Materializer

IPC Spec - Materializer

This document gives an introduction and the rationale for the Materializer, which is a tool that helps provision entire IPC subnet hierarchies, mainly for testing purposes.

Hierarchical Deployment

Deploying IPC subnets has traditionally been a laborious process which involved running scripts manually and copying values from the output of one and passing it as input to the next. The steps involved are described in theusage document.

What complicates matters is that variables aren't known up front, for example we have to create a subnet to learn its ID which we have to use in all subsequent commands, and continue recursively until the entire hierarchy is established. We have to fund accounts on subnets, join with validators, wait for the subnet to activate, then start subnet nodes.

An early attempt to help with the creation oftest environments was the creation of topologies such as thisexample which were compiled to aseries of steps in a script to provision a hierarchy. The steps were executed withmake and left behind enough artifacts to ensure that repeated execution was idempotent.

An evolution of this idea is the Fendermintmaterializer , which is available as aCLI command under the mainfendermint entrypoint.

Manifests

The materializer uses themanifest to describe entire hierarchies. This is typically a nested, recursive YAML or JSON document with some examples used in tests availablehere .

The entities in the manifest are generally organised into maps, so that they have a unique logical ID within their parent context. By logical ID we mean that it's just a static string label, e.g.alice , rather than something that we'll only know once we deployed something.

At the root of the manifest we have the following properties:

- accounts
- is technically a map, but really is just a set of logical account identifiers such asalice
- andbob
- . These IDs can be used throughout the manifest whenever we refer to an account, instead of public keys or FVM addresses. That is so we can provide a static description of the subnet, and create cryptographic keys later, which can get unpredictable FVM actor IDs when we use them on a chain.
- rootnet
- describes the L1, which can have one of the following types:
-
  - New
-
  - means we will provision the L1 chain ourselves, in which case we have to define:
-
  -
    - validators
-
  -
    - containing a mapping from account ID to collateral (voting power)
-
  -
    - balances
-
  -
    - listing the initial token balance for every account on the L1 (assumed to be Ethereum accounts because that's how we interact with IPC).
-
  -
    - nodes
-
  -
    - is the list of physical nodes to create to run the L1
-
  -
    - env
-

  -
    - contains custom environment variables passed to all nodes

---

-
  - External
-
  - means we will use an existing L1 such as Calibration net:
-
  -
    - chain_id
-
  -
    - is the numerical ID of the L1 chain
-
  -
    - deployment
-
  -
    - describes how we’ll get the IPC stack:
-
  -
    -
      - New
-
  -
    -
      - means we have to deploy it from scratch using the givendeployer
-
  -
    -
      - account, which must have balance on the L1
-
  -
    -
      - Existing
-
  -
    -
      - means it’s already deployed, and we just have to give the Ethereum addresses of thegateway
-
  -
    -
      - andregistry
-
  -
    -
      - actors

-
  -
    - urls
-
  -
    - contains a list of RPC addresses of nodes on the L1 where we can send queries and transactions

---

- *
- subnets
- is a map which contains the IPC subnets to be created on the L1 (we currently do not support connecting to an existing subnet):
-
  - creator
-
  - is the logical ID of the account to be used to create the subnet, ie. to send the transaction invoking theregistry
-
  - contract on the L1
-

  - validators
-
  - is a map with the collateral each validator in this subnet willjoin
-
  - with (for this they must have the balance on the L1; currently the materializer only supports PoS, not PoA)
-
  - balances
-
  - is a map with the amount of tokens that accounts willfund
-
  - the subnet with
-
  - nodes
-
  - lists the physical machines that will run this subnet
-
  - relayers
-
  - defines the how information flows bottom-up:
-
  -
    - submitter
-
  -
    - is the account that sends the checkpoint as a transaction to the parent network
-
  -
    - follow_node
-
  -
    - is the ID of the node which the relayer will query for changes
-
  -
    - submit_node
-
  -
    - is either a URL or the ID of the parent node to submit transactions to; URLs are used when we have an external rootnet (in which case there are no nodes in the manifest), while IDs work with new rootnets that are run by nodes we defined

---

-
  - bottom_up_checkpoint.period
-
  - defines the frequency of checkpoints on the subnet
-
  - env
-
  - is a list of custom environment variables all nodes get
-
  - subnets
-
  - recursively defines nested subnets
- *
-

Thenodes in the manifest have the following properties:

- mode
- defines whether we have aValidator
- or aFull
- node. The former has avalidator
- which is must be of the accounts, the latter just follows others without producing blocks.
- ethapi
- indicates whether the Ethereum API facade should run on this node
- seed_nodes
- is a list of node IDsin this subnet
- to bootstrap from - nodes can mutually seed from each other
- parent_node

- is only used on subnets, not on the rootnet; it can be a URL or a node ID depending on whether the parent subnet is an external rootnet, or consists of nodes defined in the manifest. It defines which node the top-down sync follows.
-

Transitive Balances

The balances and collaterals in the manifest are given inatto , which is is 1/10**18 of a FIL token.

