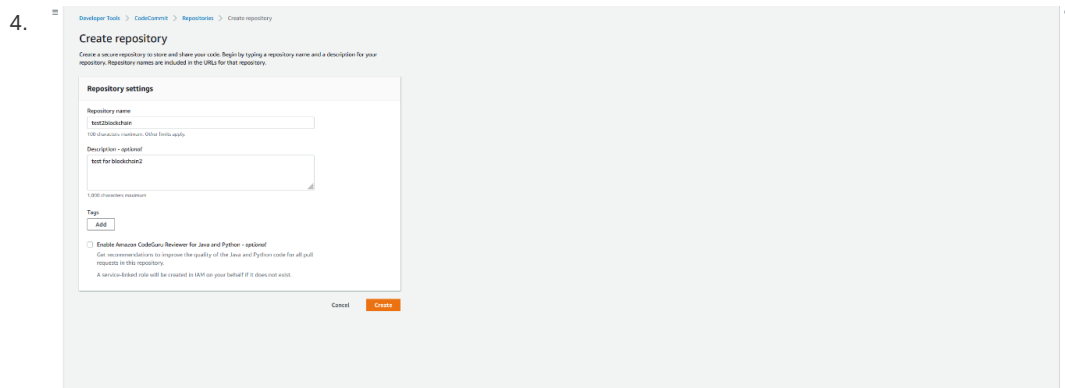
Creating Code Commit Repository

Now we should have a place to store code outside of the cloud 9 enviroment we will create later.

1. Navigate to the CodeCommit Console page.



3. Click Create Repository, and fill in the fields however you like.



5. Create a file called requirements.txt, and copy the following code block into the file.

```
aihttp==3.8.1
aiosignal==1.2.0
astroid==2.3.0
async-timeout==4.0.2
async-test==0.13.0
```



```
attrs==21.4.0
aws-cfn-bootstrap==2.0
backcall==0.2.0
base58==2.1.1
bitarray==2.5.1
boto3==1.24.17
botocore==1.27.17
certifi==2022.6.15
charset-normalizer==2.0.12
cyclers==0.11.0
cytoolz==0.11.2
decorator==5.1.1
Django==2.0.2
docutils==0.14
eth-abi==2.1.1
eth-account==0.5.8
eth-hash==0.3.2
eth-keyfile==0.5.1
eth-keys==0.3.4
eth-rlp==0.3.0
eth-typing==2.3.0
eth-utils==1.10.0
fonttools==4.33.3
frozenlist==1.3.0
git-remote-codecommit==1.16
hexbytes==0.2.2
idna==3.3
ikp3db==1.4.1
importlib-metadata==4.11.4
importlib-resources==5.8.0
ipfshttpclient==0.8.0a2
ipython==7.34.0
isort==4.3.21
jedi==0.18.1
jmespath==1.0.0
jsonschema==4.6.0
kiwisolver==1.4.3
lazy-object-proxy==1.7.1
lockfile==0.11.0
lru-dict==1.1.7
matplotlib==3.5.2
matplotlib-inline==0.1.3
mccabe==0.6.1
multiaddr==0.0.9
multidict==6.0.2
netaddr==0.8.0
numpy==1.21.6
packaging==21.3
pandas==1.3.5
parsimonious==0.8.1
parso==0.8.3
pbr==5.9.0
pexpect==4.8.0
pickleshare==0.7.5
Pillow==9.1.1
```

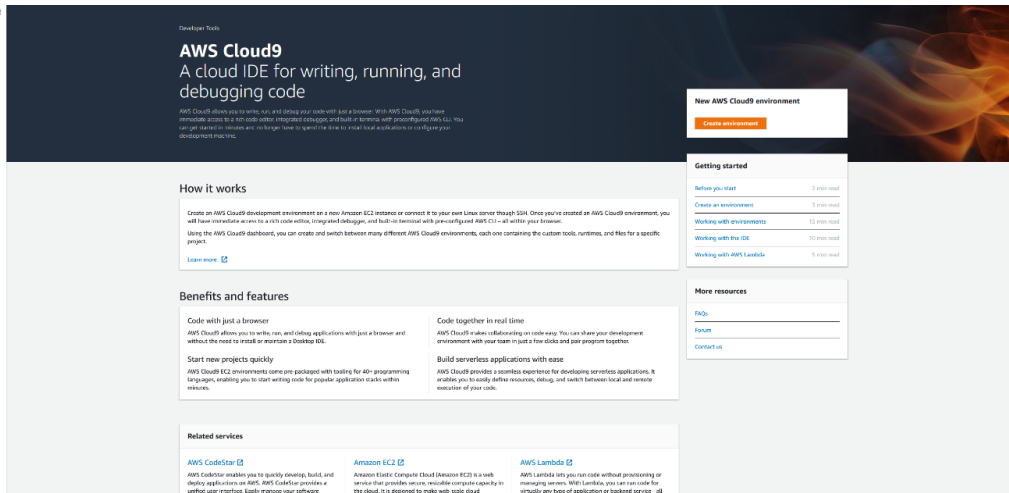
```
prompt-toolkit==3.0.29
protobuf==3.20.1
ptyprocess==0.7.0
pycryptodome==3.14.1
Pygments==2.12.0
pylint==2.4.4
pylint-django==2.3.0
pylint-flask==0.6
pylint-plugin-utils==0.7
pyparsing==3.0.9
pysistent==0.18.1
pystache==0.5.4
python-daemon==2.2.3
python-dateutil==2.8.2
pytz==2022.1
requests==2.28.0
requests-auth-aws-sigv4==0.7
rlp==2.0.1
s3transfer==0.6.0
simplejson==3.2.0
six==1.16.0
stevedore==3.5.0
toolz==0.11.2
traitlets==5.2.2.post1
typed-ast==1.5.4
typing-extensions==4.2.0
urllib3==1.26.9
varint==1.0.2
virtualenv==16.2.0
virtualenv-clone==0.5.7
virtualenvwrapper==4.8.4
wcwidth==0.2.5
web3==5.29.2
websockets==9.1
wrapt==1.14.1
yarl==1.7.2
zipp==3.8.0
```

This will allow you to install code dependencies later.

Creating Cloud 9 Development Environment

Now we need to create an environment to work in.

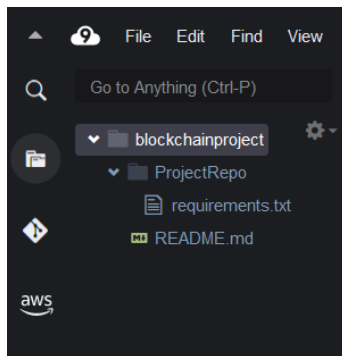
1. Navigate to the cloud9 console page. Click create environment, and fill out name and description to your liking. Click next step.
- 2.



3. Leave default settings for everything except for instance type. Depending on how many users will access your environment, you may have to upgrade to t3.small or m5.large. Click next, then create environment.

1. On the left side menu, click the second to last button (if you hover over the button, it should say 'sourcecontrol'). Click 'Clone Repository'.
2. Navigate to codecommit and under the 'clone url' column in your repo listings, click the HTTPS link for your repo. Paste the link where prompted.

Now when you click on files, your tree should look similar to below this:



Now, we will install dependencies.

1. In the bash terminal, your current directory be your environment. Navigate to your git project folder.

```
cd your_repo_name
```

2. Now, download the project dependencies by running the following command:

```
pip install -r requirements.txt
```

(Optional) Add other users to your environment.

1. Click share, and under 'Invite Members', type in the IAM Usernames of other developers involved in your project.

Storing Secrets in Secrets Manager with KMS

Later, you will need to programmatically access API keys and wallet keys to make API requests and make transactions. This section will cover how to store these keys for general account access for API keys, and individual access to wallet private keys using KMS.

Before doing this, make sure you have made accounts with Etherscan and Moralis.

Storing API Keys

1. Navigate to Secrets Manager and click 'secrets', then 'Store a new secret'.
2. For secret type, select 'Other type of secret'.
3. The key should be 'APIKey', and the value will be your actual API key.
4. Leave Encryption Key as aws/secretsmanager. This way, every user that has access to secrets manager (ie your developer group), can access the API key.
5. For secret name, create a name that lets you know what website or service you are using. (ex: EtherscanAPIKey). Leave every other setting at their default. Click 'Next'.
6. Do not rotate secret, click next. Click 'Store'.

