DFINITY
技术分享交流会

2月16日 18:30

分享嘉宾：
Liquan 18:30-19:30
分享主题：
NNS Registry Canister

分享嘉宾：
PYD 19:30-20:30
分享主题：
Internet Computer Consensus Protocol

自由交流 20:30 -21:00

INTERNET COMPUTER

2月16日
18:30

NNS Registry Canister

从 NNS Registry 获取信息

MIXLABS

背景

- 整个 Internet Computer 由很多相对独立的子网组成。这个创新解决了区块链的扩容问题。但是也引入了一个新的问题
- 子网之间怎样协调才能把所有子网整合成统一的平台。
  - 比如怎样才能成为一个新的子网，生成 Canister 的时候怎么保证 Canister Id 在所有子网中全局唯一，某个 Canister 调用其他 Canister 的时候，怎样知道对应的 Canister 在哪个子网等。
  - 有个特殊的子网，NNS 子网，里面运行着 10 个 Canister 来协调上面说的这些事情。
  - 首先是通过治理，所有人通过在 Governance canister 投票，投票后的结果通过 Governance 写入 Registry。
  - Registry 作为存全局信息的地方，其他子网从 Registry 这里获取信息。
- 先讲能够获取到哪些信息，然后再演示怎么获取的。

MIXLABS

获取 XDR 价格

- Cycles 都是通过 Cycles-minting Canister mint 出来的。
- Cycles-minting Canister 在 mint Cycles 时就需要这个参数

rs > protobuf > gen > registry > registry.conversion_rate.v1.rs > ...

```rust
/// Instruct the NNS about the market value of 1 ICP measured in IMF SDR.
#[derive(serde::Serialize, serde::Deserialize)]
#[derive(Clone, PartialEq, ::prost::Message)]
pub struct IcpXdrConversionRateRecord {
    /// The time for which the market data was queried, expressed in UNIX epoch
    /// time in seconds.
    #[prost(uint64, tag="1")]
    pub timestamp_seconds: u64,
    /// The number of 10,000ths of IMF SDR (currency code XDR) that corresponds to
    /// 1 ICP. This value reflects the current market price of one ICP token.
    /// In other words, this value specifies the ICP/XDR conversion rate to four
    /// decimal places.
    #[prost(uint64, tag="3")]
    pub xdr_permyriad_per_icp: u64,
}
```

MIXLABS

获取子网 Canister 范围

- 调用 Canister 的时候，需要先知道 Canister 所在的子网。然后再把调用请求发送过去。

rs > protobuf > gen > registry > registry.routing_table.v1.rs > ...

```rust
/// Represents a closed range of canister ids.
#[derive(serde::Serialize, serde::Deserialize)]
#[derive(Clone, PartialEq, ::prost::Message)]
pub struct CanisterIdRange {
    #[prost(message, optional, tag="3")]
    pub start_canister_id: ::core::option::Option<super::super::super::types::v1::CanisterId>,
    #[prost(message, optional, tag="4")]
    pub end_canister_id: ::core::option::Option<super::super::super::types::v1::CanisterId>,
}
```

MIXLABS

获取子网 Canister 范围

- 拿下来的结果，解码之后

```
routing_table_record:
================================================================================
error: None version: 28145

subnet id: "tdb26-jop6k-aogll-7ltgs-eruif-6kk7m-qpktf-gdiqx-mxtrf-vb5e6-eqe"
start PrincipalId: PrincipalId { raw: [0, 0, 0, 0, 0, 0, 0, 0, 1, 1] }
start PrincipalId in Text: rwlgt-iiaaa-aaaaa-aaaaa-cai
start PrincipalId from u64: 0
end PrincipalId: PrincipalId { raw: [0, 0, 0, 0, 0, 15, 255, 255, 1, 1] }
end PrincipalId in Text: n5n4y-3aaaa-aaaaa-p777q-cai
end PrincipalId from u64: 1048575

subnet id: "snjp4-xlbw4-mnbog-ddwy6-6ckfd-2w5a2-eipqo-7l436-pxqkh-l6fuv-vae"
start PrincipalId: PrincipalId { raw: [0, 0, 0, 0, 0, 16, 0, 0, 1, 1] }
start PrincipalId in Text: 5v3p4-iyaaa-aaaaa-qaaaa-cai
start PrincipalId from u64: 1048576
end PrincipalId: PrincipalId { raw: [0, 0, 0, 0, 0, 31, 255, 255, 1, 1] }
end PrincipalId in Text: b65vx-3qaaa-aaaaa-7777q-cai
end PrincipalId from u64: 2097151

subnet id: "qxesv-zoxpm-vc64m-zxguk-5sj74-35vrb-tbgwg-pcird-5gr26-62oxl-cae"
start PrincipalId: PrincipalId { raw: [0, 0, 0, 0, 0, 32, 0, 0, 1, 1] }
start PrincipalId in Text: jrlun-jiaaa-aaaab-aaaaa-cai
start PrincipalId from u64: 2097152
end PrincipalId: PrincipalId { raw: [0, 0, 0, 0, 0, 47, 255, 255, 1, 1] }
end PrincipalId in Text: v2nog-2aaaa-aaaab-p777q-cai
end PrincipalId from u64: 3145727
```

