

Programação Distribuída - Communications

Autor: Ariel Rossetto Ril

Matrícula: 15105050

Email: ariel.ril@edu.pucrs.br

Repositório: <https://github.com/arielril/distrprog-node-comms>

Introdução

Neste trabalho é apresentado uma implementação de um sistema P2P (peer-to-peer) básico, o qual possui um conjunto de nodos e supernodos. Os nodos são componentes que possuem uma lista de recursos, neste caso uma lista de arquivos, disponibilizado para consumo do cliente do sistema. Os supernodos são responsáveis por manter um tipo de "repositório" centralizado da informação de onde os recursos estão armazenados.

Cada nodo dentro do ecossistema executa um registro em um supernodo para que seja possível que outros nodos no sistema possam descobrir onde está algum recurso solicitado pelo cliente do sistema, sem ter a necessidade de ter que solicitar para todos os nodos dentro do ecossistema. Dessa maneira, ao nodo solicitar ao seu supernodo um recurso, o supernodo executa uma pesquisa local em busca do recurso e caso o supernodo não tenha a resposta local o supernodo executa a pesquisa do recurso dentro de um grupo multicast, onde todos os supernodos estão registrados para responder as solicitações de seus parceiros.

Desenvolvimento

Para realizar o desenvolvimento deste trabalho foi utilizado a linguagem de programação Python, utilizando o framework **Flask** para as comunicações que utilizam o protocolo HTTP em uma API Restful e foi utilizado o pacote `socket` para realizar a comunicação multicast (UDP) entre os supernodos.

Organização

A organização do código para este trabalho está dividido em:

1. Model
 1. Node
 2. Supernode

3. UDP
2. Api
 1. Node
 2. Supernode
3. Background
 1. Supernode Check Liveness
4. Config
5. Resources

Model - Node

Esta classe possui a lógica referente ao funcionamento de um Nodo no sistema. Utilizando esta classe é possível:

- Listar os recursos de um nodo
- Buscar as informações sobre um recurso pelo seu identificador
- Buscar as informações sobre um recurso pelo seu identificador no Supernodo em que o Nodo esta registrado

Ao inicializar uma instância de `Node` é executado o registro do Nodo em um Supernodo, o Supernodo é definido através de uma variável de ambiente. Durante o processo de inicialização, o Nodo executa a busca dos seus recursos e registro em uma "base" local.

Durante o processo de inicialização o Nodo executa uma listagem e leitura dos recursos pertencentes ao Nodo. Para definir um identificador único entre os recursos em todos os Nodos, foi utilizado uma abordagem de ler o conteúdo dos recursos e a partir do conteúdo gerar um hash SHA 256 do conteúdo, tendo assim um identificador único para o recurso.

```

class Node:
    # resource => map['hash_id'] = 'file_name'
    resources = {}
    # endpoint of the supernode
    supernode_url = ""
    # uuid v4
    node_name = ""
    # node address
    location = ""

    def __init__(self, location: str, supernode: str): ...

    def get_resources_for_registration(self): ...

    def register_to_supernode(self, app): ...

    def load_resources(self, path=""): ...

    @classmethod
    def list_resources(cls): ...

    @classmethod
    def get_resource(cls, id: str): ...

    @classmethod
    def supernode_search_resource(cls, id: str): ...

```

Model - Supernode

Esta classe possui a lógica referente ao funcionamento de um Supernodo no sistema. Utilizando esta classe é possível:

- Registrar um Nodo
- Buscar as informações sobre um recurso pelo seu identificador
- Buscar os Nodos vivos
- Realizar um processo de identificação dos Nodos que estão registrados e estão vivos

Ao inicializar uma instância de **Supernode** é inicializado uma nova thread para responder às solicitações de recursos de outros Supernodos e é inicializado um *job* (processo que executa de 5-5 segundos) para inicializar o processo de identificação de Nodos vivos. Durante o processo de identificação de Nodos registrados que estão vivos é verificado se o Nodo não responde a 2 verificações seguidas (10 segundos), este Nodo é considerado morto e removido da listagem de nodos e removido o registro de seus recursos.

```

class Supernode:
    # entry => ['<file_hash>'] -> { file_name:str, node_location }
    node_resources = {}

    # entry => ['<node_location>'] -> { is_alive:bool, last_check:time }
    nodes = {}

    multicast_location = ""
    location = ""
    multicast_group_size = []
    name = str(uuid4())

    def __init__(self, location="", multicast_loc="", multicast_group_size=0): ...

    @classmethod
    def register_node(cls, location: str, resource_list, name="") -> (bool, Exception): ...

    @classmethod
    def get_resource_location_by_id(cls, id: str): ...

    @classmethod
    def multicasting(cls, file_id=""): ...

    @classmethod
    def get_alive_nodes(cls): ...

    @classmethod
    def _remove_resources_from_node(cls, location=""): ...

    @classmethod
    def _check_and_remove_dead_node(cls, location=""): ...

    @classmethod
    def _remove_nodes(cls, node_lst=[]): ...

    @classmethod
    def check_alive_nodes(cls): ...

