Quest for the Best Tests
A retrospective on #TestingTheMerge

Parithosh Jayanthi
DevOps Engineer, Ethereum Foundation
Section 1

#TestingTheMerge Assemble
Why is the Merge complicated?

- >20 client combinations need to be tested & regressions can sneak in very easily
- Specification is under active development -> Harder to track subtle differences
- Communicating and debugging various client combinations
- Figuring out how to test this in a reliable manner!
- All future upgrades will inherit some of the complexity - build once, use many
- Debug knowledge needed for ELs and CLs are quite different
WHAT COULD POSSIBLY GO WRONG?
What tests can we have?

- **Unit tests:**
  - Handled by client teams internally
  - Usually runs on every PR
  - Reduces chance of regressions

- **Integration tests:**
  - Handled partially by teams
  - Involves local testnets or interop tests
  - Ensures interop at a high level
What tests can we have?

- **System tests:**
  - Tests end-to-end functionality
  - Involves external parties and the community

- **Production tests:**
  - Tests performance on a prod-like environment
  - Public testnets involving everyone
  - Finds issues that happen only at real-world loads
Section 2

#TestingTheMerge: The Infinity War
Spec tests

- The CI runs on every commit to the specs repo, ensuring that the specs pass tests
- Client teams import the specs and test it in their local CIs as well
- Acts as a sanity check to make sure client aren’t implementing a spec that won’t pass tests
Hive tests

- Hive tests run using a simulator that starts up the clients and runs the tests against a pre-defined interface
- Acts as a integration and regression check to make sure client aren’t failing defined edge cases
- e.g: Feed a Nethermind node two terminal blocks, assert how it transitions
- Shoutout to @elbuenmayini

<table>
<thead>
<tr>
<th>Start time</th>
<th>Suite</th>
<th>Clients</th>
<th>Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-10-10T13:24:58.444Z</td>
<td>engine-api</td>
<td>nethermind</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-10T11:19:45.834Z</td>
<td>engine-api</td>
<td>go-ethereum</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-10T08:32:22.169Z</td>
<td>engine-api</td>
<td>erigon</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-10T04:19:59.077Z</td>
<td>engine-api</td>
<td>nethermind</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-10T03:25:14.198Z</td>
<td>engine-api</td>
<td>go-ethereum</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-09T20:19:15.930Z</td>
<td>engine-api</td>
<td>nethermind</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-09T16:35:01.964Z</td>
<td>engine-api</td>
<td>erigon</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-09T11:41:45.059Z</td>
<td>engine-api</td>
<td>nethermind</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-09T10:46:59.564Z</td>
<td>engine-api</td>
<td>go-ethereum</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-09T07:59:43.300Z</td>
<td>engine-api</td>
<td>erigon</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-09T03:05:33.754Z</td>
<td>engine-api</td>
<td>nethermind</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-09T02:10:57.615Z</td>
<td>engine-api</td>
<td>go-ethereum</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-08T18:25:21.055Z</td>
<td>engine-api</td>
<td>nethermind</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-08T17:30:35.247Z</td>
<td>engine-api</td>
<td>go-ethereum</td>
<td>✓</td>
</tr>
<tr>
<td>2022-10-10T10:24:07.968Z</td>
<td>eth2-testnet</td>
<td>teku-vc, go-ethereum, teku-bn</td>
<td>✓ (2)</td>
</tr>
</tbody>
</table>
Kurtosis tests

- Kurtosis spins up a local testnet with the required EL/CL combinations and then allows them to transition/merge. It then asserts some “happy case” conditions.
- An integration test make sure client are compatible.
- Useful to rapidly iterate ideas.
- e.g: Are blocks being produced, are there tx’s...
Sync tests

- The sync test co-ordinator spins up every client combination daily and syncs to head on various testnets. Both genesis sync as well as Checkpoint sync are performed.
- Edge case sync tests are also performed: EL down, CL down, etc
- Acts as an integration test to make sure users can always sync the network
- Shoutout to @samcmAU

test:
  name: "basic"
  tasks:
    - name: run_command
      config:
        command:
          - "echo"
          - "hello!"
    - name: execution_is_healthy
    - name: consensus_is_healthy
    - name: both_are_synced
      config:
        consensus:
          percent: 100
        execution:
          percent: 100
      - name: run_command
        config:
          command:
            - "echo"
            - "done!"
  execution:
    url: http://localhost:8545
  consensus:
    url: http://localhost:5052
Shadow Forks & Testnets

- Allows us to check compatibility across all clients through the entire lifecycle
- Fresh testnets allow us to check assumptions across client pairs without much overhead
- Shadow forks allow us to stress test the clients with real state and transaction load
- We can invite participants in a controller manner to take part in the tests
- Acts as release test which triggers real world edge cases, before we recommend the releases to the general public
Antithesis & Fuzzers

- Antithesis offers a deterministic hypervisor which allows us to perform network splits, packet loss while fuzzing clients. The deterministic hypervisor allows us to re-trigger the issue, allowing for capturing the state of the client and easier debugging.
- Various fuzzers are run against different layers of the stack to find bugs.
- These bugs also allow us to re-evaluate if changes need to be made in the specs or if the bug is an implementation level issue.
So what did we still miss?

- In-memory database too low to process mainnet blocks
- Non-optimal block production: Random production of 0/few tx blocks
- Multiple terminal blocks (in a specific condition) caused missing receipts and caused failed proposals
- Lots of constant syncing nodes on mainnet led to unexpected performance degradation when compared to shadow forks
- Failover beacon node scenario -> some requests sent just to the primary
What can we reuse?

Running testnets helped show us tooling blind spots in the DevOps ecosystem:

- Metrics exporter: [https://github.com/samcm/ethereum-metrics-exporter](https://github.com/samcm/ethereum-metrics-exporter)
- Sync testing: [https://github.com/samcm/ethereum-sync-testing/actions](https://github.com/samcm/ethereum-sync-testing/actions)
- Client automation: [https://github.com/ethPandaOps/ethereum-helm-charts](https://github.com/ethPandaOps/ethereum-helm-charts)
- Scalable testnets: [https://github.com/ethPandaOps/ethereum-k8s-testnets](https://github.com/ethPandaOps/ethereum-k8s-testnets)
- Easy testnets: [https://github.com/ethereum/consensus-deployment-ansible](https://github.com/ethereum/consensus-deployment-ansible)
- Faucet: [https://github.com/komputing/FaucETH](https://github.com/komputing/FaucETH)
- Checkpoint Sync Provider: [https://github.com/samcm/checkpointz](https://github.com/samcm/checkpointz)
- PRs to explorers, validator key generation tools, load balancer
If you want to join the testing efforts contact mario.vega@ethereum.org
Thank you!
Join #TestingThe{Surge,Verge,Purge}!

Parithosh Jayanthi
DevOps Engineer
parithosh@ethereum.org

@parithosh_j