MIXLABS

获取子网列表

- 所有子网 ID 的列表：

```rust
/// A list of subnet ids of all subnets present in this instance of the IC.
#[derive(serde::Serialize, serde::Deserialize)]
#[derive(Clone, PartialEq, ::prost::Message)]
pub struct SubnetListRecord {
    #[prost(bytes="vec", repeated, tag="2")]
    pub subnets: ::prost::alloc::vec::Vec<::prost::alloc::vec::Vec<u8>>,
}
```

```
[tdb26-jop6k-aogll-7ltgs-eruif-6kk7m-qpktf-gdiqx-mxtrf-vb5e6-eqe, snjp4-xlbw4-mnbog-ddwy6-6ckfd-2w5a2-eipqo-7l436-pxqkh-l6fuv-vae, qxesv-zoxpm-vc64m-zxguk-5sj74-35vrb-tbgwg-pcird-5gr26-62oxl-cae, pae4o-o6dxf-xki7q-ezclx-znyd6-fnk6w-vkv5z-5lfwh-xym2i-otrrw-fqe, 4zbus-z2bmt-ilreg-xakz4-6tyre-hsqj4-slb4g-zjwqo-snjcc-iqphi-3qe, w4asl-4nmyj-qnr7c-6cqq4-tkwmt-o26di-iupkq-vx4kt-asbrx-jzuxh-4ae, io67a-2jmkw-zup3h-snbwi-g6a5n-rm5dn-b6png-lvdpl-nqnto-yih6l-gqe, 5kdm2-62fc6-fwnja-hutkz-ycsnm-4z33i-woh43-4cenu-ev7mi-gii6t-4ae, shefu-t3kr5-t5q3w-mqmdq-jabyv-vyvtf-cyyey-3kmo4-toyln-emubw-4qe, ejbmu-grnam-gk6ol-6irwa-htwoj-7ihfl-goimw-hlnvh-abms4-47v2e-zqe, eq6en-6jqla-fbu5s-daskr-h6hx2-376n5-iqabl-qgrng-gfqmv-n3yjr-mqe, csyj4-zmann-ys6ge-3kzi6-onexi-obayx-2fvak-zersm-euci4-6pslt-lae, lspz2-jx4pu-k3e7p-znm7j-q4yum-ork6e-6w4q6-pijwq-znehu-4jabe-kqe, lhg73-sax6z-2zank-6oer2-575lz-zgbxx-ptudx-5korm-fy7we-kh4hl-pqe, gmq5v-hbozq-uui6y-o55wc-ihop3-562wb-3qspg-nnijg-npqp5-he3cj-3ae, pjljw-kztyl-46ud4-ofrj6-nzkhm-3n4nt-wi3jt-ypmav-ijqkt-gjf66-uae, brlsh-zidhj-3yy3e-6vqbz-7xnih-xeq2l-as5oc-g32c4-i5pdn-2wwof-oae, mpubz-g52jc-grhjo-5oze5-qcj74-sex34-omprz-ivnsm-qvvhr-rfzpv-vae, qdvhd-os4o2-zzrdw-xrcv4-gljou-eztdp-bj326-e6jgr-tkhuc-ql6v2-yqe, jtdsg-3h6gi-hs7o5-z2soi-43w3z-soyl3-ajnp3-ekni5-sw553-5kw67-nqe, k44fs-gm4pv-afozh-rs7zw-cg32n-u7xov-xqyx3-2pw5q-eucnu-cosd4-uqe, opn46-zyspe-hhmyp-4zu6u-7sbrh-dok77-m7dch-im62f-vyimr-a3n2c-4ae, 6pbhf-qzpdk-kuqbr-pklfa-5ehhf-jfjps-zsj6q-57nrl-kzhpd-mu7hc-vae, e66qm-3cydn-nkf4i-ml4rb-4ro6o-srm5s-x5hwq-hnprz-3meqp-s7vks-5qe, 4ecnw-byqwz-dtgss-ua2mh-pfvs7-c3lct-gtf4e-hnu75-j7eek-iifqm-sqe, yinp6-35cfo-wgcd2-oc4ty-2kqpf-t4dul-rfk33-fsq3r-mfmua-m2ngh-jqe, w4rem-dv5e3-widiz-wbpea-kbttk-mnzfm-tzrc7-svcj3-kbxyb-zamch-hqe, cv73p-6v7zi-u67oy-7jc3h-qspsz-g5lrj-4fn7k-xrax3-thek2-sl46v-jae, o3ow2-2ipam-6fcjo-3j5vt-fzbge-2g7my-5fz2m-p4o2t-dwlc4-gt2q7-5ae, fuqsr-in2lc-zbcjj-ydmcw-pzq7h-4xm2z-pto4i-dcyee-5z4rz-x63ji-nae]
30
```

