Agents Name Service

Introduction

This file can be run on any platform supporting Python, with the necessary install permissions. This example shows how to set up the Agents Name Service contract using theuagents andcosmpy Python libraries. TheFetch.ai Name Service Smart Contract aims at enhancing the usability and accessibility of the Fetch.ai blockchain by providing a decentralized, secure, and user-friendly way to manage names for various digital entities. Indeed, the Name Service Smart Contract acts similarly to a phonebook for the Fetch.ai blockchain; It assigns memorable names (i.e., domains) to blockchain addresses, making it easier to find and interact with other agents and resources on the network. Imagine it as a way to give user-friendly names to complex wallet addresses.

In this guide we set up a communication line between two agents, Alice and Bob, where Alice sends a message to Bob every 5 seconds by first constructing Bob's address using a predefined domain (example.agent) and Bob's name (bob-0).

In turn, Bob is coded to listen for incoming messages and log them. On startup, Bob registers its name on a blockchain network using a name service contract. This ensures that Alice can address messages to Bob correctly.

Let's get started!

Walk-through

First of all, we need to create 2 Python scripts for our two agents using the following commands within your terminal:

Bob:

windows echo .

bob . py Alice:

windows echo .

alice . py

Bob

Let's start with the first agent of this example, Bob.

1. First of all, we import the required libraries and modules:

Self hosted bob.py from cosmpy . aerial . wallet import LocalWallet from uagents import Agent , Context , Model from uagents . network import get_faucet , get_name_service_contract * LocalWallet * : From thecosmpy.aerial.wallet * module, this is used to create and manage wallets for blockchain interactions. * Agent * ,Context * ,Model * : From theuagents * module, these are used to define the agent, its context, and the message model. * get_faucet * : From theuagents.network * module, this function is used to get a faucet for funding wallets. * get_name_service_contract * : From theuagents.network * module, this function is used to get the contract for registering agent names on the network.

1. We then define the Message model defining the structure of messages that Bob will be able to handle:

Self hosted bob.py class

Message (Model): message :

str We have defined aMessage data model which contains a single attributemessage of type string.

1. We then need to initialize the agent:

Self hosted bob.py bob =

Agent (name = "bob-0" , seed = "agent bob-0 secret phrase" , port = 8001 , endpoint = ["http://localhost:8001/submit"],) We have created an agent namedbob-0 with a specifiedseed . The agent listens on port8001 and has anendpoint for receiving messages.

1. We are now ready to set up theWallet
2. ,Name Service Contract
3. , andFaucet
4. :

Self hosted bob.py my_wallet = LocalWallet . from_unsafe_seed ("registration test wallet") name_service_contract =

get_name_service_contract (test = True) faucet =

get_faucet ()

DOMAIN

"example.agent"

faucet . get_wealth (my_wallet. address ()) We first create a wallet from a seed phrase and this wallet is used for interacting with the Fetch.ai blockchain. We then proceed and define the Name Service Contract. It retrieves the name service contract to register the agent's name on the blockchain. Thetest=True parameter indicates this is atest setup. Then, we define the faucet used to fund the wallet with test tokens. We then define the domain for the agent's name registration and finally proceed to request funds from the faucet to ensure the agent's wallet has sufficient funds to operate. Remember that you need to provide thename ,seed ,port ,endpoint andDOMAIN parameters to correctly run this code!

1. We continue and define Bob's functions. We start with a register_agent_name function which registers Bob's name within the blockchain using the agent's wallet, address, name and domain parameters:

Self hosted bob.py @bob . on_event ("startup") async

def

register_agent_name (ctx : Context): await name_service_contract . register (bob.ledger, my_wallet, bob.address, bob.name, DOMAIN) Here, we defined theregister_agent_name function. It registers the agent's name on the blockchain when the agent is initialized. The function uses the name service contract to registerbob-0.example.agent agent using themy_wallet wallet.It is important that you provide only lower case letters for your agent'sname andDOMAIN parameters as otherwise you will face issues the moment you run your agent .

1. We now define amessage_handler
2. function for Bob to handle incoming messages:

Self hosted bob.py @bob . on_message (model = Message) async

def

message_handler (ctx : Context ,

sender :

str ,

msg : Message): ctx . logger . info (f "Received message from { sender } : { msg.message } ")

if

name

==

"**main**" : bob . run () Here we defined a function handling incoming messages of typeMessage . It logs the sender's address and the message content.

1. Finally, we save and run the script.

The overall script for this example should look as follows:

Self hosted bob.py from cosmpy . aerial . wallet import LocalWallet from uagents import Agent , Context , Model from uagents . network import get_faucet , get_name_service_contract

NOTE : Run sender_agent.py before running receiver_agent.py

class

Message (Model): message :

str

bob

Agent (name = "bob-0" , seed = "agent bob-0 secret phrase" , port = 8001 , endpoint = ["http://localhost:8001/submit"],)

my_wallet

LocalWallet . from_unsafe_seed ("registration test wallet") name_service_contract =

get_name_service_contract (test = True) faucet =

get_faucet ()

DOMAIN

"example.agent"

faucet . get_wealth (my_wallet. address ())

@bob . on_event ("startup") async

def

register_agent_name (ctx : Context): await name_service_contract . register (bob.ledger, my_wallet, bob.address, bob.name, DOMAIN)

@bob . on_message (model = Message) async

def

message_handler (ctx : Context ,

sender :

str ,

msg : Message): ctx . logger . info (f "Received message from { sender } : { msg.message } ")

if

name

==

"**main**" : bob . run ()

Alice

Let's now define the code for our second agent, Alice.

