The demo showcases end-to-end operation of zkLLVM-based oracle+contract computing Lido's total value locked, active and exited validators in a controlled environment - the end-to-end script have complete control of both blockchain "passing of time" (blocks and slots), Consensus Layer state (aka BeaconState), and oracle invocation time (through invoking the oracle as a script).

During preparatory stage, the end-to-end script performs the following actions:

- Deploy all necessary contracts, including
    - The TVL contract itself
    - zkLLMV verifier contract (aka PlaceholderVerifier)
    - ZK-proof verification circuit compiled into verification gates.
    - Verifier Helper, providing circuit verification invocation parameters.
- Set up initial BeaconState
- Launch Consensus Layer stub API

The script contains 4 cases, each with a different BeaconState, and checking different conditions in the TVL contract:

- Happy path- report and proof are correct, report is accepted
    - Happens 2 times - with an initial and updated BeaconState
- Proof verification - report is correct, but proof do not verify; report is rejected
- Withdrawal credentials check - report contains wrong withdrawal credentials and is rejected
- Balances merkle tree root check - report contains wrong balances merkle tree root and is rejected

Step 1 - "happy path": Initial state, invoking oracle and accepting report
In step one, the script starts with the initial state of the chain. For simplicity, we're using a "synthetic" BeaconState that consists of only 5 Lido validators and 5 non-Lido validators, all in active state and initial balance of 10Gwei.

Oracle performs necessary computations (obtaining BeaconState from the Consensus Layer stub API), and submits the report. TVL contract performs all the checks (incl. verifying the proof), and accepts the report. As a final step, we read the stored report from the TVL contract and assert it is equal to expected values (5 active Lido validators, 0 exited ones, and 50Gwei total balance)

Step 2 - "fake proof": submitting a fake proof, rejecting the report
In this step, the end-to-end script **does not** invoke the oracle, and actually submits a "fake" report, containing a well-formed, but invalid proof.

TVL contract rejects the report, stating the rejection reason as "ZK proof did not verify". Additionally, we check that the report stored in TVL contract is (a) not equal to the submitted one, and (b) is equal to the previously submitted one - i.e. that no update happened.

Step 3 - "wrong withdrawal credentials": submitting an incorrect report, rejecting it
In this step, the end-to-end script verifies that TVL oracle would reject the report if incorrect withdrawal credentials are passed. TVL contract obtains valid withdrawal credentials from Lido Staking Router (that is the source of truth for withdrawal credentials).

The script **do not** invoke the oracle and instead sends a “handcrafted” report with withdrawal credentials **not** matching the actual ones. TVL contract rejects the report, and we verify it the same way as in step 2 - reading the stored report from the contract and asserting it is still equal to the report from step 1.

Step 4.1 - “wrong balances hash”: submitting the report with handcrafted balances merkle tree root

This step showcases the protection from tampering with balances after the proof has been obtained. The oracle is again **not invoked**, and instead we’re handcrafting a report with incorrect balances hash, while using a correct (precomputed) proof that would have passed the verification if balances were not tampered with.

TVL contract obtains balances hash from a trusted on-chain source (separate oracle+contract), and rejects the report since the hashes do not match. As usual, we check that the report was not updated by comparing the stored report with the report from step 1.

Step 4.2 - “happy path vol. 2”: running the oracle on the updated Beacon State, report is accepted

Final step demonstrates that oracle computes the correct report on the updated Beacon state, and the report is accepted by the TVL contract.

The BeaconState has been modified on each step since step 1, the current BeaconState and difference from the initial one are printed to emphasize the difference - both in terms of changed balances and changing validator state.

The end-to-end script invokes oracle, which computes the report and sends it to the TVL contract. Similarly to step 1, TVL contract performs the necessary checks and accepts the report.