MIXLABS

获取未分配的节点可读权限

- 获取 SSH 公钥列表，这些 SSH key 有权限读取 unassigned replicas 状态。

rs > protobuf > gen > registry > registry.unassigned_nodes_config.v1.rs > ...

```rust
/// Config applied to the set of all unassigned nodes.
#[derive(serde::Serialize, serde::Deserialize)]
#[derive(Clone, PartialEq, ::prost::Message)]
pub struct UnassignedNodesConfigRecord {
    /// The list of public keys whose owners have "readonly" SSH access to all unassigned replicas,
    /// in case it is necessary to perform subnet recovery.
    #[prost(string, repeated, tag="1")]
    pub ssh_readonly_access: ::prost::alloc::vec::Vec<::prost::alloc::string::String>,
    /// The replica version that the unassigned nodes are supposed to run.
    #[prost(string, tag="2")]
    pub replica_version: ::prost::alloc::string::String,
}
```

```
UnassignedNodesConfigRecord { ssh_readonly_access: ["ssh-ed25519 AAAAC3NzaC1lZDI1NTE
5AAAAIEVyj7FIcxAUXJWXCxiAeIBAPujik5KUZCodD7a0+h85 release-eng1", "ssh-ed25519 AAAAC3
NzaC1lZDI1NTE5AAAAIDjo0TuAqR+QygwoaLOzL6liao/tybESaxW+0YXXaFPj release-eng2", "ssh-e
d25519 AAAAC3NzaC1lZDI1NTE5AAAAIAvR3Ab5+oTABpv2F2vvL1i4fLnY0TZkX2IHqvgtT5mF release-
eng3", "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIMRG8vOFQhj0iLnnMZrcyR49Y78nwjziFtFU6b/y
sLew release-eng4"], replica_version: "" }
```

MIXLABS

获取 blessed Replica 版本列表

Replica 节点执行程序
blessed replica version 可用的节点执行程序版本列表

```rust
/// A list of blessed versions of the IC Replica
///
/// New versions are added here after a vote has been accepted by token
/// holders. Subnetworks can then be upgraded to any of those version.
#[derive(serde::Serialize, serde::Deserialize)]
#[derive(Clone, PartialEq, ::prost::Message)]
pub struct BlessedReplicaVersions {
    /// A list of version information ids.
    #[prost(string, repeated, tag="1")]
    pub blessed_version_ids: ::prost::alloc::vec::Vec<::prost::alloc::string::String>,
}
```


```
BlessedReplicaVersions { blessed_version_ids: ["0.1.0", "83036e4e0a24ca207d116d1d1f63e55801539a52", "bad306332712417e7a322a35c78da7f77064b0c3", "8a560f9510b0df9e747ffaede3b731f2ade9c0b7", "ca35377220efd5efb1f5944e34c4d6caf1aff2df", "232c2bef48f2b6598b9851c5947939bee1e1caa2", "875b404679d46475b705d3575e8f952ed3d43e2f", "951670011359c23c803f5351a3593531af4a63ad", "8dc1a28b4fb9605558c03121811c9af9701a6142", "bddbd2da1ec60684e5e79609d40e55d4acac6617", "bd131057c9023790fb9195e8e90a6df4f88b8e5a", "0b9f52064d302aa811f4d8d19b0fcbfb7af86cf9", "9bc4c929f78868ffce1285e32ba2d257e65a68c2", "7bf42abde1e612521ef8447d17a6d8543603ff39", "8846de31a45a3f5b61bd6b513ed9e579a872460f", "3761cee0c1b8ec1b02769f9146c516c1df1dd172", "cbfe66aa0a69da2200dae568d6d041459fae0190", "b0799e1d120a6cbc74014030cd4cca5fabc503fc", "1c87717b2a4efc88c851fe1ad819941ea9ea737b", "319db9ad7cd35ee39624d03580f7eb4de492ba8f", "96db09d2593cfe425b6c44c705bf5e7f27ffd0d8", "071d8388c1b4905072c9417ede0f75cea4002600", "c47a773b97f9e45b2760caaee4ad24aa6d5c9b69", "ffa50f34c3c4a412cd2cda5da1b1e08a6260f10b", "27e1eadbcbe90abfe56d9c8dfd39e1a78e52c624", "d8dabe1cb0bb1e60a11e5bb8033d1615fefb252f", "8a5b9a2e1468dfb286c77084a9b3597b9e3993b5", "a5e6cdef55ad24a761b11f26a2ee8001b905fdbb", "3db2c9acef8efc424a78d30678bc194ac37c367a", "80496d53829d906fd95bf4e1e99b4d06f193f98b", "ac80cad9faff2ebb87e51b833175c1fd101a489f", "e86ac9553a8eddbeffaa29267a216c9554d3a0c6", "3eaf8541c389badbd6cd50fff31e158505f4487d", "b099ca45f21dc84d9f9dd6d14cc8cb4e7c00e3e1", "3b34266b4c74660a002b2a2ca44d70a56b5865b4", "4f9257ae68595499b21ee654525c42e55c53e9c8", "f2fc23733b52c53c8f1cfc0
```