1. First of all, we import the required libraries and modules:

Self hosted alice.py from uagents import Agent , Context , Model 1. Let's then define the message data model similarly to what we did for Bob:

Self hosted alice.py class

Message (Model): message :

str 1. We proceed and initialize the agent:

Self hosted alice.py alice =

Agent (name = "alice-0" , seed = "agent alice-0 secret phrase" , port = 8000 , endpoint = ["http://localhost:8000/submit"],) We initialize an agent namedalice-0 with a specifiedseed . The agent listens on port8000 and has anendpoint for submitting messages.

1. We then need to define the domain of the agent. TheDOMAIN
2. specifies the domain for the agent communication. This domain is used to construct the full address of Bob:

Self hosted alice.py message :

str

alice

Agent (name = "alice-0" , seed = "agent alice-0 secret phrase" , port = 8000 , endpoint = ["http://localhost:8000/submit"],)

DOMAIN

"example.agent" 1. Finally, we define the functions and behaviours for Alice:

Self hosted alice.py @alice . on_interval (period = 5) async

def

alice_interval_handler (ctx : Context): bob_name =

"bob-0"

+

"."

+ DOMAIN ctx . logger . info (f "Sending message to { bob_name } ...") await ctx . send (bob_name, Message (message = "Hello there bob."))

if

name

==

"**main**" : alice . run () Thisalice_interval_handler() function runs at regular intervals of 5 seconds to send messages. The handler constructs Bob's address using the domain and sends a message to Bob. It combinesbob-0 name with theDOMAIN to formbob-0.example.agent . The agent logs the action and sends a message with the content "Hello there bob." to Bob agent.

Remember that you need to provide thename ,seed ,port ,endpoint andDOMAIN parameters to correctly run this code!Additionally, it is important that you provide only lower case letters for your agent'sname andDOMAIN parameters as otherwise you will face issues the moment you run your agent .

The overall script for this example should look as follows:

Self hosted alice.py from uagents import Agent , Context , Model

class

Message (Model): message :

str

alice

Agent (name = "alice-0" , seed = "agent alice-0 secret phrase" , port = 8000 , endpoint = ["http://localhost:8000/submit"],)

DOMAIN

"example.agent"

@alice . on_interval (period = 5) async

def

alice_interval_handler (ctx : Context): bob_name =

"bob-0"

+

"."

+ DOMAIN ctx . logger . info (f "Sending message to { bob_name } ...") await ctx . send (bob_name, Message (message = "Hello there bob."))

if

name

==

"**main**" : alice . run ()

Expected output

Within your terminal windows you should see something similar to the following:

1. Bob
2. :
3. INFO: [bob-0]: Registering on almanac contract...
4. INFO: [bob-0]: Registering on almanac contract...complete
5. INFO: [network]: Registering name...
6. INFO: [network]: Registering name...complete
7. INFO: [bob-0]: Starting server on http://0.0.0.0:8001 (Press CTRL+C to quit)
8. INFO: [bob-0]: Received message from agent1qwquu2d237gntfugrnwch38g8jkl76vdr05qjm4wyps6ap04fvt8vtzhpqw: Hello there bob.
9. INFO: [bob-0]: Received message from agent1qwquu2d237gntfugrnwch38g8jkl76vdr05qjm4wyps6ap04fvt8vtzhpqw: Hello there bob.
10. INFO: [bob-0]: Received message from agent1qwquu2d237gntfugrnwch38g8jkl76vdr05qjm4wyps6ap04fvt8vtzhpqw: Hello there bob.
11. INFO: [bob-0]: Received message from agent1qwquu2d237gntfugrnwch38g8jkl76vdr05qjm4wyps6ap04fvt8vtzhpqw: Hello there bob.
12. Alice
13. :
14. INFO: [alice-0]: Registering on almanac contract...
15. INFO: [alice-0]: Registering on almanac contract...complete
16. INFO: [alice-0]: Sending message to bob-0.agent...
17. INFO: [alice-0]: Starting server on http://0.0.0.0:8000 (Press CTRL+C to quit)
18. INFO: [alice-0]: Sending message to bob-0.agent...
19. INFO: [alice-0]: Sending message to bob-0.agent...
20. INFO: [alice-0]: Sending message to bob-0.agent...

Last updated on October 17, 2024

On This Page

- Introduction
- Walk-through
- Bob
- Alice
- Expected output
- Edit this page on github(opens in a new tab)