```

Model - UDP

Esta classe é responsável pelo funcionamento da comunicação multicast UDP entre os Supernodos. Utilizando esta classe é possível:

- Enviar mensagens de pesquisa de recurso. Essas mensagens são no formato `<supernode_name>;20;<hash_id>`
- Enviar mensagens de resposta sobre uma pesquisa de recurso. Nesse caso é utilizado dois códigos de mensagem: `21` e `22`. No caso do código `21` é uma resposta OK, ou seja, o Supernodo que respondeu tem um registro do recurso e enviou a localização do recurso (`<supernode_name>;21;<file_location>`). No caso

do código `22` é uma resposta NOK, ou seja, o Supernodo respondeu que não possui conhecimento sobre o recurso solicitado (`<supernode_name>;22;<hash_id>`).

- Escutar ao grupo multicast esperando solicitações de pesquisa de recursos

```
class Multi:
    #
    #
    #
    supernode = None
    supernode_name = ""
    # number of messages to expect (number of supernodes)
    messages_number = 0

    group = ("230.0.0.0", 4321)

    def __init__(self, supernode=None, msgs_num=0, supernode_name=""): ...

    def init_sock(self): ...

    def process_search_request(self, message=""): ...

    def process_search_response(self, file_hash="", message=""): ...

    def receive(self): ...

    def send(self, message=""): ...

    def request_file_search_and_listen(self, file_hash=""): ...
```

Api - Node

Este arquivo possui a declaração e a lógica por trás dos recursos disponibilizados pela API de Nodos. A API disponibiliza as funcionalidades:

- Buscar os recursos de um Nodo. `GET /node/resources`
- Buscar um recurso específico. `GET /node/resources/<hash_id>`
- Buscar uma lista de recursos. `GET /node/resources/find?hash_ids=1,2,3,4`
- Verificar a saúde do nodo. `GET /node/health`

```

@bp.route("/node/resources", methods=["GET"])
> def list_resources(): ...

@bp.route("/node/resources/<string:id>", methods=["GET"])
> def get_resource_info(id): ...

# /node/resources/find?hash_ids=13123132,12312
@bp.route("/node/resources/find", methods=["GET"])
> def find_resources(): ...

@bp.route("/node/health", methods=["GET"])
> def health(): ...

```

Api - Supernode

Este arquivo possui a declaração e a lógica por trás dos recursos disponibilizados pela API de Supernodos. A API disponibiliza as funcionalidades:

- Registro de um Nodo. `POST /supernode/register`
- Buscar um recurso. `GET /supernode/search/<hash_id>`
- Buscar uma lista de recursos. `GET /supernode/search?hash_ids=1,2,3,4`
- Buscar os Nodos que estão vivos. `GET /supernode/alive`
- Executar o processo de avaliação de Nodos vivos. `GET /supernode/xxx`

```

@bp.route("/supernode/register", methods=["POST"])
def register(): ...

# GET /search/<file_id> | GET /search?node_name=<str>&file_name=<str>&hash_ids=<str>
@bp.route("/supernode/search/<string:file_id>", methods=["GET"])
def resource_search(file_id=""): ...

@bp.route("/supernode/search", methods=["GET"])
def resource_list_search(): ...

@bp.route("/supernode/alive", methods=["POST"])
def is_node_alive(): ...

@bp.route("/supernode/alive", methods=["GET"])
def get_alive_nodes(): ...

@bp.route("/supernode/xxx", methods=["GET"])
def xxx(): ...

```

Background - Supernode Check Liveness

Este processo é responsável por solicitar ao Supernodo, através de sua API, para que seja executado o processo de verificação de vida dos Nodos. Esse processo consiste no Supernodo realizar uma request HTTP para todos os Nodos registrados no path `/node/health`. Caso um Nodo não responda a duas solicitações de saúde, o Nodo é removido da lista de Nodos registrados no Supernodo e todos seus recursos são removidos da listagem do Supernodo.

