Understanding the testnet installation

Date	
Slides	
↗ 📓 Courses	IDP
Lecture video link	

Types of subnet

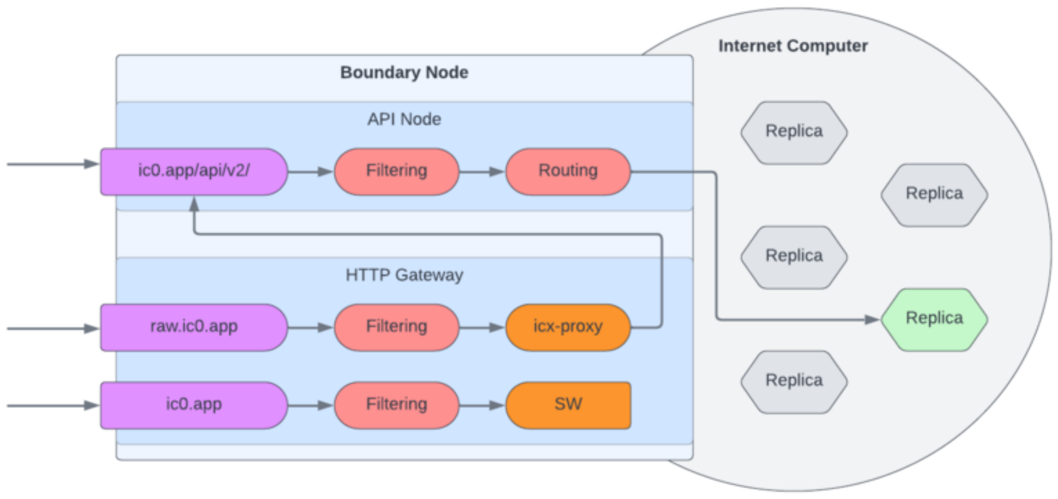
There are two types of subnet:

- `system` subnets: These subnets are reserved for canisters that are an integral part of the Internet Computer. Typically, canisters on these subnets are controlled by the NNS and they don't pay cycles. Users cannot deploy canisters on those subnets.
- `application` subnets: These are the default subnets that users can deploy canisters to. They typically have a size of 13 nodes and canisters on them have to pay cycles. If a user does not provide any specific requirements a random application subnet is chosen as the destination to create the canister.

Types of nodes

Nodes:

- **The boundary nodes** are the gateway to the Internet Computer (IC), which allow users to seamlessly access the canister smart contracts running on it: through stock browsers users are served web-content fully from the chain. To this end, the boundary nodes translate the users' requests into API canister calls, route these calls to the subnet running the canister and load balance among the subnet's replica nodes. In addition, boundary nodes provide caching and other services for improved performance.



src: https://wiki.internetcomputer.org/wiki/Boundary_Nodes

NNS and what is it built up of

The Network Nervous System of the Internet Computer is realized by a set of *canisters*. NNS canisters include:

1. **Ledger canister:** The ledger canister stores the ICP utility *token balance* of each principal and the history of ICP *transactions*.
2. **Governance canister:** The governance canister receives and stores *Proposals*, which are suggestions for how the Internet Computer should be changed. These proposals can then be voted on. The governance canister also tracks *Neurons*, which determine who is allowed to participate in governance.
3. **Registry canister:** The registry canister stores the configuration of the whole Internet Computer, e.g., which nodes belong to a certain subnet and the software each node should run.
4. **Cycles minting canister**: This canister is responsible for minting *cycles*, the fuel for canisters for computation, communication and storage. New cycles can be minted when a new canister is newly created or when an existing canister is topped up with additional cycles.
5. **Root canister**: The root canister is the controller of all other NNS canisters and responsible for upgrading them.
6. **Lifeline canister**: The lifeline canister is the controller of the root canister and responsible for upgrading it.
7. **Archive canisters**: The canisters that store the history of the ledger transactions once there are too many transactions to keep in a single canister.
8. **Genesis token canister:** This is the canister that was used to initialize the neurons that already existed during genesis.

