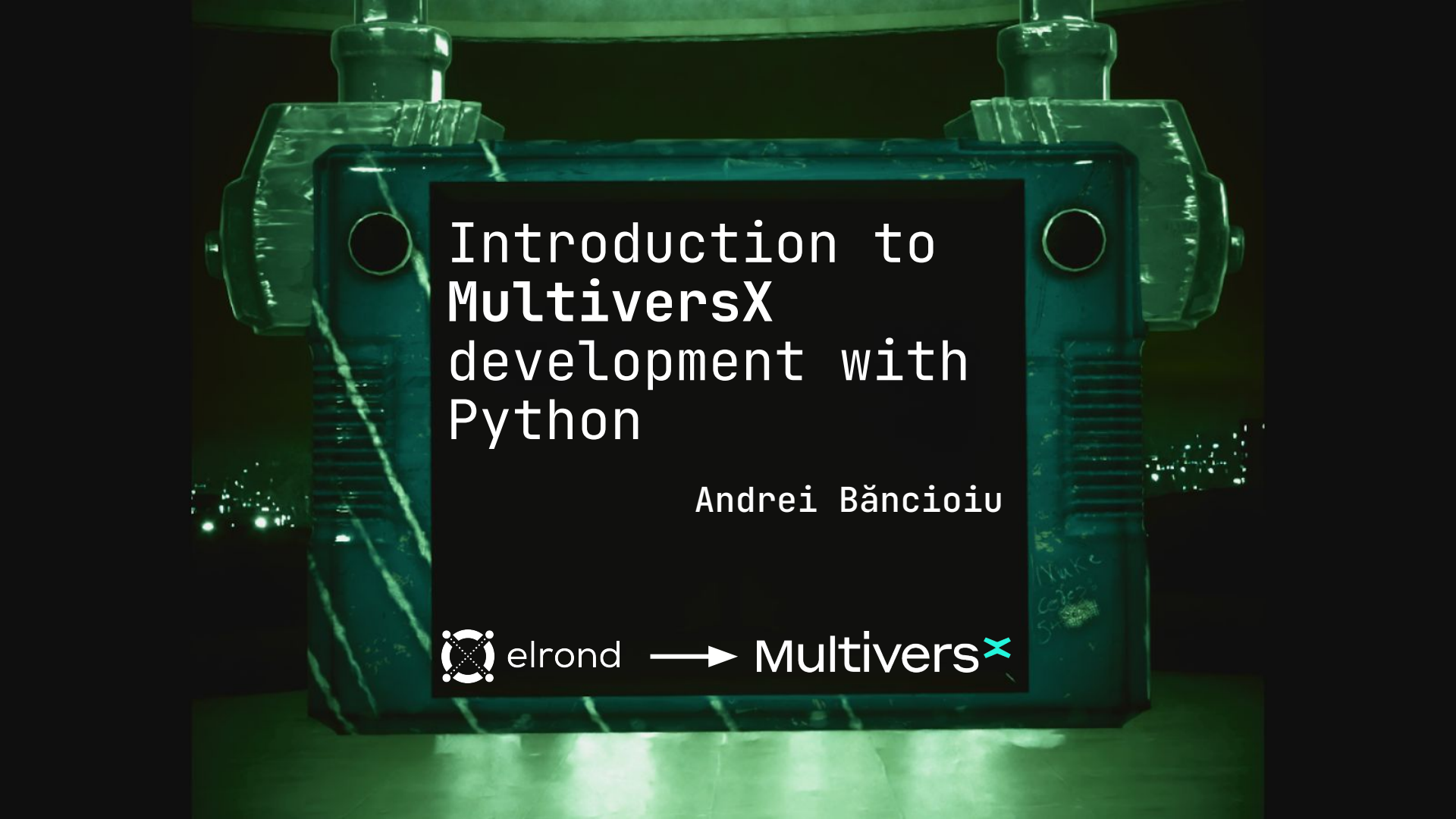
Introduction to
MultiversX
development with
Python
Andrei Băncioiu
elrond
MultiversX

2 tutorials

... to spark your creativity

You will **learn** to:

- interact with a network provider (proxy)
- create and broadcast transactions
- query account state
- handle chronology and sharding

By **means** of:

- tutorial: creating a simple **passwords manager**
- tutorial: interacting with a **deep-history** squad

Python SDK for MultiversX

- **erdpy** (primary, monolithic, swiss-knife)
- **erdpy-eggs** (experimental, modularized → erdpy vNext)
  - used within the tutorials

Tutorial 1: Passwords manager on the blockchain

- encrypt secret entries (usernames + passwords) using **pynacl**
- store passwords on **account storage** (gas cost)
- call **built-in functions** (`SaveKeyValue`)
- query and decrypt (no cost)