Este processo é inicializado ao criar um Supernodo e é executado em uma espécie de *cronjob*.

```

def start_task(supernode_addr=""):
    if supernode_addr == "":
        return

    def check_alive_nodes():
        import requests

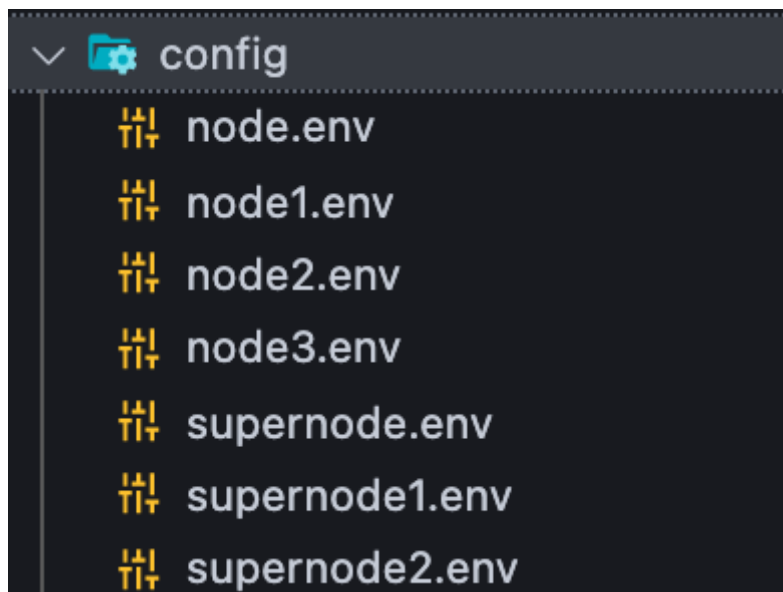
        try:
            requests.get(
                f"http://{supernode_addr}/api/supernode/xxx",
            )
            print("[+] requested supernode to run liveness check")
        except Exception as e:
            print("[.] failed to request supernode to run liveness check", e)

    sched = BackgroundScheduler()
    sched.add_job(
        func=check_alive_nodes,
        trigger="interval",
        seconds=5,
    )
    sched.start()
    atexit.register(lambda: sched.shutdown())

```

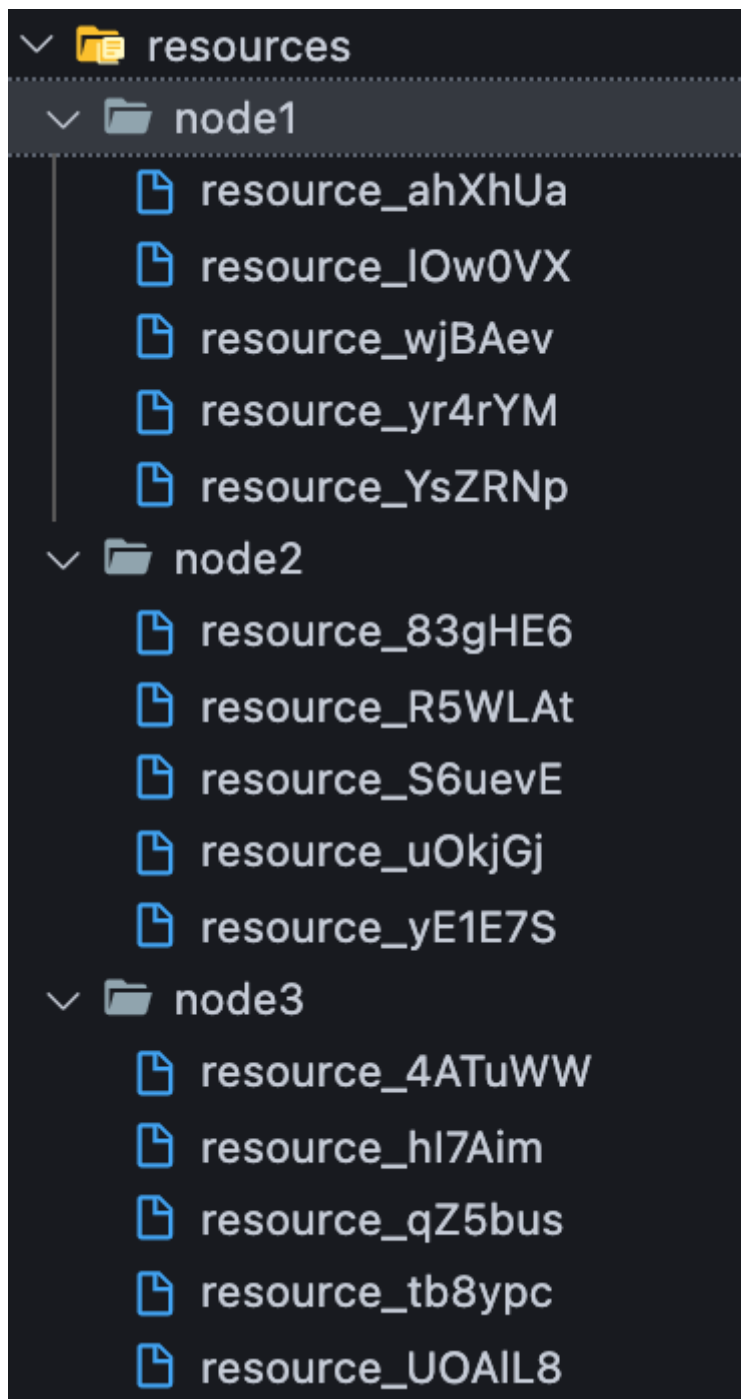
Config

Esta pasta possui os arquivos `.env` para servir de configuração tanto para os Nodos quanto os Supernodos. Cada instância de Nodo ou Supernodo possui um arquivo específico, para não misturar as configurações de cada instância. Entre as configurações de um Nodo é definido quem é seu Supernodo e o caminho para seus recursos.