The canisters that users of the Internet Computer are interacting with the most are the first two: the ledger canister for making transactions, and the governance canister for staking tokens and submitting and voting on proposals.

src: https://wiki.internetcomputer.org/wiki/NNS_Canisters

Installation process for testnet

What is happening:

1. **icos_deploy.sh**
   a. uses the inventory.py to generate the dynamic mapping of node to their ipv6 addressess.
   b. creates USB sticks for IC Nodes.
   c. creates USB sticks for boundary nodes
   d. `ic_network_redeploy` playbook with `ic_state=create`
   e. this playbook gives the role of `ic_guest`
2. **ic_guest**

   The `main.yml`

```
- name: import tasks prepare
  import_tasks: "prepare.yml"
  when: ic_state == "create"
  tags: [ "ic_guest" ]

- name: import tasks disk_pull
  import_tasks: "disk_pull.yml"
  when: ic_state == "create" and (ic_disk_path | length == 0)
  tags: [ "ic_guest" ]

- name: import tasks disk_push
```

```
  import_tasks: "disk_push.yml"
  when: ic_state == "create" and (ic_disk_path | length > 0)
  tags: [ "ic_guest" ]

- name: import tasks disk_push
  import_tasks: "aux_disk_push.yml"
  when: ic_state == "create" and ic_disk_path
  tags: [ "ic_guest" ]

- name: import tasks media_pull
  import_tasks: "media_pull.yml"
  when: ic_state == "create" and (ic_media_path | length == 0)
  tags: [ "ic_guest" ]

- name: import tasks media_push
  import_tasks: "media_push.yml"
  when: ic_state == "create" and (ic_media_path | length > 0)
  tags: [ "ic_guest" ]

- name: import tasks
  import_tasks: "create.yml"
  when: ic_state == "create"
  tags: [ "ic_guest" ]
```

1. **prepare.yml:** It creates sets the output of the commands, creates some folders and install `GNU parallel` and `zstd`
2. **disk_pull.yml:**

   This Ansible task is quite complex. It downloads disk images for different node types from specified URL. There are two options:

   - checking from a proxy server and
   - falling back to a content delivery network (CDN) if that fails.

   > **Replica**:
   >
   > - Proxy: `http://download.proxy-global.dfinity.network:8080/ic/d53b551dc677a82c8420a939b5fee2d38f6f1e8b/guest-os/disk-img-dev`
   > - CDN: `https://download.dfinity.systems/ic/d53b551dc677a82c8420a939b5fee2d38f6f1e8b/guest-os/disk-img-dev`
   >
   > 1. **Auxiliary (Aux)**:
   >   - Proxy: `http://download.proxy-global.dfinity.network:8080/farm/universal-vm/ca2ddfab45f940564503e2edf3d2c02acc05988edde4e3a7400355bd22d69d44/x86_64-linux`
   >   - CDN: `https://download.dfinity.systems/farm/universal-vm/ca2ddfab45f940564503e2edf3d2c02acc05988edde4e3a7400355bd22d69d44/x86_64-linux`
   > 2. **Boundary**:
   >   - Proxy: `http://download.proxy-global.dfinity.network:8080/ic/d53b551dc677a82c8420a939b5fee2d38f6f1e8b/boundary-os/disk-img-dev`
   >   - CDN: `https://download.dfinity.systems/ic/d53b551dc677a82c8420a939b5fee2d38f6f1e8b/boundary-os/disk-img-dev`

> 💡 I tried it to curl it and it didn't work. The proxy timed out and the cdn gives an unauthorised error

   Then the downloaded disks are unarchived

3. **disk_push.yml**

   It does the following:

   1. **Remove existing disk-img.tar.zst**: This is done to ensure a clean slate before creating a new archive.
   2. **Archive disk.img**: This task creates the archive file in the `ic_disk_path` directory.
   3. **Synchronize disk-img.tar.zst**: This task uses the Ansible `synchronize` module to copy the newly created `disk-img.tar.zst` from the source directory to the destination directory on the remote host(s).
   4. **Unarchive file disk-img.tar.zst**: This task decompresses the `disk-img.tar.zst` file into the `/var/local/ic/disk` directory on the remote host(s).

