Teku - Consensus layer

April, 2024

Disclaimer: The views expressed are mine and do not necessarily reflect the views of Consensys.

About me

Paul Harris

- Senior staff blockchain protocol engineer - Consensys
- Consensus-layer client developer (Teku)
- Beacon-api maintainer
- Keymanager-api maintainer

Team

- 7 software engineers, 4 in EU, 3 in APAC

At a high level

Each Consensus-layer client has a number of drivers

- Consensus-spec
- Execution-api - interface to Execution-layer
- Standard REST API’s (beacon-api, keymanager-api, builder-api)
- Client team direction (eg. Teku currently doesn’t implement lightclient)

Rules

Consensus layer(CL) and execution layer(EL) both define open specifications.

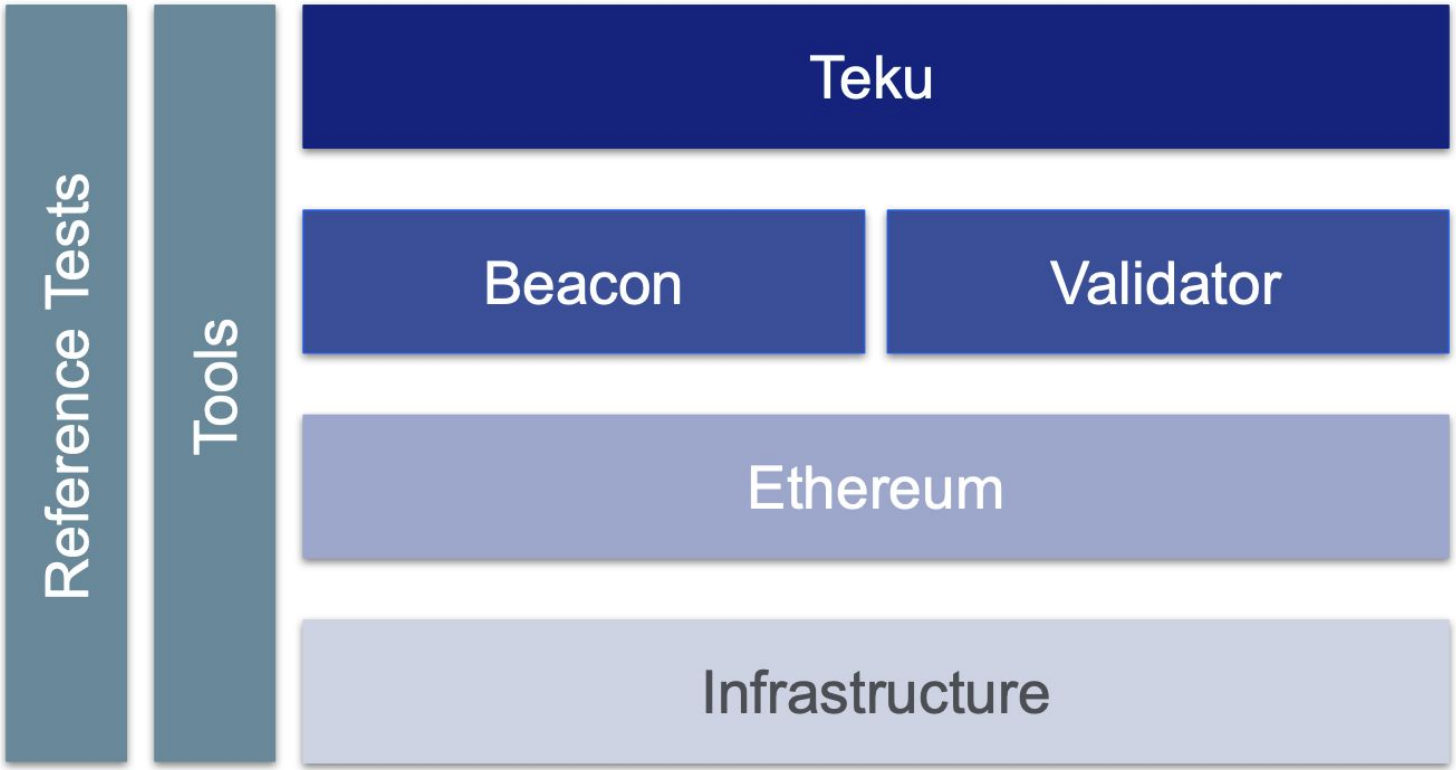
- PoS is basically a distributed game.
- Like any game,
  - interactions are fairly well defined,
  - there are rules.
- Like any complicated game, there can be grey areas
- CL uses executable specs!
  - Tests are written for the specs, and can be run on all implementations