MIXLABS

获取 Replica 执行程序

- 传入一个 Replica Version，返回该 version 安装包下载链接

```rust
/// Information about a Replica version
#[derive(serde::Serialize, serde::Deserialize)]
#[derive(Clone, PartialEq, ::prost::Message)]
pub struct ReplicaVersionRecord {
    /// The URL against which a HTTP GET request will return a release package
    /// that corresponds to this version
    #[prost(string, tag="5")]
    pub release_package_url: ::prost::alloc::string::String,
    /// The hex-formatted SHA-256 hash of the archive file served by 'release_package_url'
    #[prost(string, tag="6")]
    pub release_package_sha256_hex: ::prost::alloc::string::String,
}
```

```
ReplicaVersionRecord { release_package_url: "https://download.dfinity.
systems/ic/875b404679d46475b705d3575e8f952ed3d43e2f/guest-os/update-im
g/update-img.tar.gz", release_package_sha256_hex: "37582868b7121c6ec69
c6e3963b5651807b63d958745c446b5c83b20ff700e21" }
```

MIXLABS

获取防火墙配置

rs > protobuf > gen > registry > registry.firewall.v1.rs > ...

```rust
/// Firewall configuration
#[derive(serde::Serialize, serde::Deserialize)]
#[derive(Clone, PartialEq, ::prost::Message)]
pub struct FirewallConfig {
    /// The firewall configuration content
    #[prost(string, tag="1")]
    pub firewall_config: ::prost::alloc::string::String,
    /// List of allowed IPv4 prefixes
    #[prost(string, repeated, tag="2")]
    pub ipv4_prefixes: ::prost::alloc::vec::Vec<::prost::alloc::string::String>,
    /// List of allowed IPv6 prefixes
    #[prost(string, repeated, tag="3")]
    pub ipv6_prefixes: ::prost::alloc::vec::Vec<::prost::alloc::string::String>,
}
```


```
FirewallConfig { firewall_config: "table filter {\n  chain INPUT {\n    type filter hook input priority 0; policy drop;\n    iif lo accept\n    ct state { invalid } drop\n    ct state { established, related } accept\n    icmp type destination-unreachable accept\n    icmp type source-quench accept\n    icmp type time-exceeded accept\n    icmp type parameter-problem accept\n    icmp type echo-request accept\n    icmp type echo-reply accept\n  }\n\n  chain FORWARD {\n    type filter hook forward priority 0; policy drop;\n  }\n\n  chain OUTPUT {\n    type filter hook output priority 0; policy accept;\n  }\n}\n\ndefine IPV6_PREFIXES={\n  << ipv6_prefixes >>\n}\n\ntable ip6 filter {\n  chain INPUT {\n    type filter hook input priority 0; policy drop;\n    iif lo accept\n    ct state { invalid } drop\n    ct state { established, related } accept\n    icmpv6 type destination-unreachable accept\n    icmpv6 type packet-too-big accept\n    icmpv6 type time-exceeded accept\n    icmpv6 type parameter-problem accept\n    icmpv6 type echo-request accept\n    icmpv6 type echo-reply accept\n    icmpv6 type nd-router-advert accept\n    icmpv6 type nd-neighbor-solicit accept\n    icmpv6 type nd-neighbor-advert accept\n    ip6 saddr $IPV6_PREFIXES ct state { new } tcp dport { 22, 2497, 4100, 8080, 9090, 9091, 9100, 19531 } accept\n  }\n\n  chain FORWARD {\n    type filter hook forward priority 0; policy drop;\n  }\n\n  chain OUTPUT {\n    type filter hook output priority 0; policy accept;\n  }\n}\n", ipv4_prefixes: [], ipv6_prefixes: ["2001:438:fffd:11c::/64", "2001:470:1:c76::/64", "2001:4d78:400:10a::/64", "2001:4d78:40d::/48", "2001:920:401a:1706::/64", "2001:920:401a:1708::/64", "2001:920:401a:1710::/64", "2401:3f00:1000:22::/64", "2401:3f00:1000:23::/64", "2401:3f00:1000:24::/64", "2600:2c01:21::/64", "2600:3000:1300:1300::/64", "2600:3000:6100:200::/64", "2600:3004:1200:1200::/56", "2600:3006:1400:1500::/64", "2600:c02:b002:15::/64", "2600:c0d:3002:4::/64", "2604:1380:4091:3000::/64", "2604:1380:40e1:4700::/64", "2604:1380:40f1:1700::/64", "2604:1380:45d1:bf00::/64", "2604:1380:45e1:a600::/64", "2604:1380:45f1:9400::/64", "2604:1380:4601:6200::/64", "2604:1380:4641:6100::/64", "2604:3fc0:2001::/48", "2604:3fc0:3002::/48", "2604:6800:258:1::/64", "2604:7e00:30:3::/64", "2604:7e00:50::/64", "2604:b900:4001:76::/64", "2607:f1d0:10:1::/64", "2607:f6f0:3004::/48", "2607:f758:1220::/64", "2607:f758:c300::/64", "2607:fb58:9005::/48", "2607:ff70:3:2::/64", "2610:190:6000:1::/64", "2610:190:df01:5::/64", "2a00:fa0:3::/48", "2a00:fb01:400:100::/56", "2a00:fb01:400::/56", "2a00:fc0:5000:300::/64", "2a01:138:900a::/48", "2a01:2a8:a13c:1::/64", "2a01:2a8:a13d:1::/64", "2a01:2a8:a13e:1::/64", "2a02:418:3002:0::/64", "2a02:41b:300e::/48", "2a02:800:2:2003::/64", "2a04:9dc0:0:108::/64", "2a05:d01c:e2c:a700::/56", "2a0b:21c0:b002:2::/64", "2a0f:cd00:0002::/56", "fd00:2:1:1:1::/64"] }
```