Storing Wallet Keys

1. Navigate to the KMS console page, and click 'create a key'.
2. On the 'Configure Key' page, leave everything at default settings.
3. Name the key however you like. Click 'Next'.
4. For key administrators, select your own dev user only. Only select admin role if you trust all admins in your account. This KMS key will be used in order to unlock your private wallet address. Click 'Next'.
5. For Define key Usage permissions, only select your dev user. Click 'Next'.
6. Click 'Finish'.
7. Navigate to Secrets Manager and click 'Store a New Secret'
8. For Secret Type, click 'Other type of secret'.
9. For the key, type 'WalletPrivateKey'. For the value, put in your wallet account's secret key.
10. Under 'Encryption key', select the key that was made from steps 1 - 6. Click 'Next'.
11. Name the secret as you like. Ex: WalletPrivateKey/UserName
12. Leave everything at default settings, and click 'Next'.
13. Do not rotate secret, click next. Click 'Store'.

Creating an S3 Bucket

1. Navigate to Amazon S3 to create a bucket.
2. Click "Create Bucket" and name your bucket.
3. Leave all the default settings and click create.
4. In the bucket, create a folder and name it data or however you wish. In this project, we named ours "test."

The following folder structure is recommended for upload files:

- Data
 - Tokens

- bored_ape_yacht_club_tokens.csv
- cryptopunks_tokens.csv
-
- Current Wallet Balances
 - 0x48c04ed5691981C42154C6167398f95e8f38a7fF_wallet_balance.csv
 - 0x77C5C3487B5fD4267a618fA4F58395e8EC803C02_wallet_balance.csv
 -
- Wallet Balances Over Time
 - 0x48c04ed5691981C42154C6167398f95e8f38a7fF_wallet_balance_over_time.csv
 - 0x77C5C3487B5fD4267a618fA4F58395e8EC803C02_wallet_balance_over_time.csv
 -

Creating an Amazon Managed Blockchain Node

Later, you will also use web3 py to try to query the blockchain. In order to do this, you need to make a node that is connected to the network.

This section will cover how to create an Ethereum node with Amazon Managed Blockchain.

Code

This section covers the python code generated for this project.

Special Note:

Whenever you run files, make sure to run them in terminal by running a command similar to below

```
python filename.py
```

Connecting to Amazon Managed Blockchain Node

Special Note:

This code currently uses environment variables due to issues connecting to the AMB Node. To access the node, you have to send a request with your signature (your AWS Credentials). Make sure that before you start a new session, copy and paste the following into the terminal with your own information:

```
export AWS_ACCESS_KEY_ID=
export AWS_SECRET_ACCESS_KEY=
export AWS_REGION=us-east-1
export AMB_HTTPS_ENDPOINT=
```

```

import logging
from typing import Any
import os

import requests
from eth_typing import URI
from web3._utils.request import _get_session
from web3.providers.rpc import HTTPProvider
from web3.types import Middleware, RPCEndpoint, RPCResponse
from requests_auth_aws_sigv4 import AWSSigV4

from web3 import Web3
from web3.middleware import geth_poa_middleware

# ----- Signature and HTTP Endpoint Setup ----- #

aws_auth = AWSSigV4(
    'managedblockchain',
    aws_access_key_id=os.environ.get('AWS_ACCESS_KEY_ID'),
    aws_secret_access_key=os.environ.get('AWS_SECRET_ACCESS_KEY'),
    region=os.environ.get('AWS_REGION') # us-east-1
)

def make_post_request(
    endpoint_uri: URI, data: bytes, *args: Any, **kwargs: Any) -> bytes:
    kwargs.setdefault('timeout', 10)
    session = _get_session(endpoint_uri)
    # https://github.com/python/mypy/issues/2582
    response = session.post(endpoint_uri, data=data,
                            *args, **kwargs, auth=aws_auth) # type: ignore
    response.raise_for_status()

    return response.content

class AMBHTTPProvider(HTTPProvider):
    def make_request(self, method: RPCEndpoint, params: Any) -> RPCResponse:
        self.logger.debug("Making request HTTP. URI: %s, Method: %s",
                          self.endpoint_uri, method)

        # .decode() since the AWS sig library expects a string.
        request_data = self.encode_rpc_request(method, params).decode()
        raw_response = make_post_request(
            self.endpoint_uri,
            request_data,
            **self.get_request_kwargs()
        )
        response = self.decode_rpc_response(raw_response)
        self.logger.debug("Getting response HTTP. URI: %s, "
                          "Method: %s, Response: %s",
                          self.endpoint_uri, method, response)

        return response

# ----- Check if we are connected to the node / network ----- #

```

```

async_provider = AMBHTTPProvider(endpoint_uri=os.environ.get('AMB_HTTPS_ENDPOINT'))
async_w3 = Web3(async_provider)

# inject the poa compatibility middleware to the innermost layer (needed for Rinkeby )
async_w3.middleware_onion.inject(geth_poa_middleware, layer=0)

# confirm that the connection is successful
print("Are we connected?:\n", async_w3.isConnected())
print("client version:\n", async_w3.clientVersion)

```

Retrieving Secrets from Secrets Manager

```

# ----- Secrets Manager Code ----- #
def get_etherscan_api_key():
    secrets_manager_client = boto3.client(
        'secretsmanager'
    )
    response = secrets_manager_client.get_secret_value(
        SecretId='EtherscanAPIKey'
    )
    secret_string = response["SecretString"]
    secret_dict = json.loads(secret_string)
    return secret_dict['APIKey']

def get_moralis_api_key():
    secrets_manager_client = boto3.client(
        'secretsmanager'
    )
    response = secrets_manager_client.get_secret_value(
        SecretId='MoralisAPIKey'
    )
    secret_string = response["SecretString"]
    secret_dict = json.loads(secret_string)
    return secret_dict['APIKey']

```

```

# ----- Secrets Manager Code ----- #

def get_wallet_private_key():
    secrets_manager_client = boto3.client(
        'secretsmanager'
    )

    iam = boto3.resource('iam')
    current_user = iam.CurrentUser().user_name

    if (current_user == 'elizabeth-dev'):
        response = secrets_manager_client.get_secret_value(
            SecretId='ElizabethWalletPrivateKeyTest'
        )
        secret_string = response["SecretString"]
        secret_dict = json.loads(secret_string)
        return secret_dict['ElizabethWalletPrivateKey']

```

```

elif (current_user == 'mrwong1'):
    response = secrets_manager_client.get_secret_value(
        SecretId='mrwong1testprivkey'
    )
    secret_string = response["SecretString"]
    secret_dict = json.loads(secret_string)
    return secret_dict['mrwong1testwalletprivkey']
else:
    print("You do not have access to any of these secrets.")

def get_etherscan_api_key():
    secrets_manager_client = boto3.client(
        'secretsmanager'
    )
    response = secrets_manager_client.get_secret_value(
        SecretId='BlockchainProjectSecrets'
    )
    secret_string = response["SecretString"]
    secret_dict = json.loads(secret_string)
    return secret_dict['EtherscanAPIKey']

def get_moralis_api_key():
    secrets_manager_client = boto3.client(
        'secretsmanager'
    )
    response = secrets_manager_client.get_secret_value(
        SecretId='MoralisAPIKey'
    )
    secret_string = response["SecretString"]
    secret_dict = json.loads(secret_string)
    return secret_dict['APIKey']

```

Using Web3 py to Make Transactions

Note: This code uses environment variables to get public wallet addresses and secrets manager to get the wallet private key. Using this method can be useful for working in cloud9 with multiple people, since the sender address, the receiver address, and the sender's private key are not hard coded. This means that the person running it sends and signs a transaction with their private key, instead of having one wallet always send the key.