Resources

Esta pasta possui um conjunto de pastas e arquivos que servem como sendo os recursos que cada Nodo possui a sua disposição.



Demonstração

Máquinas Virtuais

Para iniciar as máquinas virtuais para a demonstração é necessário ter a ferramenta **vagrant**(<https://www.vagrantup.com>) instalada além de possuir VirtualBox instalado. Após instalar as ferramentas necessárias é preciso estar na pasta `machine`, a qual contém o arquivo `Vagrantfile` com as configurações necessárias para iniciar 3 máquinas virtuais, para executar o comando `vagrant up`.

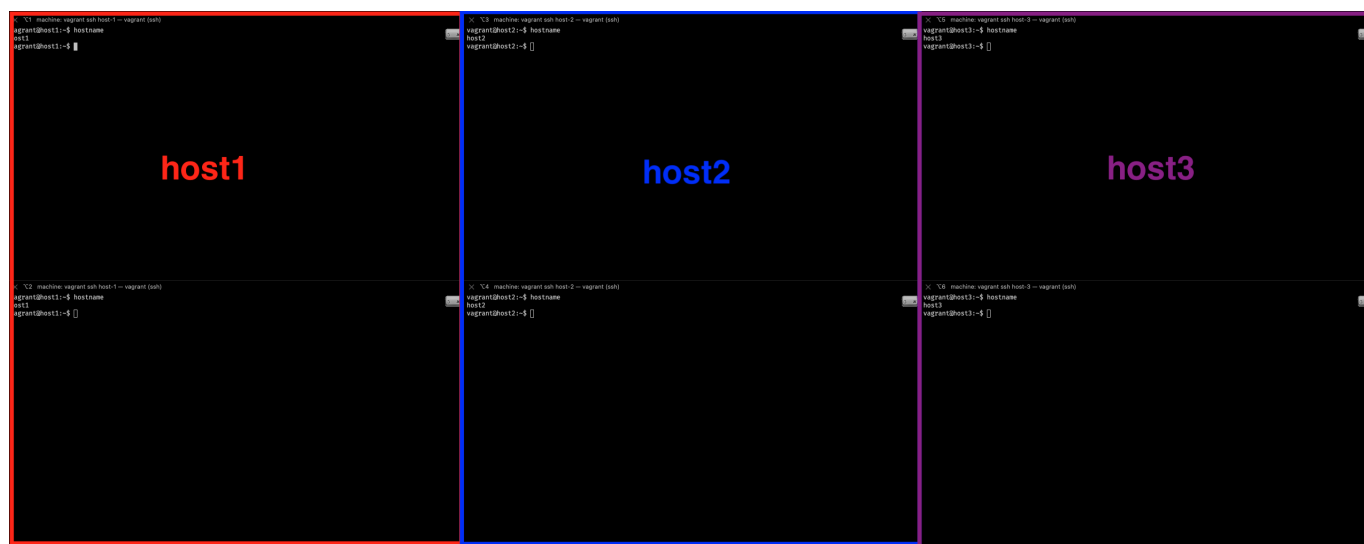
```

distrprog-node-comms.nosync/machine on  master [?] via  v2.2.16
> vagrant up
Bringing machine 'host-1' up with 'virtualbox' provider...
Bringing machine 'host-2' up with 'virtualbox' provider...
Bringing machine 'host-3' up with 'virtualbox' provider...
==> host-1: Checking if box 'debian/buster64' version '10.20210228.1' is up to date...
^C==> host-1: Waiting for cleanup before exiting...
==> host-1: There was a problem while downloading the metadata for your box
==> host-1: to check for updates. This is not an error, since it is usually due
==> host-1: to temporary network problems. This is just a warning. The problem
==> host-1: encountered was:
==> host-1:
==> host-1:
==> host-1:
==> host-1: If you want to check for box updates, verify your network connection
==> host-1: is valid and try again.

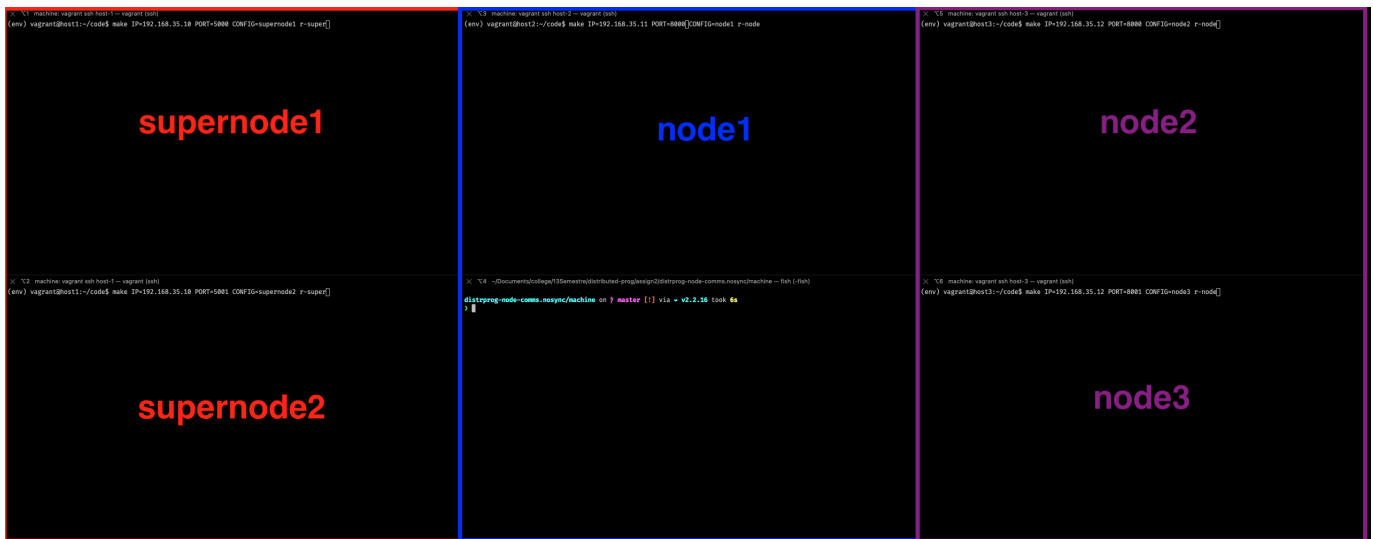
```