The subnet balances describe the desired state after the whole hierarchy has been provisioned, not when a particular subnet is created. That means that the balances of nested subnets are not subtracted from the balances of ancestor subnets, but rather brought in recursively from the rootnet balance. For example if we say we wantalice to be a validator in subnet/root/foo/bar with 100 collateral, then we don't have to listalice with a balance of 100 in/root/foo ; we just have to make surealice has the necessary starting balance in/root and the necessaryfund transactions will be issued to move the funds from/root to/root/foo and thenjoin the/root/foo/bar subnet there.

Materializers

The materializer figures out the steps necessary to provision the subnet hierarchy and execute them while leaving behind a trail of artifacts on the file system that allows it to be idempotent. For example:

- every account has their own directory where the generated secret/public key, FVM and EVM addresses are stored in various formats; if the directory exists creating an account is skipped
- every subnet has their own directory where the subnet ID is written to a file; if it exists, we don't have to create the subnet again
- every node has their own directory where the database files and logs are mounted
-

The logic of which steps to execute is contained in thetestnet module, and depends purely on the contents of the manifest. Where wecan have different strategies is how to physically materialize the nodes and subnets:

- We can provision everything as local docker containers
- We could remotely provision instances on cloud infrastructure
- We could run multiple instances in-memory on different ports and override their behaviour in tests
-

The only currently available implementation isusing docker to provision local containers, potentially connecting to an external rootnet. This is used by thesetup command.

We can also implement materializers that do not actually provision resources:

- Visit the manifest and validate that accounts exist, that balances are feasible
- Log actions before forwarding them to another materializer
- Print CLI commands that the operator could execute in the terminal
-

The combination of the first two is how thevalidate command works.

Use Cases

Integration testing

The integration testsuse a docker materializer to instantiate testnets on the fly and run assertions against specific nodes. The tests are organised intomodules , according to whichtest manifest they use.

The machinery available in the tests try to make it easy to connect to specific nodes, so a test can look for example like this:

```

Copy letnode_id=testnet.root().node("node-2"); letnode=testnet.node(&node_id)?;

letprovider=node .ethapi_http_provider()? .ok_or_else(||anyhow!("node-2 has ethapi enabled"))?;

letbn=provider.get_block_number().await?;

ifbn<=U64::one() { bail!("expected node-2 to sync with node-1"); }

```

Connecting to Calibration

For debugging purposes it is possible to use the materializer to provision a subnet on Calibration net and then stop the local containers, and use the artifacts left behind to run a node through the IDE with a debugger.

The materializer CLI has afendermint materializer import-key command to import some secret key that already exists on an external L1 (funded through the faucet) to be associated with a logical account ID in the manifest.

For example the following manifest was used to create a local stand-alone node to run a subnet we create on Calibration net:

- External Standalone Manifest

-
  ```
- Copy
- accounts:
- fridrik:{}
- rootnet:
- type:External
- chain_id:314159
- deployment:
- type:Existing
- gateway:0x6d25fbFac9e6215E03C687E54F7c74f489949EaF
- registry:0xc938B2B862d4Ef9896E641b3f1269DabFB2D2103
- urls:
- -https://api.calibration.node.glif.io/rpc/v1
- subnets:
- iceland:
- creator:fridrik
- validators:
- fridrik:'1000000000'
- balances:
- fridrik:'20000000000000000000'
- bottom_up_checkpoint:
- period:100
- relayers:
- nodes:
- moso:
- mode:
- type:Validator
- validator:fridrik
- parent_node:"https://api.calibration.node.glif.io/rpc/v1"
- seed_nodes:[]
- ethapi:true
- ```
-

The following commands provision the subnet locally:

```

Copy cargo run-q-p fendermint_app--release--\ materializer--data-dir PWD/testing/materializer/tests/docker-materializer-data \ import-key --manifest-file ./testing/materializer/tests/manifests/external.yaml \ --account-id fridrik --secret-key ~/.ipc/validator_0.sk

cargo run -q -p fendermint_app --release -- \ materializer --data-dir PWD/testing/materializer/tests/docker-materializer-data \ setup --manifest-file ./testing/materializer/tests/manifests/external.yaml

```
```