To make the environment variables, copy and paste the following commands with your own info in the terminal:

```

export SENDER_WALLET_ADDRESS =
export RECEIVER_WALLET_ADDRESS =

```

```

sender_wallet_address = os.environ.get('SENDER_WALLET_ADDRESS')
receiver_wallet_address = os.environ.get('RECEIVER_WALLET_ADDRESS')
sender_wallet_private_key = get_wallet_private_key()

def make_transaction(wei_to_send):
    signed_txn = async_w3.eth.account.sign_transaction(dict(
        nonce=async_w3.eth.get_transaction_count(sender_wallet_address),

```



```

        maxFeePerGas=3000000000,
        maxPriorityFeePerGas=2000000000,
        gas= 100000 ,
        to=receiver_wallet_address,
        value= wei_to_send,
        data=b'',
        type=2, # (optional) the type is now implicitly set based on appropriate transaction
        chainId= async_w3.eth.chainId,
    ),
    sender_wallet_private_key,
)

transaction_hash = async_w3.eth.send_raw_transaction(signed_txn.rawTransaction)

def make_several_transactions(wei_to_send_list):
    loop_nonce = async_w3.eth.get_transaction_count(sender_wallet_address)
    for list_value in wei_to_send_list:
        signed_txn = async_w3.eth.account.sign_transaction(dict(
            nonce= loop_nonce,
            maxFeePerGas=3000000000,
            maxPriorityFeePerGas=2000000000,
            gas= 100000 ,
            to=receiver_wallet_address,
            value= list_value,
            data=b'',
            type=2, # (optional) the type is now implicitly set based on appropriate transaction
            chainId= async_w3.eth.chainId,
        ),
        sender_wallet_private_key,
    )

        transaction_hash = async_w3.eth.send_raw_transaction(signed_txn.rawTransaction)
        loop_nonce = loop_nonce+1

```

Using Web3 py to Query the AMB Node

```

def get_latest_transaction_from_wallet(wallet_address):
    current_block = async_w3.eth.get_block('latest')
    current_block = current_block['number']
    block_transaction_list = async_w3.eth.get_block(current_block)
    block_transaction_list = block_transaction_list['transactions']
    account_transaction_hash = ''
    transactions_found = 0

    print('current block number:', current_block)
    while (transactions_found < 1):
        print('starting on block:', current_block)
        print("number of transaction in list:", len(block_transaction_list))
        for transaction in block_transaction_list:
            transaction_data = async_w3.eth.get_transaction(Web3.toHex(transaction))
            if ((transaction_data['to'] == wallet_address) or (transaction_data['from'] == wallet_address)):
                transactions_found += 1
                account_transaction_hash_list = Web3.toHex(transaction)
                print('Found latest transaction')

```

```

        print('Done with block:', current_block)
        current_block = current_block - 1
        block_transaction_list = async_w3.eth.get_block(current_block)
        block_transaction_list = block_transaction_list['transactions']

    return account_transaction_hash

def get_wallet_history(wallet_address):
    account_transaction_count = 4 #async_w3.eth.get_transaction_count(wallet_address)
    current_block = async_w3.eth.get_block('latest')
    current_block = current_block['number']
    block_transaction_list = async_w3.eth.get_block(current_block)
    block_transaction_list = block_transaction_list['transactions']
    account_transaction_hash_list = []
    transactions_found = 0

    print('current block number:', current_block)
    while (transactions_found < account_transaction_count):
        print('starting on block:', current_block)
        print("number of transaction in list:", len(block_transaction_list))
        for transaction in block_transaction_list:
            transaction_data = async_w3.eth.get_transaction(Web3.toHex(transaction))
            if ((transaction_data['to'] == wallet_address) or (transaction_data['from']
                transactions_found += 1
                account_transaction_hash_list.append(Web3.toHex(transaction))
                print('Found transaction number', transactions_found, 'of', account_tr
        print('Done with block:', current_block)
        current_block = current_block - 1
        block_transaction_list = async_w3.eth.get_block(current_block)
        block_transaction_list = block_transaction_list['transactions']

    return account_transaction_hash_list

```

Making API Requests to Etherscan

```

def mainnet_make_api_url(module, action, mainnet_address, **kwargs):
    url = MAINNET_BASE_URL + f"?module={module}&action={action}&address={mainnet_address}"

    for key, value in kwargs.items():
        url += f"&{key}={value}"

    return url

def mainnet_make_api_url_no_address(module, action, **kwargs):
    url = MAINNET_BASE_URL + f"?module={module}&action={action}&apikey={API_KEY}"

    for key, value in kwargs.items():
        url += f"&{key}={value}"

    return url

```

Etherscan Examples

```

def mainnet_get_transactions(mainnet_address):
    transactions_url = mainnet_make_api_url("account", "txlist", mainnet_address, star
    response = get(transactions_url)
    data = response.json()["result"]

    internal_tx_url = mainnet_make_api_url("account", "txlistinternal", mainnet_addres
    response2 = get(internal_tx_url)
    data2 = response2.json()["result"]

    data.extend(data2)
    data.sort(key=lambda x: int(x['timeStamp']))

    current_balance = 0
    balances = []
    times = []

    for tx in data:
        to = tx["to"]
        from_addr = tx["from"]
        value = int(tx["value"]) / ETHER_VALUE

        if "gasPrice" in tx:
            gas = int(tx["gasUsed"]) * int(tx["gasPrice"]) / ETHER_VALUE
        else:
            gas = int(tx["gasUsed"]) / ETHER_VALUE

        time = datetime.fromtimestamp(int(tx['timeStamp']))
        money_in = to.lower() == mainnet_address.lower()

        if money_in:
            current_balance += value
        else:
            current_balance -= value + gas

        balances.append(current_balance)
        times.append(time)

    list_to_return = [times, balances]
    return list_to_return

```

Making API Requests to Moralis

Note: Moralis uses pagination. The code below collects up to 1,000 records (or 10 pages with 100 records each) by getting the 'cursor' from each response. To get all records, replace the while loop condition

```

while( (req['page'] * req['page_size']) < 1000): # limit to 1,000 records for now

```

with

```

while( (req['page'] * req['page_size']) < req['total']): # get all records

```

```

from requests.auth import HTTPBasicAuth

def moralis_get_nft_owners(contract_address, **kwargs):
    moralis_api_key = get_moralis_api_key()
    url = 'https://deep-index.moralis.io/api/v2/nft/' + contract_address + '/owners?c'
    headers = {'Accept': 'application/json', 'x-api-key': moralis_api_key}
    auth = HTTPBasicAuth('x-api-key', moralis_api_key)
    req = get(url, headers=headers, auth=auth)
    req = req.json()

    cursor = req['cursor']
    results = req['result']

    while( (req['page'] * req['page_size']) < 1000): # limit to 1,000 records for now
        url = 'https://deep-index.moralis.io/api/v2/nft/' + contract_address + '/owner'
        headers = {'Accept': 'application/json', 'x-api-key': moralis_api_key}
        auth = HTTPBasicAuth('x-api-key', moralis_api_key)
        time.sleep(1.1) # needed to not hit request rate limit
        req = get(url, headers=headers, auth=auth)
        req = req.json()
        print('page:', req['page'])
        print('page size:', req['page_size'])
        print((req['page'] * req['page_size']), "/", req['total'], "seen")
        results += req['result']
        cursor = req['cursor']

    return results

```

Moralis Examples

```

# Need prices of tokens in a certain collection for each wallet
def get_token_balances_specific_address(wallet_address):
    moralis_api_key = get_moralis_api_key()
    url = 'https://deep-index.moralis.io/api/v2/' + wallet_address + '/erc20?chain=et'
    headers = {'Accept': 'application/json', 'x-api-key': moralis_api_key}
    auth = HTTPBasicAuth('x-api-key', moralis_api_key)

    req = get(url, headers=headers, auth=auth)
    req = req.json()

    return req

def get_token_balances_specific_wallet_address_and_collection(wallet_address, token_ac
    moralis_api_key = get_moralis_api_key()
    url = 'https://deep-index.moralis.io/api/v2/' + wallet_address + '/erc20?chain=et'
    headers = {'Accept': 'application/json', 'x-api-key': moralis_api_key}
    auth = HTTPBasicAuth('x-api-key', moralis_api_key)

    req = get(url, headers=headers, auth=auth)
    req = req.json()

    return req

```

Making API Requests to OpenSea

```
def retrieve_collection(collection_slug):
    url = "https://api.opensea.io/api/v1/collection/" + collection_slug
    response = requests.get(url)
    return response.text

def retrieve_collection_stats(collection_slug):
    url = "https://api.opensea.io/api/v1/collection/" + collection_slug + "/stats"
    headers = {"Accept": "application/json"}
    response = requests.get(url, headers=headers)
    return response.text
```

Graphing Data Using Matplotlib

```
def make_graph(xdata, ydata, figure_number, file_name):

    plt.figure(figure_number)
    plt.plot(xdata, ydata)
    plt.savefig(file_name)
```

```
make_graph(data[0], data[1], 1, 'rinkebygraph.png')
```