Passwords manager / overview

- define presentation layer (CLI)
- implement functionality:
  - initialize manager (wallet, secret)
  - create, retrieve, update and delete secret entries

Passwords manager / CLI

- arguments subparsers
  - init
  - upsert (update and insert)
  - get

```
parser = ArgumentParser()
subparsers = parser.add_subparsers()

sub = subparsers.add_parser("init", help="initialize passwords manager")
sub.set_defaults(func=init)

sub = subparsers.add_parser("upsert", help="insert and update entries")
sub.add_argument("--secret", required=True)
sub.add_argument("--wallet", required=True)
sub.add_argument("--url", required=True)
sub.set_defaults(func=upsert_entries)

sub = subparsers.add_parser("get", help="retrieve entries")
sub.add_argument("--secret", required=True)
sub.add_argument("--address", required=True)
sub.add_argument("--url", required=True)
sub.set_defaults(func=retrieve_entries)
```

Passwords manager / theory / account storage and SaveKeyValue

- the network allows storing **arbitrary data** under an account as **key-value pairs**
- `SaveKeyValue` is a **built-in** function

Passwords manager / theory / account storage and SaveKeyValue

```
SaveKeyValueTransaction {
    Sender: <account address of the wallet owner>
    Receiver: <same as sender>
    Value: 0
    GasLimit: 300000 + additional gas limit*
    Data: "SaveKeyValue" +
          "@" + <key in hexadecimal encoding> +
          "@" + <value in hexadecimal encoding> +
          "@" + <key in hexadecimal encoding> +
          "@" + <value in hexadecimal encoding> +
          ...
}
```

Passwords manager / theory / account storage and SaveKeyValue

```
GET https://devnet-gateway.elrond.com/address/erd1.../keys
{
   "data": {
       "pairs": {
           "5041...48a0": "e0bd...e64b",
           "5041...96bf": "d17d...622f",
           "5041...9e90": "33e4...2f08",
           "bb8e...2c38": "9d52...08a0"
       }
   }
}
```

Passwords manager / theory / encryption

- use a secret that's separate from the MultiversX wallet
- use symmetric encryption / `nacl.secret.SecretBox`

```
key = nacl.utils.random(nacl.secret.SecretBox.KEY_SIZE)
box = nacl.secret.SecretBox(key)
encrypted = box.encrypt(b"important secret")
plaintext = box.decrypt(encrypted)
print(plaintext)
```

Passwords manager / initialize manager

```
$ main.py init
```

```python
from erdpy_wallet import generate_pem_file

def init(args: Any):
    # Generate PEM wallet (to sign transactions)
    generate_pem_file(Path("wallet.pem"))

    # Generate secret (for pynacl's SecretBox)
    key = nacl.utils.random(nacl.secret.SecretBox.KEY_SIZE)
    with open(Path("secret.hex"), "w") as file:
        return file.write(key.hex())
```

Passwords manager / insert and update entries

```
$ main.py upsert --secret=secret.hex --wallet=wallet.pem
    --url=https://devnet-gateway.elrond.com
```

```python
from erdpy_wallet import UserSigner

def upsert_entries(args: Any):
    # Prepare the key-value pairs for account storage
    secret_key = load_secret_key(Path(args.secret))
    entries: List[SecretEntry] = ask_upsert_entries()
    pairs = [entry.to_key_value(secret_key) for entry in entries]

    # Create, sign & broadcast transaction
    signer = UserSigner.from_pem_file(Path(args.wallet))
    network_provider = CustomNetworkProvider(args.url)
    tx = create_transaction(signer, network_provider, pairs)
    tx_hash = network_provider.send_transaction(tx)
```

Passwords manager / SecretEntry

```
class SecretEntry:
    def __init__(self, label: str, username: str, password: str):
        ...

    def to_key_value(self, secret_key: bytes) -> AccountKeyValue:
        key = b"PYCHAIN_2022" + self.encrypt_label(secret_key)
        value = self.encrypt(secret_key)
        return AccountKeyValue(key, value)

    def encrypt(self, secret_key: bytes) -> bytes:
        box = nacl.secret.SecretBox(secret_key)
        data = self.serialize()
        encrypted = box.encrypt(data)
        return encrypted
```

Passwords manager / SecretEntry

```python
@classmethod
def load_many_from_storage(cls, pairs: List[AccountKeyValue], secret_key: bytes):
    return [
        SecretEntry.decrypt(pair.value, secret_key)
        for pair in pairs if pair.key.startswith(b"PYCHAIN_2022")
    ]

@classmethod
def decrypt(cls, encrypted: bytes, secret_key: bytes):
    box = nacl.secret.SecretBox(secret_key)
    data = box.decrypt(encrypted)
    return SecretEntry.deserialize(data)
```