4. **aux_disk_push.yml:** This does the same tasks as disk_push but pushes the disc file to the aux folder
5. **media_pull.yml:** Debug message for CI/CD Pipelines
6. **media_push.yml:** The Ansible tasks are responsible for copying the disk image files (`media.img`) to the remote hosts
7. **create.yml:**

   a. copy file media.img

   b. copy file disk.img for replica

   c. copy file media.img for boundary node VMs

   d. copy file disk.img for boundary nodes

   e. copy file disk.img for aux nodes

   f. Prepare the Guest template file

   g. Define (create) a guest

   h. Check if dfinity-hsm-agent service exist

   i. Stop the dfinity-hsm-agent.service

   j. Ensure potentially conflicting kernel modules are not loaded

3. Once this is done, it goes back to `icos_deploy` and starts the playbook `icos_network_redeploy.yml` with the `ic_state=start` . It again has the role `ic-guest` and then executes the `start.yml`

   This Ansible task is used to start a virtual machine guest and set it to autostart using the `virsh` command, which is a command-line interface tool for managing guest operating systems and hypervisor.

4. Once this is done, the NNS canisters are installed. It goes back to `icos_deploy` and starts the playbook `icos_network_redeploy.yml` with the `ic_state=install` . It again has the role `ic-guest` and then executes the `start.yml`

NNS Installation process

1. **Wait for replica to listen on all NNS nodes on port 8080:** This task waits for the application to start listening on port 8080 for all nodes defined in the 'nns' group. It uses an external task file named `url_waitfor_connect.yml`
2. **Check if the initial neuron config exists:** The `stat` module is used here to check if a file (`initial-neurons.csv`) exists in the same directory as the inventory file being used by Ansible. The next task sets the value of `initial_neurons` to the path of the `initial-neurons.csv` file
3. **Get Custom NNS canisters:**This task only executes if the `custom_canister_dir` variable is defined. Copies the custom wasm file in to the `canister` folder
4. **Install NNS Canisters:** performs a series of operations using shell commands to install and verify the installation of NNS (Neuron Network Service) canisters.

a. The task fetches the node name and the IPv6 address of the first node in the `nns` group and create the NNS URL.

b. It checks whether an NNS subnet is already installed, and if not, it ensures that it can reach the replica via a curl command. If the curl command fails, it prints an error message and exits.

> NOTE: whenever they are trying to curl something, they are always using IPV6 address in order to do that

```
NODE0_NAME={{ groups['nns'][0] }}
NODE0_IPV6={{ hostvars[groups['nns'][0]].ipv6_address }}
NNS_URL="http://[$NODE0_IPV6]:8080"

echo "Running on $(hostname -f)"

if ! "{{ ic_media_path }}/bin/ic-admin" --nns-url http://[$NODE0_IPV6]:8080 \
    get-subnet 0 | grep -q 'subnet_type'; then
```

c. If the NNS subnet is not already installed, it begins the installation process. It first sets the `GOVERNANCE_PB` variable, which stores the initial **governance protobuf** file.

   it can be found here `rs/nns/governance/proto/ic_nns_governance/pb/v1/governance.proto`

   it stores all the information about the nns.

d. Then it invokes the `ic-nns-init` binary (corresponding to the folder in `rs/nns/init` where this binary is being built using rust), which is a tool to install and initialize NNS canisters on a subnet. This script is given a series of parameters, including the URL to the NNS, the path to the local registry store directory, the path to governance protobuf file if defined, initial neuron configuration, test ledger accounts, HSM details (if HSM usage is enabled), and the path to wasm directory for canisters.

e. After the initialization, it verifies the NNS installation by checking the subnet again.

The task is executed on the localhost (`delegate_to: localhost`), meaning the machine from which Ansible is run.

Inference

- They directly initialise the NNS from the source code. From the source code, they get the wasm for the other canister.
- There is a strong dependency on the IPV6 address in order to complete the installation.
- Boundary nodes use host- and guest-OS architecture, which is based on standard Ubuntu and often referred to as “IC-OS”. this is what is being pulled from their server and being installed and ran on the these nodes. This is something that we would need to run on our nodes as well.

Open questions?

Q: where is this `initial-governance.pb` file comes from?

   `rs/nns/governance/proto/ic_nns_governance/pb/v1/governance.proto`

Q: how are all the other NNS canisters installed?

   has to be figured out from `ic-nns-init`

play scenario → nodes, network, stacks → Roles → run that on all the nodes.