- Explain what (1) means in this example function call

Creating CSV Files

```
def make_csv_of_NFT_tokens(collection_slug, response):
    # Create file name and path
    string_list = collection_slug.split(" ")
    file_path = ""
    for i in string_list:
        file_path = file_path + i + "_"
    file_path = file_path + "tokens.csv"
    file_to_upload = open(file_path, 'w')

    # Prepare csv and csv header
    writer = csv.writer(file_to_upload)
    header = ['token_address', 'token_id', 'amount', 'owner_of', 'token_hash', 'block_
    writer.writerow(header)

    # write response data to csv
    for i in range (0, len(response)):
        row = [response[i]['token_address'],
        response[i]['token_id'],
        response[i]['amount'],
        response[i]['owner_of'],
        response[i]['token_hash'],
        response[i]['block_number_minted'],
        response[i]['block_number'],
        response[i]['updated_at'],
```

```

        response[i]['contract_type'],
        response[i]['name'],
        response[i]['symbol'],
        response[i]['token_uri']]
    writer.writerow(row)

    file_size = os.path.getsize(file_path)
    print(file_path, "is :", file_size, "bytes")

    file_to_upload.close()

    return file_to_upload

```

```

response = moralis_get_nft_owners(TOP_TEN_NFT_COLLECTIONS["bored ape yacht club"])
make_csv_of_NFT_tokens("bored ape yacht club", response)

```

Uploading Files To S3 Bucket

```

def upload_file(file_name, bucket, object_name=None):
    """Upload a file to an S3 bucket

    :param file_name: File to upload
    :param bucket: Bucket to upload to
    :param object_name: S3 object name. If not specified then file_name is used
    :return: True if file was uploaded, else False
    """

    # If S3 object_name was not specified, use file_name
    if object_name is None:
        object_name = os.path.basename(file_name)

    # Upload the file
    s3_client = boto3.client('s3')
    response = s3_client.upload_file(file_name, bucket, object_name)

```

```

file = make_csv_of_NFT_wallets_balance_general("bored ape yacht club", sorted_owners)
upload_file(file.name, 'blockchainproject-generaldata', 'test/wallet_balance_general/'

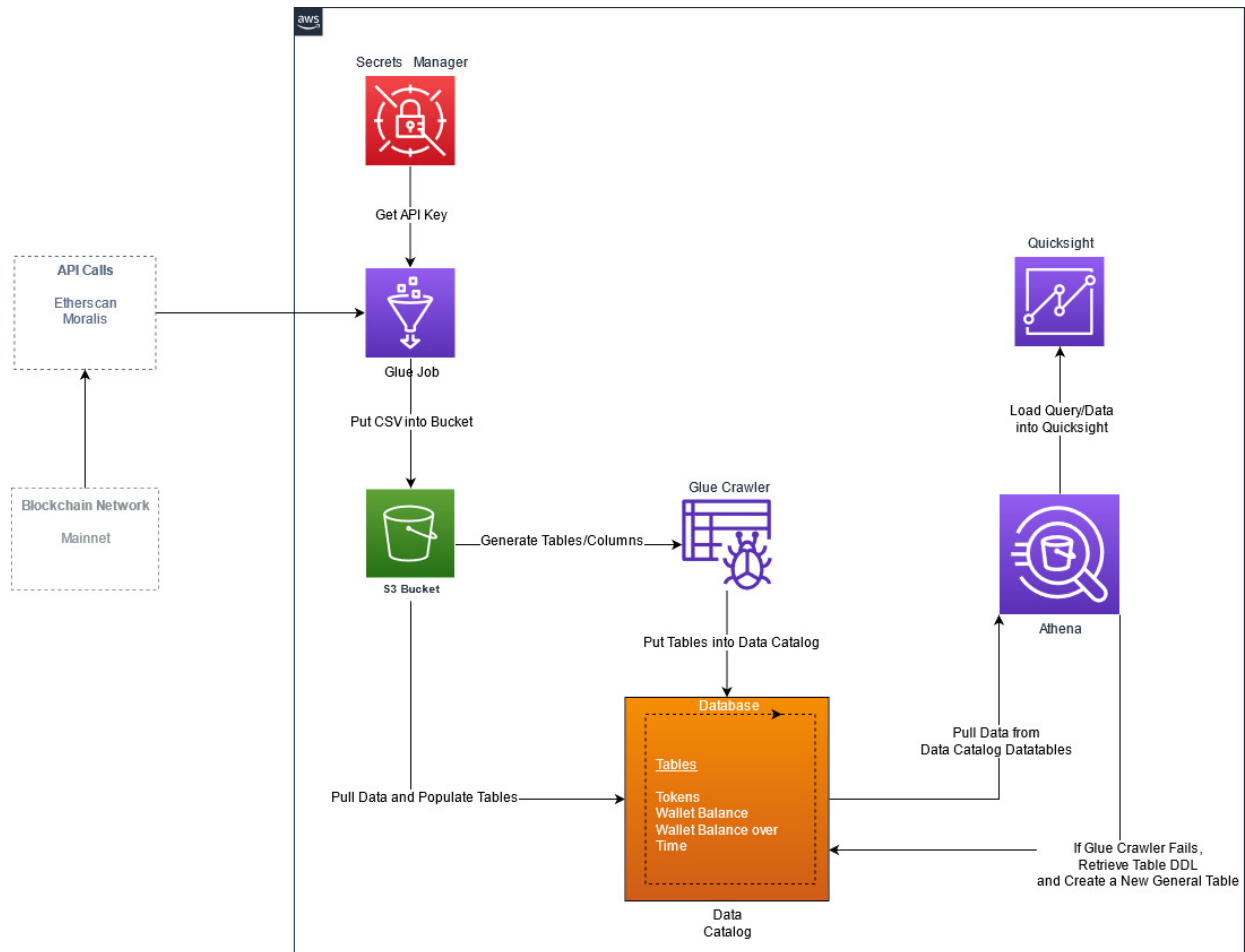
```

The following folder structure is recommended for upload files:

- Data
 - Tokens
 - bored_ape_yacht_club_tokens.csv
 - cryptopunks_tokens.csv
 -
 - Current Wallet Balances
 - 0x48c04ed5691981C42154C6167398f95e8f38a7fF_wallet_balance.csv
 - 0x77C5C3487B5fD4267a618fA4F58395e8EC803C02_wallet_balance.csv

-
- Wallet Balances Over Time
 - 0x48c04ed5691981C42154C6167398f95e8f38a7fF_wallet_balance_over_time.csv
 - 0x77C5C3487B5fD4267a618fA4F58395e8EC803C02_wallet_balance_over_time.csv
 -

DATA INGESTION PIPELINE



Set Up Service Roles

In order for the job to run, you have to pass permissions to access the S3 bucket and secrets manager.

Creating Glue Job

The code block below was run as a Glue job. To create the glue job:

1. Navigate to the glue console, and click jobs
2. Select 'Python shell script editor' and click create
3. Copy and paste the code below

```

import sys
import os # To use get_size() function to find out csv file size
import csv # creating the files to upload
import time # For forcing stops (time.sleep()) between API requests to avoid limit
import json # For interpreting json responses from API calls
import boto3 # To get API Keys from Secrets manager and put objects in an S3 bucket
import logging
from requests import get # For Etherscan and Moralis API calls
from requests.auth import HTTPBasicAuth # For Moralis API calls
from datetime import datetime # For putting timestamps in data

# ----- Globals Creation----- #
API_KEY = ""
MAINNET_BASE_URL = "https://api.etherscan.io/api"
ETHER_VALUE = 10 ** 18

TOP_TEN_NFT_COLLECTIONS = {
    'bored ape yacht club': '0xabc4ca0eda7647a8ab7c2061c2e118a18a936f13d',
    'mutant ape yacht club': '0x60e4d786628fea6478f785a6d7e704777c86a7c6',
    'otherdeed for otherside': '0x34d85c9CDeB23FA97cb08333b511ac86E1C4E258',
    'art blocks curated': '0xa7d8d9ef8d8ce8992df33d8b8cf4aebabd5bd270',
    'azuki': '0xed5af388653567af2f388e6224dc7c4b3241c544',
    'decentraland': '0xf87e31492faf9a91b02ee0deaad50d51d56d5d4d',
    'clone x x-takashi murakami': '0x49cf6f5d44e70224e2e23fdcdd2c053f30ada28b',
    'The Sandbox': '0x5cc5b05a8a13e3fbbdb0bb9fccd98d38e50f90c38',
    'moonbirds': '0x23581767a106ae21c074b2276d25e5c3e136a68b',
    'cryptopunks': '0xb47e3cd837dDF8e4c57F05d70Ab865de6e193BBB'
}

# ----- Secrets Manager Code ----- #
def get_etherscan_api_key():
    secrets_manager_client = boto3.client(
        'secretsmanager'
    )
    response = secrets_manager_client.get_secret_value(
        SecretId='BlockchainProjectSecrets'
    )
    secret_string = response["SecretString"]
    secret_dict = json.loads(secret_string)
    return secret_dict['EtherscanAPIKey']

# ----- Etherscan API Calls ----- #
def mainnet_make_api_url(module, action, mainnet_address, **kwargs):
    API_KEY = get_etherscan_api_key()
    # print("See if API_KEY has a value in mainnet_make_api_url:", API_KEY)
    url = MAINNET_BASE_URL + f"?module={module}&action={action}&address={mainnet_address}"
    for key, value in kwargs.items():
        url += f"&{key}={value}"
    return url

def mainnet_make_api_url_no_address(module, action, **kwargs):
    print("See if API_KEY has a value in mainnet_make_api_url:", API_KEY)
    url = MAINNET_BASE_URL + f"?module={module}&action={action}&apikey={API_KEY}"

    for key, value in kwargs.items():
        url += f"&{key}={value}"
    return url