MIXLABS

创世纪的超级管理员名单

放心，现在这个名单已经清空了

```rust
#[derive(serde::Serialize, serde::Deserialize)]
#[derive(Clone, PartialEq, ::prost::Message)]
pub struct ProvisionalWhitelist {
    #[prost(enumeration="provisional_whitelist::ListType", tag="1")]
    pub list_type: i32,
    /// This must be empty if list_type is of variant ALL.
    #[prost(message, repeated, tag="2")]
    pub set: ::prost::alloc::vec::Vec<super::super::super::types::v1::PrincipalId>,
```

```
ProvisionalWhitelist { list_type: Set, set: [] }
```

MIXLABS

获取某个 node operator 对应的信息

需要传入一个 operator id

```rust
#[derive(candid::CandidType, serde::Serialize, candid::Deserialize)]
#[derive(Clone, PartialEq, ::prost::Message)]
pub struct NodeOperatorRecord {
    /// The principal id of the node operator. This principal is the entity that
    /// is able to add and remove nodes.
    ///
    /// This must be unique across NodeOperatorRecords.
    #[prost(bytes="vec", tag="1")]
    pub node_operator_principal_id: ::prost::alloc::vec::Vec<u8>,
    /// The remaining number of nodes that could be added by this node operator.
    /// This number should never go below 0.
    #[prost(uint64, tag="2")]
    pub node_allowance: u64,
    /// The principal id of this node operator's provider.
    #[prost(bytes="vec", tag="3")]
    pub node_provider_principal_id: ::prost::alloc::vec::Vec<u8>,
    /// The ID of the data center where this Node Operator hosts nodes.
    #[prost(string, tag="4")]
    pub dc_id: ::prost::alloc::string::String,
    /// A map from node type to the number of nodes for which the associated Node
    /// Provider should be rewarded.
    #[prost(btree_map="string, uint32", tag="5")]
    pub rewardable_nodes: ::prost::alloc::collections::BTreeMap<::prost::alloc::string::String, u32>,
}
```

```
NodeOperatorRecord { node_operator_principal_id: [10, 94, 169, 75, 121, 105,
119, 81, 231, 4, 100, 84, 209, 127, 22, 156, 112, 234, 157, 235, 221, 120, 15
8, 38, 167, 29, 89, 22, 2], node_allowance: 3, node_provider_principal_id: [2
41, 55, 129, 125, 140, 42, 62, 197, 159, 165, 107, 83, 132, 76, 29, 57, 105,
229, 20, 26, 144, 45, 85, 236, 250, 114, 28, 163, 2], dc_id: "sg2", rewardabl
e_nodes: {"type0": 14} }
```

