Introduction to Tezos for Developers

https://github.com/tzConnectBerlin/developers-presentation/

Tezos in facts and figures

- 60s blocktime (30s in next release, 10s by end of year)
- 3 elliptic curves supported: tz1 = ed25519, tz2 = Secp256k1, tz3=NIST P256
- Octez, official tezos implementation is in OCaml
- proof of stake, miners in Tezos are referred to as bakers
- on-chain democracy
- auto-upgradeable, typically every 3 months there is a new version of the protocol
- smart contracts use stack-based VM
- intercontract calling has a different model than ethereum
- gas per contract call is subject to a maximum, but you don’t pay extra for gas used
- native token is called tez

Tezos components we are going to talk about

- the **node**
- **smart contract languages** CameLIGO and SmartPy
- **javascript library** Taquito
- **python library** PyTezos

The node

- relatively easy to run oneself--hardware requirements are low
- a new version every 3 months,
- there is a dockerised sandbox called Flextesa for testing
- there are a variety of public nodes, list in the repository for this presentation
- Apart from mainnet, there are multiple testnets, for the next version of the protocol, current, and (for a while) previous versions
- comes with the tezos-client command line tool, which is useful for contract interactions

Smart contract constraints

- all operations on the contract must be O(1) -- no loops
- when the contract is invoked (and each time it is invoked) the storage is loaded and parsed, and this uses gas. **except**
- there is a data structure called a **big map**, and its storage is not loaded when the contract is loaded, so it's cheaper **but**
- you cannot iterate through the keys in the big map (but see the first point)
- big maps can only live at the top level in the contract's storage
- inter-contract calls in tezos happen **after** the calling contract is done (no re-entrancy) **and**
- storage is expensive...

Data storage

- Why not to store data on the blockchain
  - It's really expensive to store data on the blockchain.
    - 0.001 tez/byte -> ~ $2k/Megabyte
  - Sometimes it's illegal (GDPR)
    - Unsalted hash of sensitive data is sensitive
    - Cyphertext of sensitive data is sensitive
- So we don't do that.
- Instead we store a *hash* of the data on the chain and the data somewhere else.
  - Salted hash - for privacy reasons, and so it's not illegal
- The blockchain just records ownership, it doesn't store the thing which is owned.

digression: Michelson

- stack based
- strongly typed
- functional
- as you can see from the right, not very human friendly
- but it supports formal proofs, so if your contract needs to be proven correct, you are in luck
- contracts are **pure functions** which are called with a **parameter** and **storage**, and return a **list of operations to be performed** and **the new storage**.

```
DIG 3 ;
DROP ;
DIG 4 ;
DROP ;
SWAP ;
DUP ;
DUG 2 ;
CAR ;
CDR ;
CDR ;
DUP 3 ;
CAR ;
CAR ;
CAR ;
DIG 2 ;
DUP ;
CAR ;
MAP { DUP 4 ;
      SWAP ;
      DUP ;
      DUG 2 ;
      CDR ;
      MEM ;
      NOT ;
      IF { DROP ; DUP 7 ; FAILWITH }
         { DUP 3 ;
           SWAP ;
           DUP ;
           DUG 2 ;
           CDR ;
           DUP 3 ;
           CAR ;
           PAIR ;
           PAIR ;
           DUP 8 ;
           SWAP ;
           EXEC ;
           SWAP ;
           PAIR } } ;
DIG 2 ;
DROP ;
DIG 2 ;
```

Intro to our problem

- Voting application on the blockchain. The rules
- there are 3 types of person:
  - administrator: can create polls, add and remove eligible voters
  - eligible voter: one of a list who may vote in polls. Multiple votes are permitted in the same poll, only the last counts.
  - everyone else, may not change the contract state
- the contract keeps a running total of votes in each poll
- a poll has a number of valid questions, and an end date, after which no votes will be counted.
- one can vote as often as one likes, previous votes are overwritten

Our first contract storage (spoiler alert: doesn't work)

```
type poll_id = string

type poll_metadata = {
    end_date : timestamp;
    num_options : nat;
}

type votes = ( address, nat ) map
type totals = ( nat, nat ) map

type poll = {
    metadata : poll_metadata;
    votes : votes;
    totals : totals;
}

type polls = ( poll_id, poll ) map
type voters = ( address, unit ) map

type storage = {
    polls : polls;
    voters : voters;
    administrator : address;
}
```

Our working architecture

```
type voters = (address, unit) big_map
type poll_id = string
type poll_option = nat
type votes = ((address * poll_id), poll_option) big_map
type totals = (poll_option, nat) map


type poll_metadata = {
    end_date : timestamp;
    num_options : nat;
}

type poll = {
    metadata : poll_metadata;
    totals : totals;
}

let empty_totals_map : (poll_option, nat) map = Map.empty

type polls = (poll_id, poll) big_map

type storage = {
    polls : polls;
    votes : votes;
    voters : voters;
    administrator : address;
}
```

CameLIGO

- one of the LIGO family of languages
- syntax is that of ML, one of the first functional languages
- the most mature of the LIGO languages, and
- looks like OCaml, the native language of the node!
- a bit of a steep learning curve for many, **but**
- we find it the most natural way to code for tezos
- (but it’s up to you).