Teku

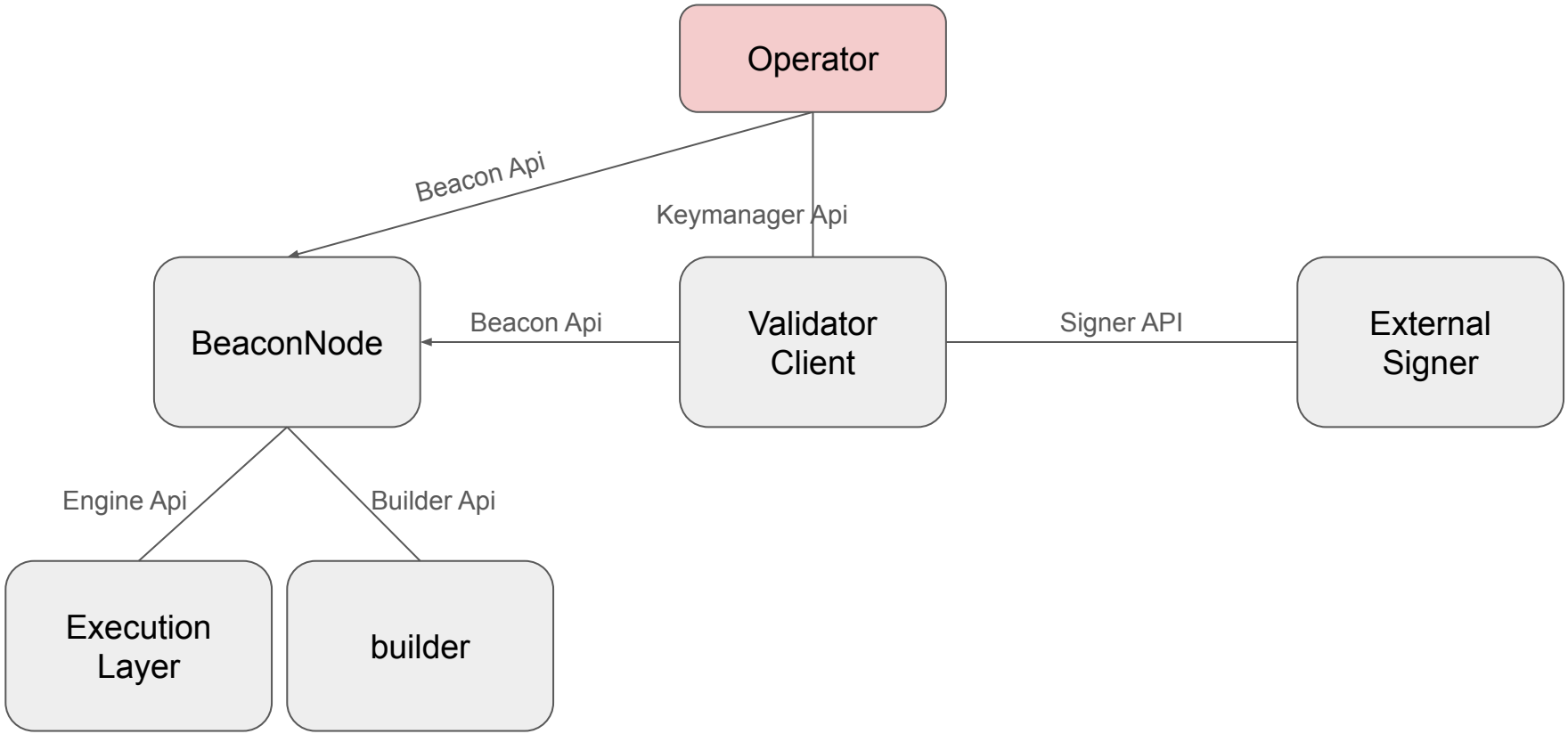
Teku wants to be the client of choice for institutional stakers.

- Written in Java
- Fairly extensive metrics (prometheus / grafana)
- Focus on testing (unit, integration, system)
- Relatively clear logging
- We maintain 1 code-stream generally (aka master development), and release frequently
- Well documented, open source, strive to offer good support
- We are node operators, and we run teku! (eating our own dog-food)

Teku Module Layout



API's


Operator
Beacon Api
Keymanager Api
BeaconNode
Beacon Api
Validator Client
Signer API
External Signer
Engine Api
Builder Api
Execution Layer
builder

API’s

- Interface to the world, and for the world to access beacon-api
- Security by default (only accessible on localhost)
- Declarative interface framework (`TypeDefinition`)

TypeDefinition allows us to define parts of our object structure to expose, and parts to not expose.

- SSZ Objects are the basis of our type definitions.

TypeDef Example

`PostAttestation` - submit a signed attestation to the beacon-node.