```



```

def mainnet_get_account_balance(mainnet_address):
    balance_url = mainnet_make_api_url("account", "balance", mainnet_address, tag="lat
    response = get(balance_url)
    data = response.json()
    value = int(data["result"]) / ETHER_VALUE
    return value

def mainnet_get_transactions(mainnet_address):
    transactions_url = mainnet_make_api_url("account", "txlist", mainnet_address, star
    response = get(transactions_url)
    data = response.json()["result"]

    internal_tx_url = mainnet_make_api_url("account", "txlistinternal", mainnet_addres
    response2 = get(internal_tx_url)
    data2 = response2.json()["result"]

    data.extend(data2)
    data.sort(key=lambda x: int(x['timeStamp']))

    current_balance = 0
    balances = []
    times = []

    for tx in data:
        to = tx["to"]
        from_addr = tx["from"]
        value = int(tx["value"]) / ETHER_VALUE

        if "gasPrice" in tx:
            gas = int(tx["gasUsed"]) * int(tx["gasPrice"]) / ETHER_VALUE
        else:
            gas = int(tx["gasUsed"]) / ETHER_VALUE

        time = datetime.fromtimestamp(int(tx['timeStamp']))
        money_in = to.lower() == mainnet_address.lower()

        if money_in:
            current_balance += value
        else:
            current_balance -= value + gas

        balances.append(current_balance)
        times.append(time)

    list_to_return = [times, balances]
    return list_to_return

def make_wallet_balance_over_time_csv(wallet_address, times_balances_data_list):
    file_path = wallet_address + "_wallet_balance_over_time.csv"
    file_to_upload = open(file_path, 'w')

    times = times_balances_data_list[0]
    balances = times_balances_data_list[1]

```

```

# Prepare csv and csv header
writer = csv.writer(file_to_upload)
header = ['wallet_address', 'balance', 'timestamp']
writer.writerow(header)

for i in range(0, len(times)):
    row = [wallet_address, balances[i], times[i]]
    writer.writerow(row)

file_to_upload.close()

return file_to_upload

# ----- Moralis API Calls -----#
def moralis_get_nft_owners(contract_address, **kwargs):
    url = 'https://deep-index.moralis.io/api/v2/nft/' + contract_address + '/owners?c'
    headers = {'Accept': 'application/json', 'x-api-key': 'RA5E6jx6IdGb0fCLccijReUXvNN'}
    auth = HTTPBasicAuth('x-api-key', 'RA5E6jx6IdGb0fCLccijReUXvNNyWzSTjuMMCqTRtgkDS9z')

    req = get(url, headers=headers, auth=auth)
    req = req.json()

    cursor = req['cursor']
    results = req['result']

    while( (req['page'] * req['page_size']) < 1000): # limit to 1,000 records for now
        url = 'https://deep-index.moralis.io/api/v2/nft/' + contract_address + '/owner'
        headers = {'Accept': 'application/json', 'x-api-key': 'RA5E6jx6IdGb0fCLccijReUXvNNyWzSTjuMMCqTRtgkDS9z'}
        auth = HTTPBasicAuth('x-api-key', 'RA5E6jx6IdGb0fCLccijReUXvNNyWzSTjuMMCqTRtgkDS9z')
        time.sleep(1.1) # needed to not hit request rate limit
        req = get(url, headers=headers, auth=auth)
        req = req.json()
        print('page:', req['page'])
        print('page size:', req['page_size'])
        print((req['page'] * req['page_size']), "/", req['total'], "seen")
        results += req['result']
        cursor = req['cursor']

    return results

def make_csv_of_NFT_tokens(collection_slug, response):
    # Create file name and path
    string_list = collection_slug.split(" ")
    file_path = ""
    for i in string_list:
        file_path = file_path + i + "_"
    file_path = file_path + "tokens.csv"
    file_to_upload = open(file_path, 'w')

    # Prepare csv and csv header
    writer = csv.writer(file_to_upload)
    header = ['token_address', 'token_id', 'amount', 'owner_of', 'token_hash', 'block_']
    writer.writerow(header)

```

```

# write response data to csv
for i in range (0, len(response)):
    row = [response[i]['token_address'],
           response[i]['token_id'],
           response[i]['amount'],
           response[i]['owner_of'],
           response[i]['token_hash'],
           response[i]['block_number_minted'],
           response[i]['block_number'],
           response[i]['updated_at'],
           response[i]['contract_type'],
           response[i]['name'],
           response[i]['symbol'],
           response[i]['token_uri']]
    writer.writerow(row)

file_size = os.path.getsize(file_path)
print(file_path, "is :", file_size, "bytes")

file_to_upload.close()

return file_to_upload

```

```

def make_csv_of_NFT_wallets_balance_general(collection_slug, wallets_list):
    # Create file name and path
    string_list = collection_slug.split(" ")
    file_path = ""
    for i in string_list:
        file_path = file_path + i + "_"
    file_path = file_path + "wallet_balance_general.csv"
    file_to_upload = open(file_path, 'w')

    # Prepare csv and csv header
    writer = csv.writer(file_to_upload)
    header = ['wallet_address', 'total_wallet_balance']
    writer.writerow(header)

    # write response data to csv
    for i in wallets_list:
        row = []
        row.append(i[0]) # wallet address
        row.append(mainnet_get_account_balance(i[0]))
        print(row)
        writer.writerow(row)
        time.sleep(0.1)

    file_size = os.path.getsize(file_path)
    print(file_path, "is :", file_size, "bytes")

    file_to_upload.close()

    return file_to_upload