MIXLABS

获取某个节点的信息

传入某个
node id

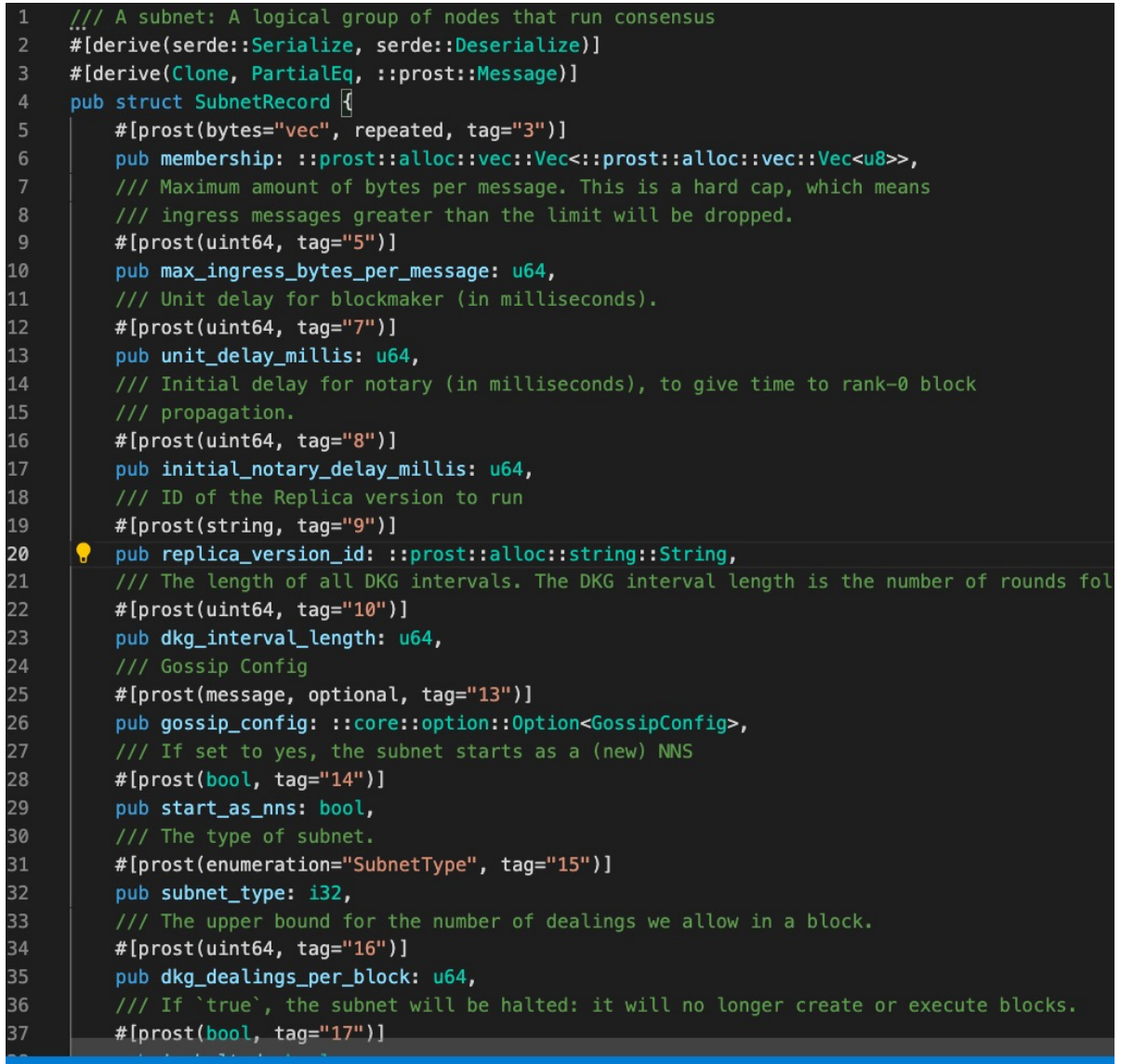
```rust
/// A node: one machine running a replica instance.
#[derive(serde::Serialize, serde::Deserialize)]
#[derive(Clone, PartialEq, ::prost::Message)]
pub struct NodeRecord {
    /// The endpoint where this node receives xnet messages.
    #[prost(message, optional, tag="5")]
    pub xnet: ::core::option::Option<ConnectionEndpoint>,
    /// The endpoint where this node receives http requests.
    #[prost(message, optional, tag="6")]
    pub http: ::core::option::Option<ConnectionEndpoint>,
    /// The P2P flow end points.
    #[prost(message, repeated, tag="8")]
    pub p2p_flow_endpoints: ::prost::alloc::vec::Vec<FlowEndpoint>,
    /// Endpoint where the node provides Prometheus format metrics over HTTP
    #[prost(message, optional, tag="10")]
    pub prometheus_metrics_http: ::core::option::Option<ConnectionEndpoint>,
    /// Endpoints on which the public API is served.
    #[prost(message, repeated, tag="11")]
    pub public_api: ::prost::alloc::vec::Vec<ConnectionEndpoint>,
    /// Endpoints on which private APIs are served.
    #[prost(message, repeated, tag="12")]
    pub private_api: ::prost::alloc::vec::Vec<ConnectionEndpoint>,
    /// Endpoints on which metrics compatible with the Prometheus export
    /// format are served.
    #[prost(message, repeated, tag="13")]
    pub prometheus_metrics: ::prost::alloc::vec::Vec<ConnectionEndpoint>,
    /// Endpoints on which the XNet API is served
    #[prost(message, repeated, tag="14")]
    pub xnet_api: ::prost::alloc::vec::Vec<ConnectionEndpoint>,
    /// The id of the node operator that added this node.
    #[prost(bytes="vec", tag="15")]
    pub node_operator_id: ::prost::alloc::vec::Vec<u8>,
}
```

```
NodeRecord { xnet: Some(ConnectionEndpoint { ip_addr: "2401:3f00:1000:
22:5000:c3ff:fe44:36f4", port: 2497, protocol: Http1 }), http: Some(Co
nnectionEndpoint { ip_addr: "2401:3f00:1000:22:5000:c3ff:fe44:36f4", p
ort: 8080, protocol: Http1 }), p2p_flow_endpoints: [FlowEndpoint { flo
w_tag: 1234, endpoint: Some(ConnectionEndpoint { ip_addr: "2401:3f00:1
000:22:5000:c3ff:fe44:36f4", port: 4100, protocol: Http1 }) }], promet
heus_metrics_http: Some(ConnectionEndpoint { ip_addr: "2401:3f00:1000:
22:5000:c3ff:fe44:36f4", port: 9090, protocol: Http1 }), public_api: [
], private_api: [], prometheus_metrics: [], xnet_api: [], node_operato
r_id: [10, 94, 169, 75, 121, 105, 119, 81, 231, 4, 100, 84, 209, 127,
22, 156, 112, 234, 157, 235, 221, 120, 158, 38, 167, 29, 89, 22, 2] }
"qffmn-uqkl2-uuw6l-jo5i6-obdek-tix6f-u4odv-j3265-pcpcn-jy5le-lae"
→  nns_sync git:(master) ✗
```