Passwords manager / create transaction

```
def create_transaction(signer, network_provider, items: List[AccountKeyValue]):
    address = signer.get_address()
    chain_id = network_provider.get_chain_id()
    nonce = network_provider.get_account_nonce(address)
    data = SaveKeyValuesBuilder().add_items(items).build()
    gas_limit = compute_gas_limit(items, data.length())

    tx = Transaction(nonce=nonce, sender=address, receiver=address,
        gas_limit=gas_limit, data=data, chain_id=chain_id)

    tx.apply_signature(signer.sign(tx))
    return tx
```

Passwords manager / transaction

```
POST https://devnet-gateway.elrond.com/transactions
{
     "nonce": 4,
     "value": "0",
     "receiver": "erd1...",
     "sender": "erd1...",
     "gasPrice": 1000000000,
     "gasLimit": 3614000,
     "data": "U2F2...YmQ0",
     "chainID": "D",
     "version": 1,
     "signature": "5185...c707"
}
```

Passwords manager / run

```
$ main.py upsert --secret=secret.hex --wallet=wallet.pem --url=...

Next entry? (y/n)y
Label: mail.google.com
Username: foobar
Password:
Entered password of length 27
Next entry? (y/n)n

Transaction:
{ … }
Transaction is ready to be broadcasted, continue? (y/n)y
Transaction hash 533ff...2dcb
```

Passwords manager / retrieve entries

```
$ python main.py get --secret=secret.hex --address=erd1…
    --url=https://devnet-gateway.elrond.com
```

```python
def retrieve_entries(args: Any):
    secret_key = load_secret_key(Path(args.secret))
    address = Address(args.address)
    network_provider = CustomNetworkProvider(args.url)
    pairs = network_provider.get_storage(address)
    entries = SecretEntry.load_many_from_storage(pairs, secret_key)
    ask_reveal_entries(entries)
```

Passwords manager / run

```
$ python main.py get --secret=secret.hex --address=erd1... --url=...

Choose one of the following entries:
0) mail.google.com
1) facebook.com
Index:
0

Username: foobar
1) Reveal password
2) Hold password in clipboard (for a limited time)
Pick a choice!
2
```

Tutorial 2:
Querying historical state

- interact with a **deep-history** squad
  - query state at arbitrary block in the past
  - *Q: what was your EGLD balance on May the 4th?*
  - *Q: how much liquidity was there in the WEGLD / UTK pool on 1st of November?*
- serve a simple **bottle.py** API
- map a timestamp to a block index

Deep-history app / overview

- define presentation layer (**bottle.py** controllers)
- implement functionality (custom network provider):
  - fetch balances for native and custom currencies (**ESDT**)
  - fetch account storage (**Smart Contracts** included)
  - ... by **timestamp**

Deep-history app / bottle.py controllers

Get balance of native currency (EGLD):
`/<network>/accounts/<address>/native?timestamp={timestamp}`

Get balance of custom tokens (ESDTs):
`/<network>/accounts/<address>/tokens/<token>?timestamp={timestamp}`

Get whole account storage:
`/<network>/accounts/<address>/storage?timestamp={timestamp}`

Get account storage entry:
`/<network>/accounts/<address>/storage/<key>?timestamp={timestamp}`

```python
app: Any = Bottle()

@app.route("/api/<network>/accounts/<address>/native")
def get_native_balance(network: str, address: str):
    time = parse_query_parameters()
    ...

@app.route("/api/<network>/accounts/<address>/token/<token>")
def get_token_balance(network: str, address: str, token: str):
    ...

@app.route("/api/<network>/accounts/<address>/storage")
def get_whole_storage(network: str, address: str):
    ...

@app.route("/api/<network>/accounts/<address>/storage/<key>")
def get_storage_entry(network: str, address: str, key: str):
    ...
```

Deep-history app / network provider client

```
from erdpy_network import ProxyNetworkProvider

class CustomNetworkProvider(ProxyNetworkProvider):
    def __init__(self, url):
        ...

    def get_native_balance(self, address: str, time: datetime.datetime):
        block_nonce = self.get_block_by_time(address, time)
        url = f"address/{address}?blockNonce={block_nonce}"
        response = self.do_get(url)
        return response
```

Deep-history app / get_block_by_time

```
def get_block_by_time(self, address_of_interest: str, time: datetime.datetime):
    ...
```

Deep-history app / theory / rounds and blocks

- mainnet **genesis**: 2020-07-30 14:00:00 **UTC**
- round duration: 6 seconds

Examples:

- `2022-11-01 00:00:00 UTC → round 11857200`
- `2022-11-15 00:00:00 UTC → round 12058800`

Deep-history app / get_block_by_time

```
def get_block_by_time(self, address_of_interest: str, time: datetime.datetime):
    round = self.get_round_by_time(time)
    ...
```

Deep-history app / theory / sharding

```
erd1spya...66jx → shard 0
erd1qyu5...r6th → shard 1
erd1k2s3...jse8 → shard 2
...
```

Deep-history app / theory / sharding

rounds	11857200	11853385	...
$blocks_{shard\ 0}$ →	11853384	11853385	...
$blocks_{shard\ 1}$ →	11848798	11848799	...
$blocks_{shard\ 2}$ →	11853879	11853880	...
...			

Deep-history app / theory / sharding

$$\text{block}_{\text{shard 0,round}} \neq \text{block}_{\text{shard 1,round}} \neq \text{block}_{\text{shard 2,round}} \neq \text{round}$$

... due to imperfect hit rate

Deep-history app / get_block_by_time

```
def get_block_by_time(self, address_of_interest: str, time: datetime.datetime):
    round = self.get_round_by_time(time)
    shard = self.get_shard_of_address(address_of_interest)
    ...
    block = self.get_block_of_shard_by_round(shard, round)  # How?
    ...
```

Deep-history app / theory / blocks by round

```
GET https://gateway.elrond.com/blocks/by-round/11857200
```


```
[
    {
        "nonce": 11853384,
        "shard": 0
    },
    {
        "nonce": 11848798,
        "shard": 1
    },
    {
        "nonce": 11853879,
        "shard": 2
    }
]
```

Deep-history app / get_block_by_time

```
def get_block_by_time(self, address_of_interest: str, time: datetime.datetime):
    round = self.get_round_by_time(time)
    shard = self.get_shard_of_address(address_of_interest)

    for _ in range(0, MAX_NUM_BLOCKS_LOOKAHEAD):
        block = self.get_block_of_shard_by_round(shard, round)
        if block:
            return block

    raise Exception(f"Unexpected: no blocks ~{time}")
```

Deep-history app / network provider client

```
class CustomNetworkProvider(ProxyNetworkProvider):
...
    def get_token_balance(self, address: str, token: str, time: datetime.datetime):
        block_nonce = self.get_block_by_time(address, time)
        url = f"address/{address}/esdt/{token}?blockNonce={block_nonce}"
        ...

    def get_whole_storage(self, address: str, time: datetime.datetime):
        block_nonce = self.get_block_by_time(address, time)
        url = f"address/{address}/keys?blockNonce={block_nonce}")
        ...
```

Deep-history app / run (with dashboard)

Run api / mainnet / accounts / erd1k2s324ww2g0yj38qn2ch2jwctdy8mnfxep94q9arncc6xecg3xaq6mjse8 / native ? timestamp = 2022-11-01T00:00:00

Run api / mainnet / accounts / erd1k2s324ww2g0yj38qn2ch2jwctdy8mnfxep94q9arncc6xecg3xaq6mjse8 / token / WEGLD-bd4d79 ? timestamp = 2022-11-01T00:00:00

Run api / mainnet / accounts / erd1k2s324ww2g0yj38qn2ch2jwctdy8mnfxep94q9arncc6xecg3xaq6mjse8 / storage ? timestamp = 2022-11-01T00:00:00

Run api / mainnet / accounts / erd1k2s324ww2g0yj38qn2ch2jwctdy8mnfxep94q9arncc6xecg3xaq6mjse8 / storage / AABBCCDD ? timestamp = 2022-11-01T00:00:00

Response


```
{
    "blockInfo": {
        "hash": "40f51e295e2bb579bb3551c40779c20cc64eae8b68360c5188e539ca181fd427",
        "nonce": 11853879,
        "rootHash": "63f8e5a492ac62e14a0fb060557ba2828c48ecc2fa1c5b9730bf1674f6814e6e"
    },
    "tokenData": {
        "balance": "0",
        "properties": "",
        "tokenIdentifier": "WEGLD-bd4d79"
    }
}
```

Conclusion

What have we done:

- tutorial: creating a simple **passwords manager**
- tutorial: interacting with a **deep-history** squad

What have we learned:

- interact with a network provider (proxy)
- create and broadcast transactions
- query account state
- handle chronology and sharding

thank You,
have fun building!

Andrei Băncioiu

MultiversX

https://github.com/ElrondNetwork/pychain-2022

APPENDIX

Passwords manager / retrieve entries / clipboard

```python
import pyperclip

def hold_in_clipboard(data: str, seconds: int = 10):
    pyperclip.copy(data)
    sleep(seconds)
    pyperclip.copy("")
```