Simulação

Para executar a simulação é possível iniciar a quantidade que quiser de conexões com as máquinas virtuais, para este trabalho foi utilizado 3 máquinas virtuais e 5 conexões. Durante a simulação foi utilizado 2 Supernodos e 3 Nodos.



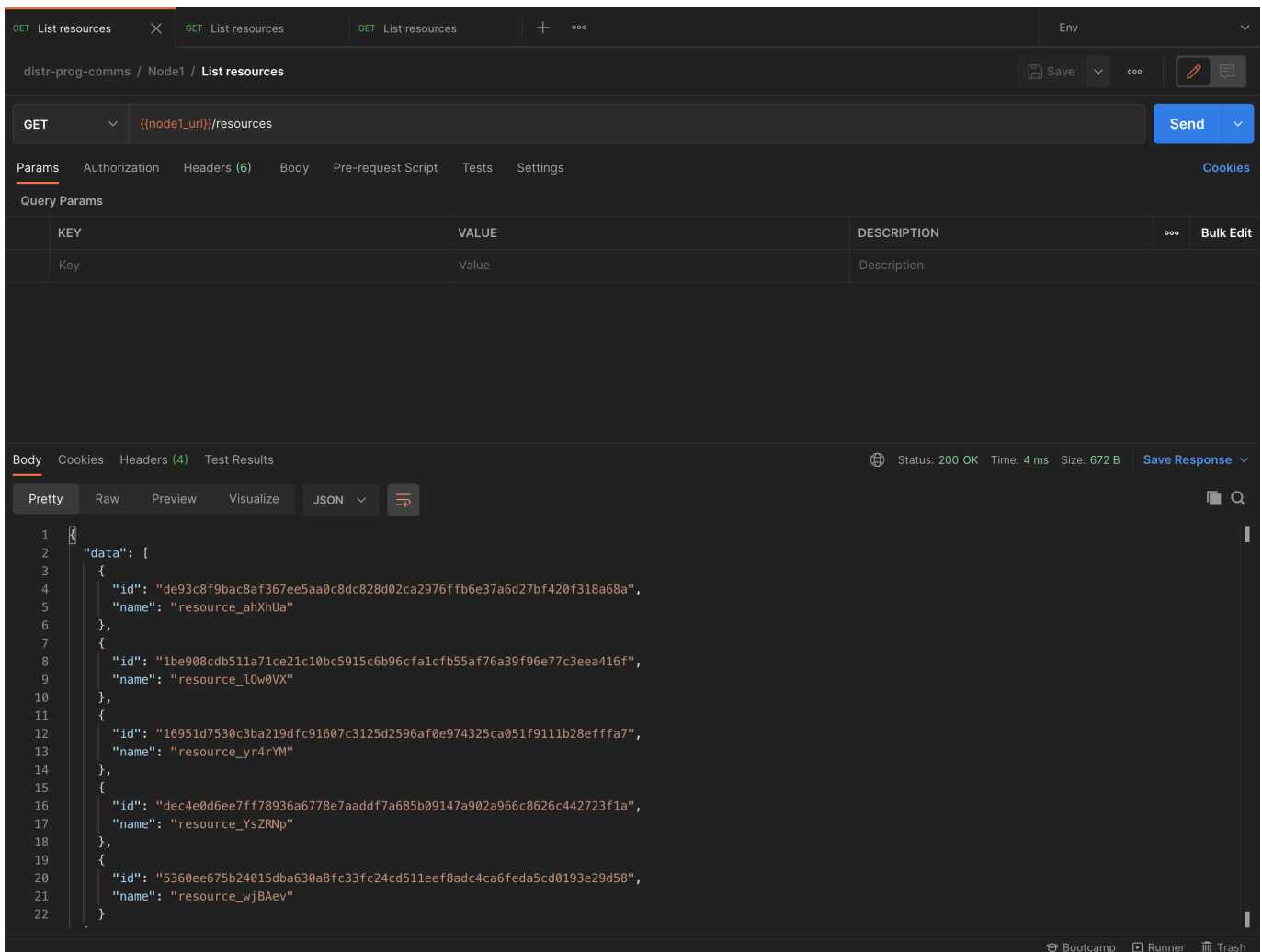
Na máquina 1 (**host-1**) foram executados os dois Supernodos, um disponibilizado no endereço 192.168.35.10:5000 e outro no endereço 192.168.35.10:5001. Na máquina 2 (**host-2**) foi executado um Nodo (**node1**), o qual foi disponibilizado no endereço 192.168.35.11:8000. Na máquina 3 (**host-3**) foram disponibilizados outros dois Nodos, o Nodo 2 disponibilizado no endereço 192.168.35.12:8000 e o Nodo 3 disponibilizado no endereço 192.168.35.12:8001.