MIXLABS

获取某个子网相关信息

- 传入子网id

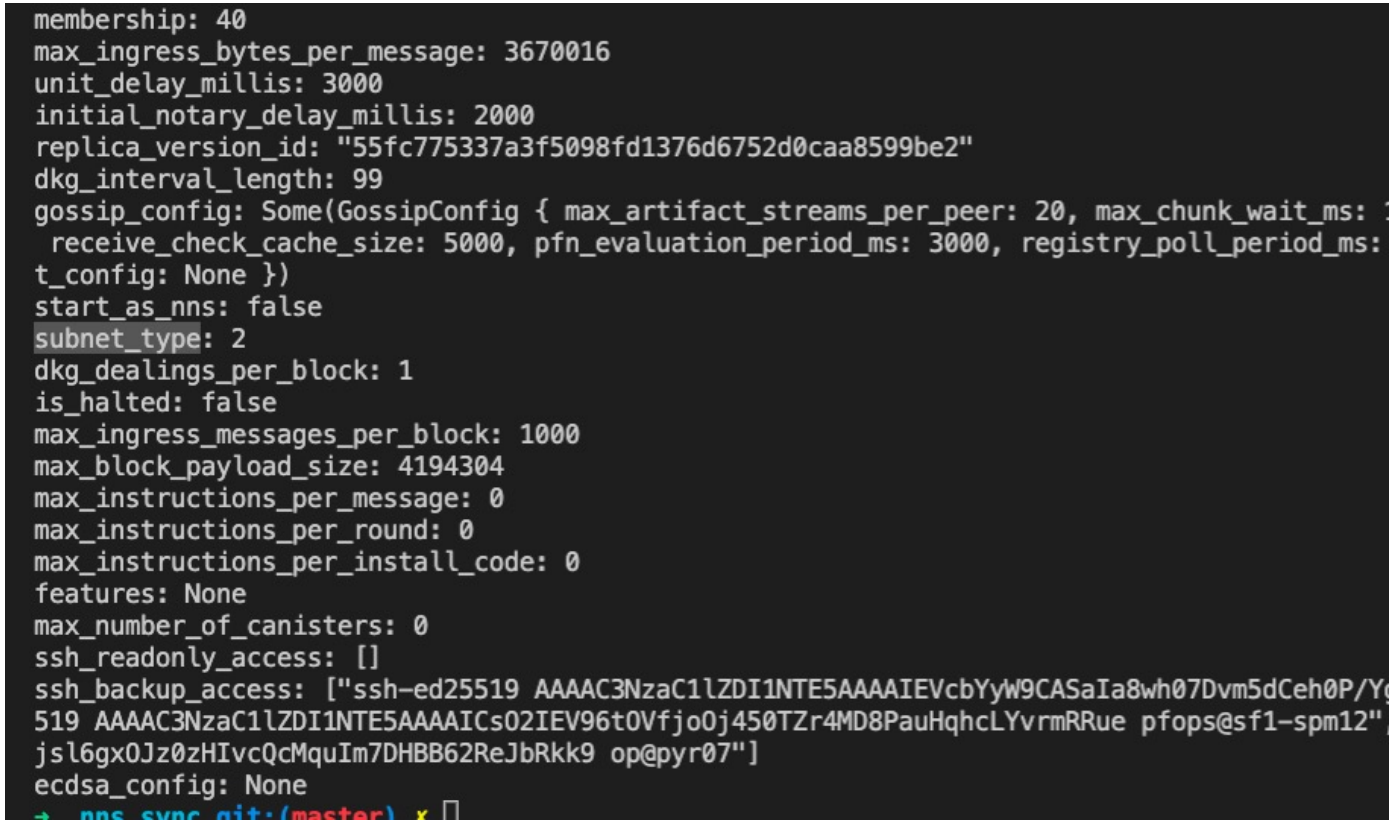


```rust
/// A subnet: A logical group of nodes that run consensus
#[derive(serde::Serialize, serde::Deserialize)]
#[derive(Clone, PartialEq, ::prost::Message)]
pub struct SubnetRecord {
    #[prost(bytes="vec", repeated, tag="3")]
    pub membership: ::prost::alloc::vec::Vec<::prost::alloc::vec::Vec<u8>>,
    /// Maximum amount of bytes per message. This is a hard cap, which means
    /// ingress messages greater than the limit will be dropped.
    #[prost(uint64, tag="5")]
    pub max_ingress_bytes_per_message: u64,
    /// Unit delay for blockmaker (in milliseconds).
    #[prost(uint64, tag="7")]
    pub unit_delay_millis: u64,
    /// Initial delay for notary (in milliseconds), to give time to rank-0 block
    /// propagation.
    #[prost(uint64, tag="8")]
    pub initial_notary_delay_millis: u64,
    /// ID of the Replica version to run
    #[prost(string, tag="9")]
    pub replica_version_id: ::prost::alloc::string::String,
    /// The length of all DKG intervals. The DKG interval length is the number of rounds fol
    #[prost(uint64, tag="10")]
    pub dkg_interval_length: u64,
    /// Gossip Config
    #[prost(message, optional, tag="13")]
    pub gossip_config: ::core::option::Option<GossipConfig>,
    /// If set to yes, the subnet starts as a (new) NNS
    #[prost(bool, tag="14")]
    pub start_as_nns: bool,
    /// The type of subnet.
    #[prost(enumeration="SubnetType", tag="15")]
    pub subnet_type: i32,
    /// The upper bound for the number of dealings we allow in a block.
    #[prost(uint64, tag="16")]
    pub dkg_dealings_per_block: u64,
    /// If `true`, the subnet will be halted: it will no longer create or execute blocks.
    #[prost(bool, tag="17")]
```