SmartPy

- Everyone knows Python
- Meta-programming language: python flow control structures are executed at **compile time**, there are special flow control commands for contract execution
- Python is **weakly typed** which means there are some kludgy parts to a SmartPy contract
- Optimisation is not as good as CameLIGO (this may be a controversial opinion)
- Has its own Michelson VM, so unit testing is excellent
- It’s up to you to choose which of the two to use.

SmartPy

```python
import smartpy as sp

class Poll(sp.Contract):
    def __init__(self, admin: sp.TAddress):
        self.init(
            administrator = admin,
            voters = sp.big_map(),
            votes = sp.big_map(),
            polls = sp.big_map().
        )

    def add_voter_internal(self, params):
        self.data.voters = sp.update_map(self.data.voters, params, sp.some(sp.unit))

    def remove_voter_internal(self, params):
        self.data.voters = sp.update_map(self.data.voters, params, sp.none)

    # @sp.entry_point
    def create_poll_internal(self, id, params):
        poll_meta = sp.record(end_date = params.e, num_option = params.n)
        tot = sp.map(tkey = sp.TNat, tvalue = sp.TNat)
        poll = sp.record(metadata = poll_meta, totals = tot)
        self.data.polls = sp.update_map(self.data.polls, id, sp.some(poll))

    def decrement_old_vote(self,old_vote, totals_map):
        total_vote = totals_map[old_vote]
        totals_map = sp.update_map(totals_map, old_vote, sp.some(abs(total_vote - sp.nat(1))))
        sp.result(totals_map)
```

Deploying our contract

- When we deploy a contract we do several things
- we upload the contract to the chain
- we may transfer tez to the contract
- we almost certainly initialise the storage to something appropriate
- we get back a contract identifier, which starts with ‘KT1’.
- we pay for the storage used on the chain.

Compiling our contract using CameLIGO

```
newby@stink:~/projects/tezos/developers-presentation$ ligo compile-contract poll.mligo main
{ parameter
    (or (or (address %addVoter)
            (pair %createPoll
               (string %poll_id)
               (pair %poll_metadata (timestamp %end_date) (nat %num_options))))
        (or (address %removeVoter) (pair %vote (string %poll_id) (nat %vote)))) ;
  storage
    (pair (pair (address %administrator)
                (big_map %polls
                   string
                   (pair (pair %metadata (timestamp %end_date) (nat %num_options))
                         (map %totals nat nat))))
          (pair (big_map %voters address unit) (big_map %votes (pair address string) nat))) ;
  code { LAMBDA
           address
           unit
           { SENDER ;
             COMPARE ;
             NEQ ;
             IF { PUSH string "error_NOT_AN_ADMINISTRATOR" ; FAILWITH } { UNIT } } ;
         LAMBDA
           (pair nat (map nat nat))
           nat
           { UNPAIR ; GET ; IF_NONE { PUSH nat 0 } {} } ;
         DIG 2 ;
```

Compiling our storage using CameLIGO

```
newby@stink:~/projects/tezos/developers-presentation$ ligo compile-storage poll.mligo main '
{ polls = ( Big_map.empty : polls );
  votes = ( Big_map.empty : votes );
  voters = ( Big_map.empty : voters );
  administrator = ("tz1RjonN5qEJM8cZhKcfGyoEqhw1FNB4ti6w" : address);
}'
(Pair (Pair "tz1RjonN5qEJM8cZhKcfGyoEqhw1FNB4ti6w" {}) {} {})
```

Deploying our contract using the command line (attempt 1)

```
newby@stink:~/projects/tezos/developers-presentation$ tezos-client originate contract poll transferring 0
from florence running poll.tz --init '(Pair (Pair "tz1RjonN5qEJM8cZhKcfGyoEqhw1FNB4ti6w" {}) {} {})'
Warning:

                This is NOT the Tezos Mainnet.

           Do NOT use your fundraiser keys on this network.

Node is bootstrapped.
Fatal error:
  The operation will burn ꜩ0.424 which is higher than the configured burn cap (ꜩ0).
   Use `--burn-cap 0.424` to emit this operation.
```

Deploying our contract using the command line (attempt 2)

```
newby@stink:~/projects/tezos/developers-presentation$ tezos-client originate contract poll transferring 0
from florence running poll.tz --init '(Pair (Pair "tz1RjonN5qEJM8cZhKcfGyoEqhw1FNB4ti6w" {}) {} {})' --bur
n-cap 0.424
Warning:

                This is NOT the Tezos Mainnet.

           Do NOT use your fundraiser keys on this network.

Node is bootstrapped.
Estimated gas: 7746.729 units (will add 100 for safety)
Estimated storage: 1696 bytes added (will add 20 for safety)
Operation successfully injected in the node.
Operation hash is 'onyShgLTixrmMozn22yjbn9CUzqc6Q4hQMe63VYo2m2Sh4wPRet'
Waiting for the operation to be included...
Operation found in block: BKidxzeaXRdQrut8g9Lg8SKcxU8rEdr8CX4bbBaavqRpxCRjP4P (pass: 3, offset: 2)
```