Como todos os serviços foram disponibilizados através de uma API utilizando o protocolo HTTP, foi possível utilizar o programa *Postman* para realizar requisições HTTP e interagir com todos os serviços.

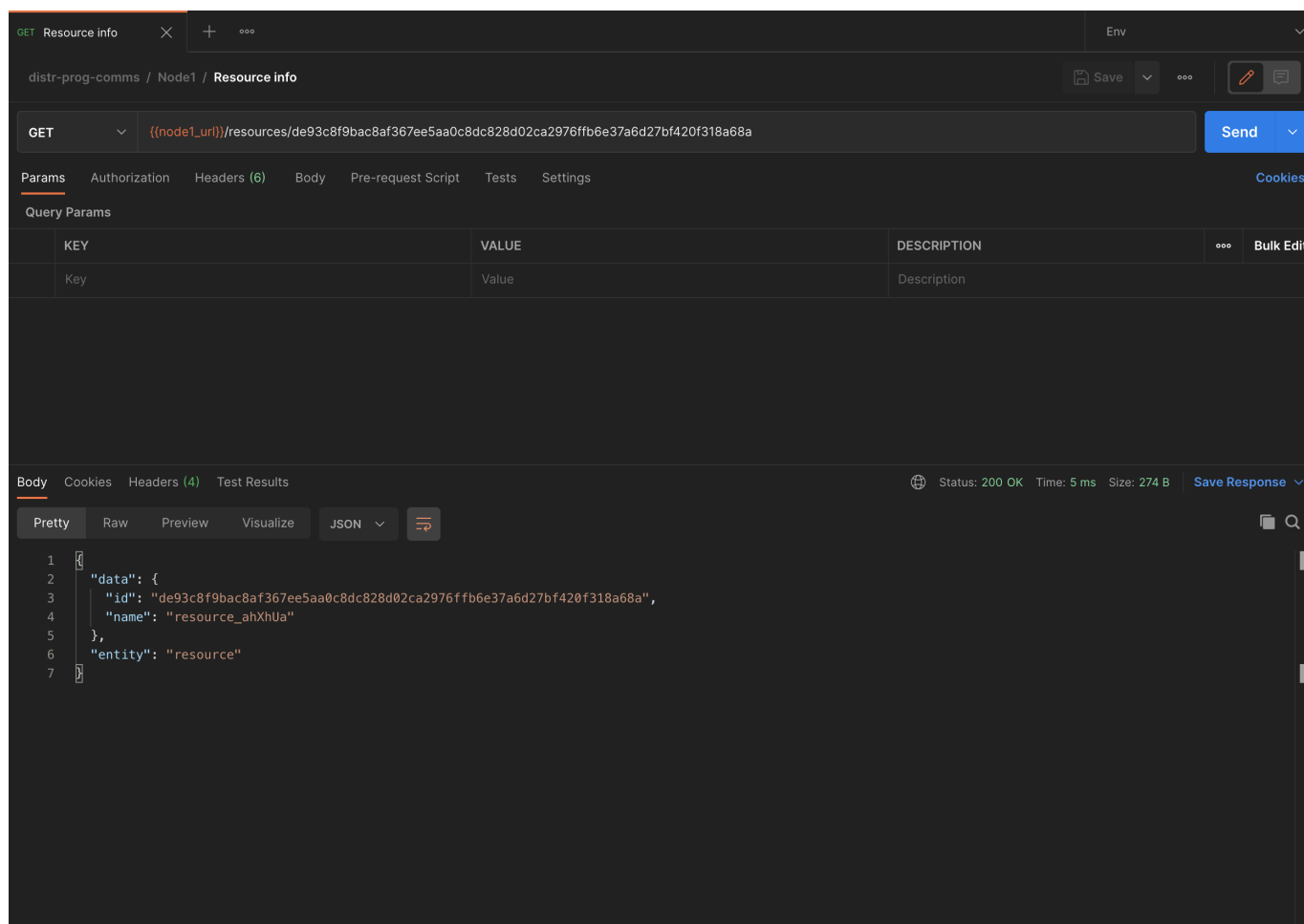
Listagem de recursos de um Nodo

Neste exemplo foi executado a listagem de recursos do Nodo 1.



Busca de informações sobre um recurso

Neste exemplo foi executado a busca de informações de um recurso provido pelo Nodo 1. Neste caso, não foi necessário solicitar informações ao Supernodo sobre o recurso.



The screenshot shows a REST client interface with the following details:

- Request:** GET `{{(node1_url)}}/resources/de93c8f9bac8af367ee5aa0c8dc828d02ca2976ffb6e37a6d27bf420f318a68a`
- Response:** Status: 200 OK, Time: 5 ms, Size: 274 B
- Response Body (JSON):**

```
1 {
2   "data": {
3     "id": "de93c8f9bac8af367ee5aa0c8dc828d02ca2976ffb6e37a6d27bf420f318a68a",
4     "name": "resource_ahXhUa"
5   },
6   "entity": "resource"
7 }
```

Busca de informações sobre um recurso - Supernodo

Neste exemplo foi executado uma busca de informações de um recurso no Nodo 2 mas o recurso é provido pelo Nodo 3, o que, neste caso, fez com que fosse executado uma pesquisa de recursos no Supernodo do Nodo 2 juntamente da comunicação em multicast entre os Supernodos para a descoberta da localização do recurso.

GET List resources

GET Resource info

GET List resources

+

...

Env

distr-prog-comms / Node2 / Resource info

GET

{{(node2_url)}}/resources/a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body

Cookies

Headers (4)

Test Results

Pretty

Raw

Preview

Visualize

JSON

```
1  {
2    "data": {
3      "id": "a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a",
4      "info": {
5        "location": "192.168.35.12:8001"
6      }
7    },
8    "entity": "resource"
9  }
```