- Implementation
- Unit tests
- Definition test

```
EndpointMetadata.post("/eth/v1/beacon/pool/attestations")
    .operationId("submitPoolAttestations")
    .summary("Submit Attestation objects to node")
    .description(
        "Submit signed attestations to the beacon node"
      + "This endpoint does not protected against slashing.")
    .tags(TAG_BEACON, TAG_VALIDATOR_REQUIRED)
    .requestBodyType(
        DeserializableTypeDefinition.listOf(
            schemaDefinitionCache
                .getSchemaDefinition(SpecMilestone.PHASE0)
                .getAttestationSchema()
                .getJsonTypeDefinition()))
    .response(SC_OK, "Attestations received and broadcast")
    .response(
        SC_BAD_REQUEST,
        "Errors with one or more attestations",
        ErrorListBadRequest.getJsonTypeDefinition())
    .build());
```

Typedef example - GetBlockRoot

`GetBlockRoot` - get the root of a block, plus some other attributes

- Implementation | Integration Output
- Unit Test
- Documented API

```
SerializableTypeDefinition<ObjectAndMetaData<Bytes32>>  RESPONSE_TYPE =
   SerializableTypeDefinition.<ObjectAndMetaData<Bytes32>> object()
       .name("GetBlockRootResponse")
       .withField("data", ROOT_TYPE, ObjectAndMetaData::getData)
       .withField(EXECUTION_OPTIMISTIC, BOOLEAN_TYPE, ObjectAndMetaData::isExecutionOptimistic)
       .withField(FINALIZED, BOOLEAN_TYPE, ObjectAndMetaData::isFinalized)
       .build();

SerializableTypeDefinition<Bytes32>ROOT_TYPE =
    SerializableTypeDefinitionobject(Bytes32.class)
        .withField("root", BYTES32_TYPE, Function.identity())
        .build();
```

Any questions on our APIs?

EIP prototyping

Prototyping is pretty important

- Once an EIP stops being research and starts being a dev project
- Sometimes discussed on discord, sometimes telegram
- Engineering mindset vs. theory mindset
- Make an EIP actually fit into the spec, define actual EL/CL interaction
- Figure out missed bits (aka implementation details)

Example - EIP-7251

There was a desire in EIP-7251 to reduce initial penalty

- Why? Because once MAX balance scales up, initial penalty would be 64 eth…
- That seems fair, so with the reduction it approaches 0…

Throwing rocks at it, the other thing that comes into play is Whistleblower rewards.

- When slashed, the whistleblower reward happens, and we'd be paying around 4 eth if a 2048 eth validator got slashed…
  - We don't want to mint eth!

Making it fit

Ref. the _features folder for eip_7251

- Data structures needed early - BeaconState and dependents
  - https://github.com/rolfyone/teku/pull/3

Teku has a couple of object definitions currently due to tech debt, so it's quite a big change to alter the state.

'Annoying' tests to show what we've missed (mostly)

Rare use of 'branch development' in teku, usually we would use trunk development.

is_eligible_for_activation_queue

In java, we're not using snake case, so lets translate that to camel case

`isEligibleForActivationQueue`

Which is in COMMENTS in `AbstractEpochProcessor`

This at least finds the 'change', then we'd go and decide on how to reflect the spec change in our code. (draft PR)

- is_eligible_for_activation_queue
- has_compounding_withdrawal_credential
- is_compounding_withdrawal_credential
- COMPOUNDING_WITHDRAWAL_PREFIX

Planning / Ownership

Typically from a ‘team’ perspective, we allocate a project owner

- Break the project down into chunks
- Present an overview to the team
- Keep up with developments and testnets
- May or may not implement the whole feature depending on size
- Eg. withdrawals was not completely implemented by me, but I owned it and dived into the spec. I had another team member help me with sections to get it done in a timely manner.

Performance example

Slashing protection files

- Synchronized file access in `LocalSlashingProtector`

```
public synchronized SafeFuture<Boolean> maySignBlock(
      final BLSPublicKey validator,
      final Bytes32 genesisValidatorsRoot,
      final UInt64 slot)
```

- Row level locking removes the need to synchronize in `LocalSlashingProtectorConcurrentAccess`
- Feature toggled by `--validator-is-local-slashing-protection-synchronized-enabled`

How do I become a core dev?

- Get involved! Pick a project, maybe you use it or maybe not
- Familiarize yourself with how it works, look for their discord etc.
  - They will either have ‘good first issue’ issues or be able to help you find something
  - Even being in Eth RnD and following and getting up to speed
- Keep an eye on hiring pages!
- Don’t let self doubt be the only thing that holds you back!

References

https://github.com/ethereum/consensus-specs

https://github.com/ConsenSys/teku

https://ethereum-magicians.org/t/eip-7251-increase-the-max-effective-balance/15982

https://github.com/ethereum/consensus-specs/tree/dev/specs/_features/eip7251

Eth R&D discord