```

```

def get_distribution_of_owners(response):
    """ A little bit of a helper method for make_csv_of_NFT_wallets_balance. """
    newlist = sorted(response, key=lambda d: d['owner_of']) # Might make things faster
    # Get distribution of owners (how many tokens of that collection does each wallet
    owners_num_of_tokens = {}
    for i in newlist:
        if (i['owner_of'] in owners_num_of_tokens):
            owners_num_of_tokens[i['owner_of']] += 1
        else:
            owners_num_of_tokens[i['owner_of']] = 1

    return owners_num_of_tokens

# ----- General Methods ----- #
def upload_file(file_name, bucket, object_name=None):
    """Upload a file to an S3 bucket

    :param file_name: File to upload
    :param bucket: Bucket to upload to
    :param object_name: S3 object name. If not specified then file_name is used
    :return: True if file was uploaded, else False
    """

    # If S3 object_name was not specified, use file_name
    if object_name is None:
        object_name = os.path.basename(file_name)

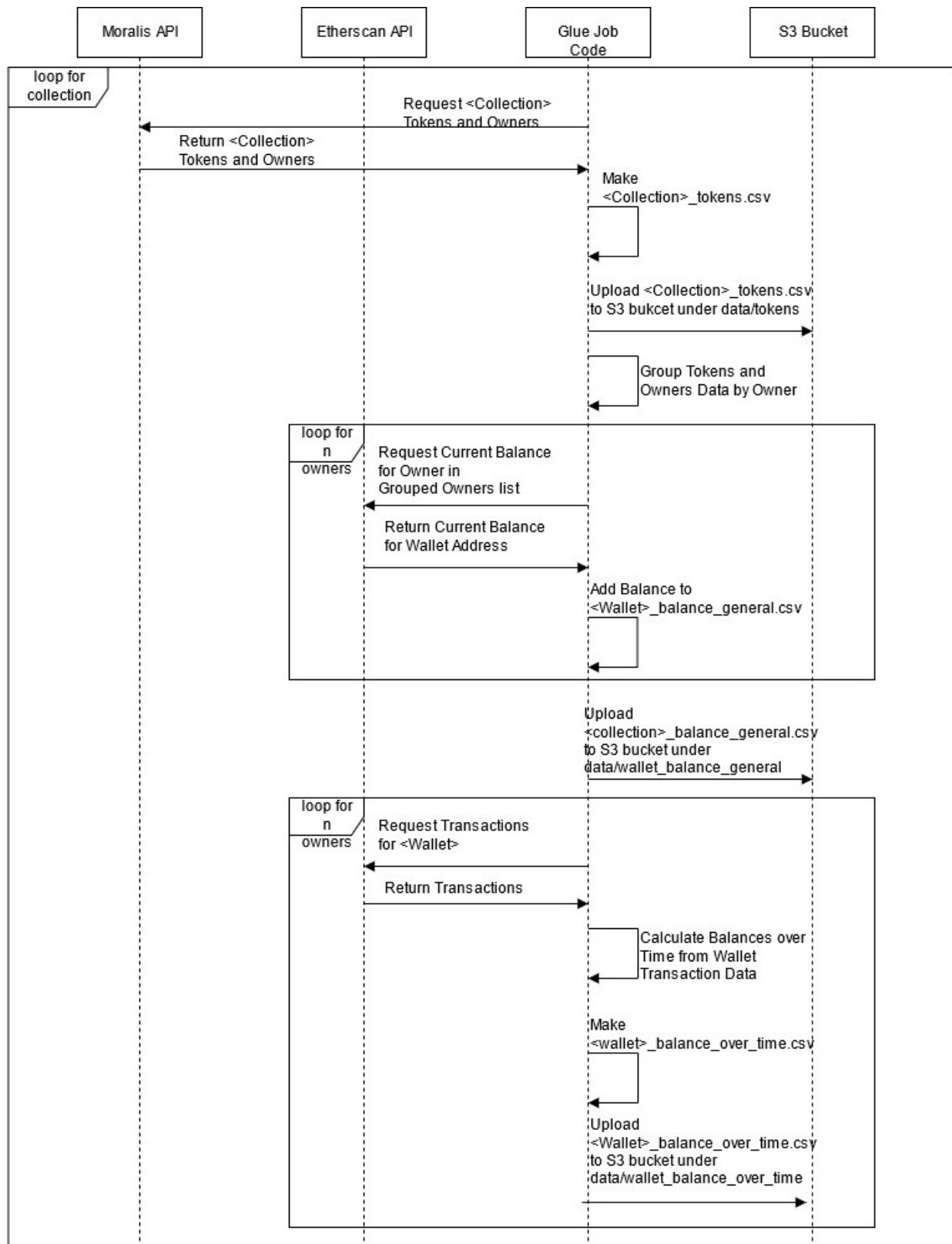
    # Upload the file
    s3_client = boto3.client('s3')
    response = s3_client.upload_file(file_name, bucket, object_name)

# TODO: Make this more like a loop and parameterize it for any collection
def main():
    response = moralis_get_nft_owners(TOP_TEN_NFT_COLLECTIONS["bored ape yacht club"])
    file = make_csv_of_NFT_tokens("bored ape yacht club", response)
    upload_file(file.name, 'blockchainproject-generaldata', 'test/tokens/'+file.name)
    owners_num_of_tokens = get_distribution_of_owners(response)
    print("Get addresses for that collection (sorting by value)")
    sorted_owners = sorted(owners_num_of_tokens.items(), key=lambda x: x[1], reverse=True)
    print(sorted_owners)
    file = make_csv_of_NFT_wallets_balance_general("bored ape yacht club", sorted_owners)
    upload_file(file.name, 'blockchainproject-generaldata', 'test/wallet_balance_general.csv')
    for i in sorted_owners:
        print(i)
        address = i[0]
        times_balances_data_list = mainnet_get_transactions(address)
        file = make_wallet_balance_over_time_csv(address, times_balances_data_list)
        upload_file(file.name, 'blockchainproject-generaldata', 'test/wallet_balance_over_time.csv')

main()

```

Here is a sequence diagram for how this code works:



General Debugging Tips:

- Under the job run details, you can check the output logs and error logs for that particular job run.
- Some debugging statements are already printed into the output logs. Feel free to delete these print statements.
- If you are having persistent problems, debug in cloud9 rather than Glue to lower costs.

Set Up Data Catalog

To set up the data catalog, your S3 Bucket must be populated. You can do this by either running the glue job to automatically upload the csv files into your S3 bucket, or you can manually upload the files after running something similar to the code under Creating CSV Files.

Using Glue Crawler to Create Table Definitions

After you have populated an S3 bucket with csv files, you will need to set up a data catalog database, with tables that follow your S3 bucket folder structure.

1. Navigate to the glue console, and select crawlers
2. Click Add crawler
3. Follow instructions to name the crawler. You do not have to open advanced settings.

Specify crawler source type

Choose Existing catalog tables to specify catalog tables as the crawler source. The selected tables specify the data stores to crawl. This option doesn't support JDBC data stores.

Crawler source type

☒ Data stores

☐ Existing catalog tables

Repeat crawls of S3 data stores

☒ Crawl all folders

Crawl all folders again with every subsequent crawl.

☐ Crawl new folders only

Only Amazon S3 folders that were added since the last crawl will be crawled. If the schemas are compatible, new partitions will be added to existing tables.

☐ Crawl changed folders identified by Amazon S3 Event Notifications

Rely on Amazon S3 events to control what folders to crawl.

BackNext

4. In specify crawler source type, leave default settings.

Add a data store

Choose a data store

S3

Connection

Select a connection

Optionally include a Network connection to use with this S3 target. Note that each crawler is limited to one Network connection so any future S3 targets will also use the same connection (or none, if left blank).

Add connection

Crawl data in

☒ Specified path

Include path

s3://[REDACTED]/data

All folders and files contained in the include path are crawled. For example, type s3://MyBucket/MyFolder/ to crawl all objects in MyFolder within MyBucket.

Sample size (optional)

Enter a number between 1 and 249

This field sets the number of files in each leaf folder to be crawled. If not set, all the files are crawled.

► Exclude patterns (optional)

Back Next

5. In 'Add a data store', select S3 as your data store, leave connection blank, and include the path to your data folder, which should include sub-folders for the type of data you are storing (ex: tokens, wallet_balance). Click next.
6. No other data store is needed, click next.

Choose an IAM role

The IAM role allows the crawler to run and access your Amazon S3 data stores. [Learn more](#)

☐ Update a policy in an IAM role

☐ Choose an existing IAM role

☒ Create an IAM role

IAM role ⓘ

AWSGlueServiceRole- YourCrawlerName

To create an IAM role, you must have **CreateRole**, **CreatePolicy**, and **AttachRolePolicy** permissions.

Create an IAM role named **"AWSGlueServiceRole-rolename"** and attach the AWS managed policy, **AWSGlueServiceRole**, plus an inline policy that allows read access to:

- s3://blockchainproject-generaldata/data

You can also create an IAM role on the [IAM console](#).

Back Next

7. Under 'Choose an IAM role', select Create an IAM role to allow the console to automatically make a role for you.

Configure the crawler's output

Database ⓘ

Choose a database to contain tables

Add database

Prefix added to tables (optional) ⓘ

Type a prefix added to table names

- Grouping behavior for S3 data (optional)
- Configuration options (optional)

Back
Next

8. In 'Configure the crawler's output', choose 'Add database'. (See definition of database in [blue data catalog documentation](#))

Add database

Database name

yourdatabasename

▸ Description and location (optional)

Resource link name

Enter resource link name

Shared database suggestions

Choose a database to autofill form

Shared database

Enter database to link to

Shared database owner account ID

Enter an AWS account ID

Create

9. Fill in the database name. You do not have to fill in any other field. This will create a new data catalog database.

10. Keep default setting, click next, and review.

Now, to populate your data catalog database, run the crawler.

Currently, there is an issue with crawler where it does not recognize header names and the headers are included in the data. Here are a few resources for troubleshooting:

<https://stackoverflow.com/questions/54373335/aws-glue-crawler-cannot-extract-csv-headers>

<https://docs.aws.amazon.com/glue/latest/dg/catalog-and-crawler.html>

If you cannot solve this issue, the data can be cleaned in quicksight. Follow instructions there.