Interação do Nodo 2 com o Supernodo 1 e a comunicação multicast entre os Supernodos.

```
% X1 machine.vagrant ssh host1 -- vagrant (ssh)
$ cd .
$ [INFO] *2021-05-23 17:16:45,657* werkzeug : 192.168.35.12 ~ [23/May/2021 17:15:41] "POST /api/supernode/register HTTP/1.1" 200
$ [INFO] *2021-05-23 17:16:45,618* app : searching resource (a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a) in other supernodes
[MULT-Recv] Multicast sending message (a76d292-7aee-4a49-a3f4-8c4d2a189a78;20;a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a) from (('18.8.3.15', 49682))
[MULT-Send] Multicast received (a76d292-7aee-4a49-a3f4-8c4d2a189a78;20;a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a) from myself (20;a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a)
[MULT-Recv] Multicast received (7f9a678b-deed-4add-87b7-e7a6754df566;21;192.168.35.12;8001) from (('18.8.3.15', 49168))
[MULT-Recv] Multicast is awaiting messages
[MULT-Send] Multicast message (a76d292-7aee-4a49-a3f4-8c4d2a189a78;20;a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a) was sent
[MULT-RSearch] Multicast is awaiting for search responses, file [a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a]
[MULT-RSearch] Multicast received search response from (('18.8.2.15', 49802)), (a76d292-7aee-4a49-a3f4-8c4d2a189a78;20;a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a)
Process search response (a76d292-7aee-4a49-a3f4-8c4d2a189a78;20;a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a)
[MULT-Recv] Ignoring message from myself (20;a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a)
[MULT-RSearch] Multicast is awaiting for search responses, file [a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a]
[MULT-RSearch] Multicast received search response from (('18.8.2.15', 49168)), (7f9a678b-deed-4add-87b7-e7a6754df566;21;192.168.35.12;8001)
Process search response (7f9a678b-deed-4add-87b7-e7a6754df566;21;192.168.35.12;8001)
main() received file location 192.168.35.12:8001
$ [INFO] *2021-05-23 17:16:45,622* werkzeug : 192.168.35.12 ~ [23/May/2021 17:16:05] "GET /api/supernode/search/a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a HTTP/1.1" 200
$
```

```
% X2 machine.vagrant ssh host2 -- vagrant (ssh)
(env) vagrant@host2:~/code$ make IP=192.168.35.11 PORT=8000 CONFIG=node1 r-mode
make[1]: Entering directory '/home/vagrant/code'
$ Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
$ Debug mode: off
$ node (5c85ac79-9a53-99aa-8fb6-4af181a7c032) is registered
$ [INFO] *2021-05-23 17:15:46,362* app : [-] node startup
$ [INFO] *2021-05-23 17:15:46,362* app : api startup
$ [INFO] *2021-05-23 17:15:41,608* werkzeug : * Running on http://192.168.35.11:8000/ (Press CTRL+C to quit)
$