MIXLABS

获取某个子网相关信息

- 传入子网id



获取某个子网的 bls 公钥

- 传入subnetId

```rust
#[derive(serde::Serialize, serde::Deserialize)]
#[derive(Clone, PartialEq, ::prost::Message)]
pub struct PublicKey {
    #[prost(uint32, tag="1")]
    pub version: u32,
    #[prost(enumeration="AlgorithmId", tag="2")]
    pub algorithm: i32,
    #[prost(bytes="vec", tag="3")]
    pub key_value: ::prost::alloc::vec::Vec<u8>,
    #[prost(message, optional, tag="4")]
    pub proof_data: ::core::option::Option<::prost::alloc::vec::Vec<u8>>,
}
```

```
PublicKey { version: 0, algorithm: ThresBls12381, key_value: [129, 76, 14, 11
0, 199, 31, 171, 88, 59, 8, 189, 129, 55, 60, 37, 92, 60, 55, 27, 46, 132, 13
4, 60, 152, 164, 241, 224, 139, 116, 35, 93, 20, 251, 93, 156, 12, 213, 70, 2
17, 104, 95, 145, 58, 12, 11, 44, 197, 52, 21, 131, 191, 75, 67, 146, 228, 10
3, 219, 150, 214, 91, 155, 180, 203, 113, 113, 18, 248, 71, 46, 13, 90, 77, 2
0, 80, 95, 253, 116, 132, 176, 18, 145, 9, 28, 95, 135, 185, 136, 131, 70, 63
, 152, 9, 26, 11, 170, 174], proof_data: None }
```

MIXLABS

怎么获取到

- Registry canister 有一个 query 接口 get_value，传入对于的 key，就能返回结果的 bytes。根据对应的数据结构解码就可以得到。

```rust
#[export_name = "canister_query get_value"]
fn get_value() {
    let response_pb: RegistryGetValueResponse = match deserialize_get_value_request(arg_data()) {
        Ok((key: Vec<u8>, version_opt: Option<u64>)) => {
            let registry: &Registry = registry();
            let version: u64 = version_opt.unwrap_or_else(|| registry.latest_version());
            let result: Option<&RegistryValue> = registry.get(&key, version);
            match result {…
        }
        Err(error: Error) => RegistryGetValueResponse {
            error: Some(RegistryError {
                code: Code::MalformedMessage as i32,
                key: Vec::<u8>::default(),
                reason: error.to_string(),
            }),
            version: 0,
            value: Vec::<u8>::default(),
        },
    };
    let bytes: Vec<u8> = serialize_get_value_response(response_pb).expect(msg: "Error serializing response"
    reply(payload: &bytes);
}
```

MIXLABS

- 接下来请 PYD 用 go-agent 演示一下

- https://github.com/dfinity/ic/
- https://github.com/mix-labs/IC-Go/blob/v0.0.1/example/registry/registry_test.go
- https://github.com/mix-labs/nns_sync/blob/master/src/main.rs

MIXLABS