Using Athena to Correct Glue Crawler Table Definitions

One issue that was common in this project was that the crawler would create tables for every csv file, even when they have the same columns across all of the files in a folder. To correct for this:

1. Navigate to the Athena console.
2. Choose a place to store Athena query results.
3. Make sure that you select that data catalog and database that was created with glue crawler.
4. Select a tables ending in tokens, then select generate table DDL.
5. Copy the result from the top to the s3 location name.
6. Change the table name so that it only says 'tokens' instead of the whole selection.
7. Paste the newly formatted code into the query box, and run the query. In the tables sections, you should see your newly created tables.
8. Repeat steps 4 - 7 for every other table type.

DATA VISUALIZATION IN QUICKSIGHT

- The following images were taken from the dashboard after creating our dataset.
- Here, left joins were made for the wallet addresses from all datasets.
- Instructions on how to make the datasets and import them can be found in the tutorials section at <https://us-east-1.quicksight.aws.amazon.com/sn/tutorial-videos>
- If the error occurs where some rows are populated with the column titles, simply create a new calculated field to exclude those rows.

The screenshot displays the Amazon QuickSight interface for configuring a data visualization. On the left, the 'Fields' panel lists available fields, including 'token_address', 'Tokens', 'Tokens Owned', 'Wallet Address', 'token_hash', 'block_number_minted', 'block_number', 'updated_at', and 'contract_type'. The 'Filters' panel shows 'No filters applied'. The main workspace shows a data model with three tables: 'tokens', 'wallet_balance_general_1_csv', and 'wallet_balance_over_1_csv'. The 'tokens' table is joined to the other two tables. The 'Join configuration' panel at the bottom shows a left join between 'tokens' (owner_of) and 'wallet_balance_general_1_csv' (wallet_address). The 'Join type' section shows 'Left' join selected. The bottom status bar shows 'Query mode: SPICE' and '29.9GB of remaining'.

Fields All fields included

Add calculated field

Search fields

Focus

All fields

Select All | None

token_address

Tokens

Tokens Owned

Wallet Address

token_hash

block_number_minted

block_number

updated_at

contract_type

Excluded fields No fields excluded

Filters No filters applied

Add filter

Query mode

Refresh now

SPICE

29.9GB of remaining

Data

tokens

wallet_balance_gene...

wallet_balance_over_...

Join configuration

Cancel

Apply

Join clauses

tokens

wallet_balance_over_time_csv

owner_of

=

wallet_address

Join type

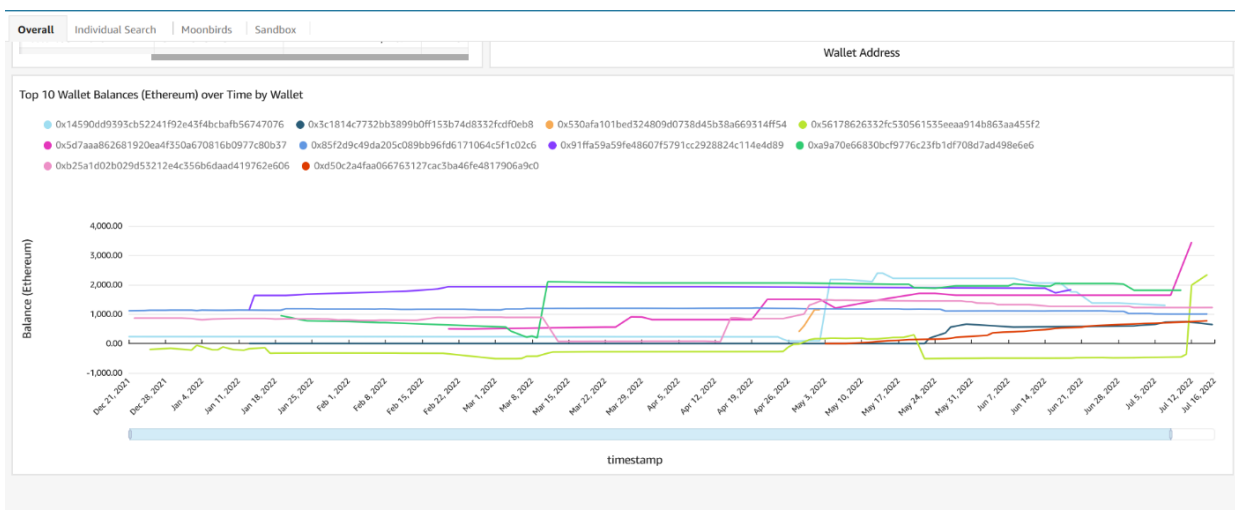
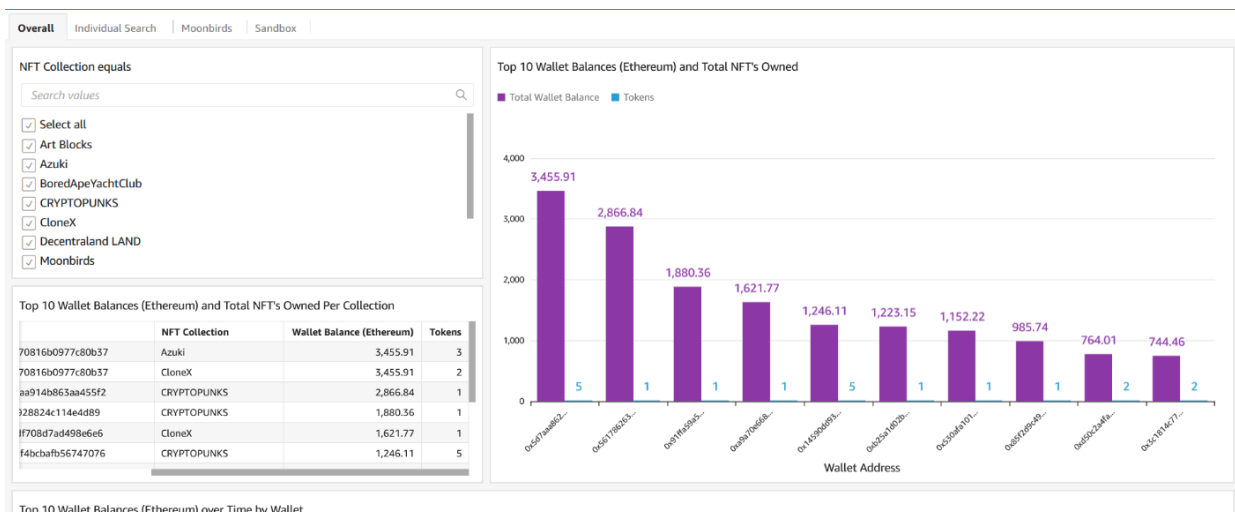
Inner

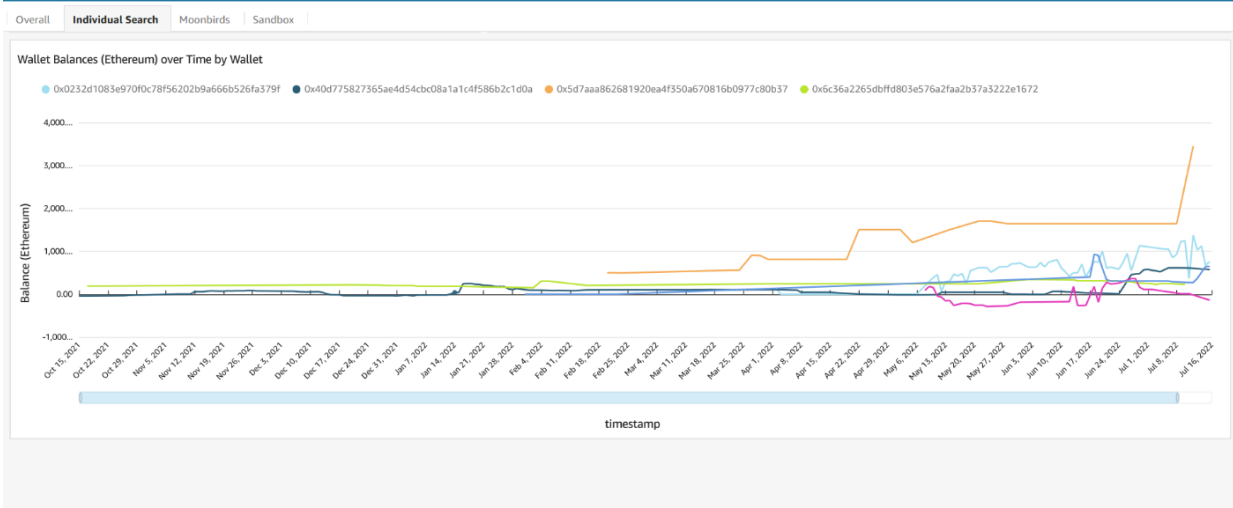
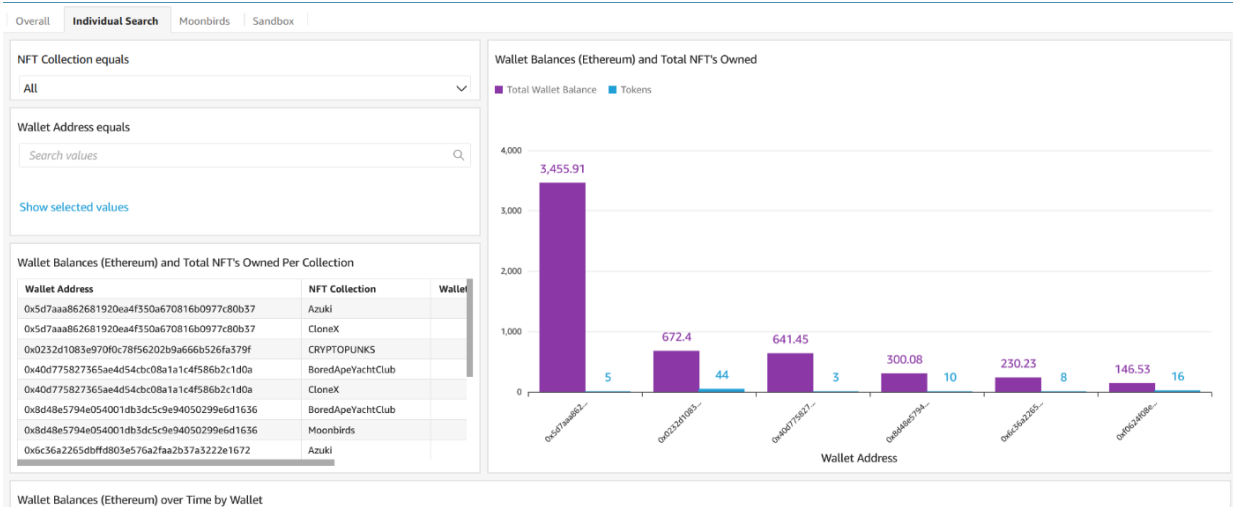
Left