deployment (more detail)

```
        Initial storage:
          (Pair (Pair "tz1RjonN5qEJM8cZhKcfGyoEqhw1FNB4ti6w" {}) {} {})
        No delegate for this contract
        This origination was successfully applied
        Originated contracts:
          KT1B5uCAmStQazfTxrvk4jNDoaTFVyYKEc3h
        Storage size: 1439 bytes
        Updated big_maps:
          New map(88723) of type (big_map (pair address string) nat)
          New map(88722) of type (big_map address unit)
          New map(88721) of type (big_map string (pair (pair %metadata (timestamp %end_date) (nat %num_opt
ions))
                 (map %totals nat nat)))
        Paid storage size diff: 1439 bytes
        Consumed gas: 7746.729
        Balance updates:
          tz1RjonN5qEJM8cZhKcfGyoEqhw1FNB4ti6w ... -ꜩ0.35975
          tz1RjonN5qEJM8cZhKcfGyoEqhw1FNB4ti6w ... -ꜩ0.06425
```

Accessing our contract using Taquito

```
export const initPollContract = async (
  pollContractAddress: string | null = null
): Promise<void> => {
  if (!pollContractAddress || tezos === null) {
    throw new Error("Poll contract address not set or Tezos not initialized");
  }
  pollContract = await tezos.wallet.at(pollContractAddress);
};

export const createPoll = async (
  pollId: string,
  endDate: Date,
  noOfOptions: number
) => {
  const op = await pollContract.methods
    .createPoll(pollId, endDate.toISOString(), noOfOptions)
    .send();
  return op.opHash;
};
```

Wallet connection using Beacon

- Beacon is a standard for connecting wallets
- Wallets for Tezos include Kukai, Temple, Galleon.
- By using Beacon you can also use DirectAuth, permitting users to sign transactions using e.g. Google or Facebook login
- There is sample code in the repo accompanying this presentation
- Use Beacon.

Accessing our contract using PyTezos

```
(venv) newby@stink:~/projects/tezos/developers-presentation$ python3
Python 3.8.10 (default, Jun  2 2021, 10:49:15)
[GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pytezos import pytezos
>>> client = pytezos.using(key = 'edsk4LzAuuQF1FkFHV5qXmpL8a5YNtJh1pTtkAYjAVBKCSAbp6LCCD', shell = 'http:/
/florence.newby.org:8732')
>>> contract = client.contract('KT1B5uCAmStQazfTxrvk4jNDoaTFVyYKEc3h')
>>> contract.createPoll({ "poll_id": "my_poll_id_2", "poll_metadata": { "end_date": 1625660357, "num_optio
ns": 4 } }).inject()
{'chain_id': 'NetXxkAx4woPLyu', 'hash': 'oovRrKym8H5HXgVzsqmFngpjCsxR7UpQVi7RmTK78xsornvVjDJ', 'protocol':
 'PsFLorenaUUuikDWvMDr6fGBRG8kt3e3D3fHoXK1j1BFRxeSH4i', 'branch': 'BMYV8R95HzQTXhMoyZ8qzgJoi1CUNVrYC5LhkQM
F5Ucn37XMfJy', 'contents': [{'kind': 'transaction', 'source': 'tz1RjonN5qEJM8cZhKcfGyoEqhw1FNB4ti6w', 'fee
': '1290', 'counter': '198828', 'gas_limit': '9716', 'storage_limit': '182', 'amount': '0', 'destination':
 'KT1B5uCAmStQazfTxrvk4jNDoaTFVyYKEc3h', 'parameters': {'entrypoint': 'createPoll', 'value': {'prim': 'Pai
r', 'args': [{'string': 'my_poll_id_2'}, {'prim': 'Pair', 'args': [{'string': '2021-07-07T12:19:17Z'}, {'i
nt': '4'}]}]}}}], 'signature': 'sigdcVafcKdC53Gvs5FS1WJTup1eSEesZnZ4nSzX6XhSWcS65SUAfSYQs9Lt78Nuq2q2GNLAaK
16eUdnCkJQNysSpVbAm5y2'}
>>>
```

Indexers

- the node is slow, and it can’t tell us all the keys to big maps
- so we need something which can cache the data for us
- there are several indexers for Tezos. The ones we use most often are:
- Better Call Dev https://better-call.dev/
- TzKT https://tzkt.io/
- You can inspect your contract, and its storage, see operations and so on.
- The indexers are open source, and you can run your own using docker.

Operations


ALL
SENT
RECEIVED
DELEGATIONS
Call createPoll from tz1RjonN...4ti6w
8 minutes ago
applied
transaction
Call createPoll from tz1RjonN...4ti6w
9 minutes ago
applied
transaction
Call createPoll from tz1RjonN...4ti6w
11 minutes ago
applied
transaction
Created with balance +0 ꜩ ($0.00)
31 minutes ago
applied
origination

Storage view in TzKT

Questions