```

```
% X3 machine.vagrant ssh host3 -- vagrant (ssh)
(env) vagrant@host3:~/code$ make IP=192.168.35.12 PORT=8000 CONFIG=node2 r-mode
make[1]: Entering directory '/home/vagrant/code'
$ Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
$ Debug mode: off
$ node (78f6dbbd-daf0-48f9-8d66-3152b46cc718) is registered
$ [INFO] *2021-05-23 17:15:41,877* app : [-] node startup
$ [INFO] *2021-05-23 17:15:41,877* app : api startup
$ [INFO] *2021-05-23 17:15:41,805* werkzeug : * Running on http://192.168.35.12:8000/ (Press CTRL+C to quit)
$ [INFO] *2021-05-23 17:16:49,424* werkzeug : 192.168.35.1 ~ [23/May/2021 17:16:03] "GET /api/node/resources/a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a HTTP/1.1" 302 -
$
```

```
% X4 -Documents\College\3Semester\distributed_prog\assign2\distrprog-node-comms\ns3\machine -- fab (fab)
(env) vagrant@host1:~/code$ make IP=192.168.35.18 PORT=5881 GROUP_SIZE=2 CONFIG=supernode3 r-super
make[1]: Entering directory '/home/vagrant/code'
$ Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
$ Debug mode: off
[MULT-Recv] Multicast is awaiting messages
$ [INFO] *2021-05-23 17:15:39,684* app : api startup
$ [INFO] *2021-05-23 17:15:39,787* werkzeug : * Running on http://192.168.35.18:5881/ (Press CTRL+C to quit)
$ [INFO] *2021-05-23 17:15:42,473* werkzeug : 192.168.35.12 ~ [23/May/2021 17:16:05] "GET /api/supernode/register HTTP/1.1" 200
$ [INFO] *2021-05-23 17:15:42,473* werkzeug : 192.168.35.12 ~ [23/May/2021 17:15:42] "POST /api/supernode/register HTTP/1.1" 200
$ [MULT-Recv] Multicast received (a76d292-7aee-4a49-a3f4-8c4d2a189a78;20;a165db4b039ff22779f88e8e847a938fcbab8229769c5ce81111c88cfc8cbb85a) from (('18.8.3.15', 49682))
[MULT-Send] Multicast sending message (7f9a678b-deed-4add-87b7-e7a6754df566;21;192.168.35.12;8001)
[MULT-Send] Multicast message (7f9a678b-deed-4add-87b7-e7a6754df566;21;192.168.35.12;8001) was sent
[MULT-Recv] Multicast is awaiting messages
[MULT-Send] Multicast received (7f9a678b-deed-4add-87b7-e7a6754df566;21;192.168.35.12;8001) from (('18.8.3.15', 49168))
[MULT-Recv] Multicast is awaiting messages
$
```

```
distrprog-node-comms\ns3\machine on } master [1] via v2.2.16 took 6s
$
```

```
% X5 machine.vagrant ssh host5 -- vagrant (ssh)
(env) vagrant@host5:~/code$ make IP=192.168.35.12 PORT=8001 CONFIG=node3 r-mode
make[1]: Entering directory '/home/vagrant/code'
$ Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
$ Debug mode: off
$ node (c6d4dbde-388f-455c-a7bb-3916ff1ee464) is registered
$ [INFO] *2021-05-23 17:15:42,473* app : [-] node startup
$ [INFO] *2021-05-23 17:15:42,473* app : api startup
$ [INFO] *2021-05-23 17:15:42,562* werkzeug : * Running on http://192.168.35.12:8001/ (Press CTRL+C to quit)
$
```