Right

Full

- These are snapshots of our dashboard after having loaded the data and creating the visuals.
- The dashboard was made to show the top 10 wallets by Ethereum balance across the top 10 NFT collections, the top 10 within each NFT collection, and an individual wallet address lookup.





Estimated Costs

Cost and usage

Info

Current month costs

\$7.99

Forecasted month end costs

\$126.82






Down 56% from last month

Last month costs

\$290.15

Costs shown are unblended. [Learn more](#)

Top costs for current month

	Amazon Managed Blockchain	\$7.37
	Amazon Elastic Compute Cloud - C...	\$0.31
	AWS Secrets Manager	\$0.12
	AWS Key Management Service	\$0.12
	AWS Glue	\$0.04

Future Steps:

For future steps, we recommend

- using Cloudformation to create a template for deployment.
- running the glue script to collect all nft owners
- parametrize glue job so that is collects info for any collection rather than the top ten NFTs

References and Resources

BLOCKCHAIN DEMO

This website has a good interactive explanation of how blockchain works.

- <https://andersbrownworth.com/blockchain/>

TRACK ETHEREUM TRANSACTIONS AND BALANCE USING PYTHON

This website covers how to make API Calls to the Etherscan API. Some of the code from here is used in the project. You can watch the video here if you want another explanation.

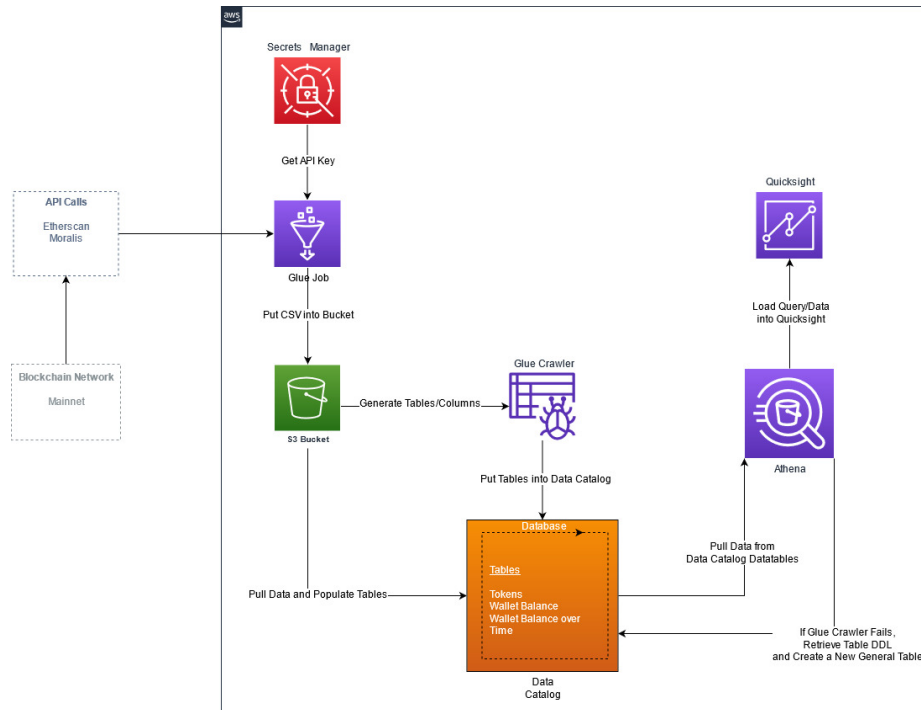
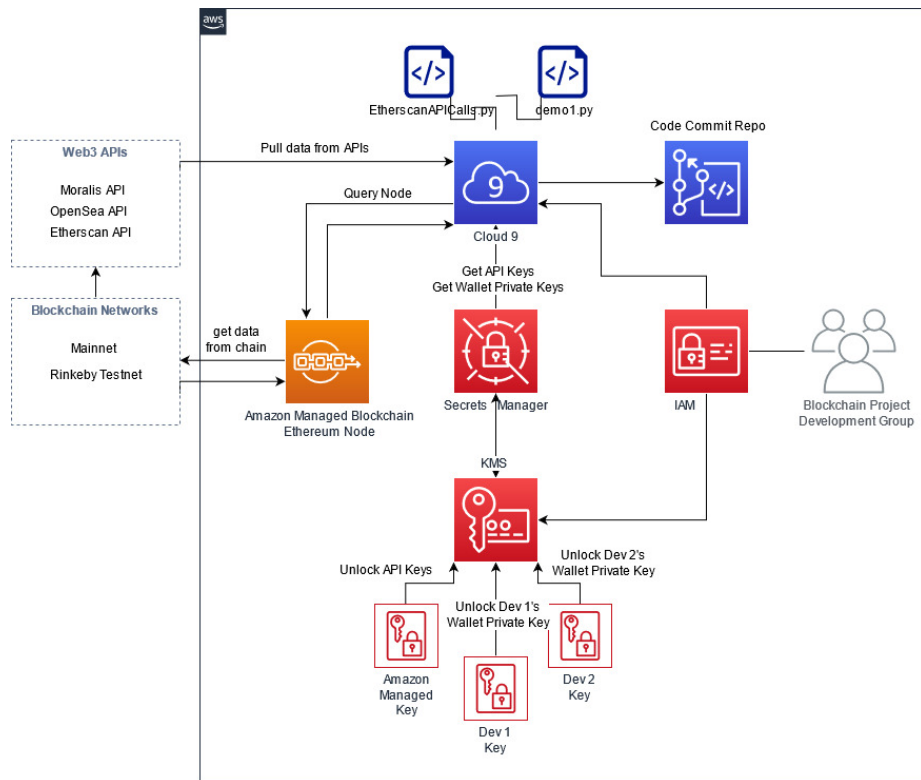
- Youtube link: <https://www.youtube.com/watch?v=x5FHbr0Em5A>
- Github link: <https://github.com/techwithtim/Ethereum-Wallet-Tracker.git>

SOLIDITY, BLOCKCHAIN, AND SMART CONTRACT COURSE – BEGINNER TO EXPERT PYTHON TUTORIAL

This youtube course covers everything including giving a nice introduction to how the blockchain works and covering different terms. You don't need to go through the whole video to understand the project.

- Youtube link: <https://youtu.be/M576WGiDBdQ>
- Github link: <https://github.com/smartcontractkit/full-blockchain-solidity-course-py>

Draft things



Put Used Cloud 9 local storage
Make Flow Diagram of Code

