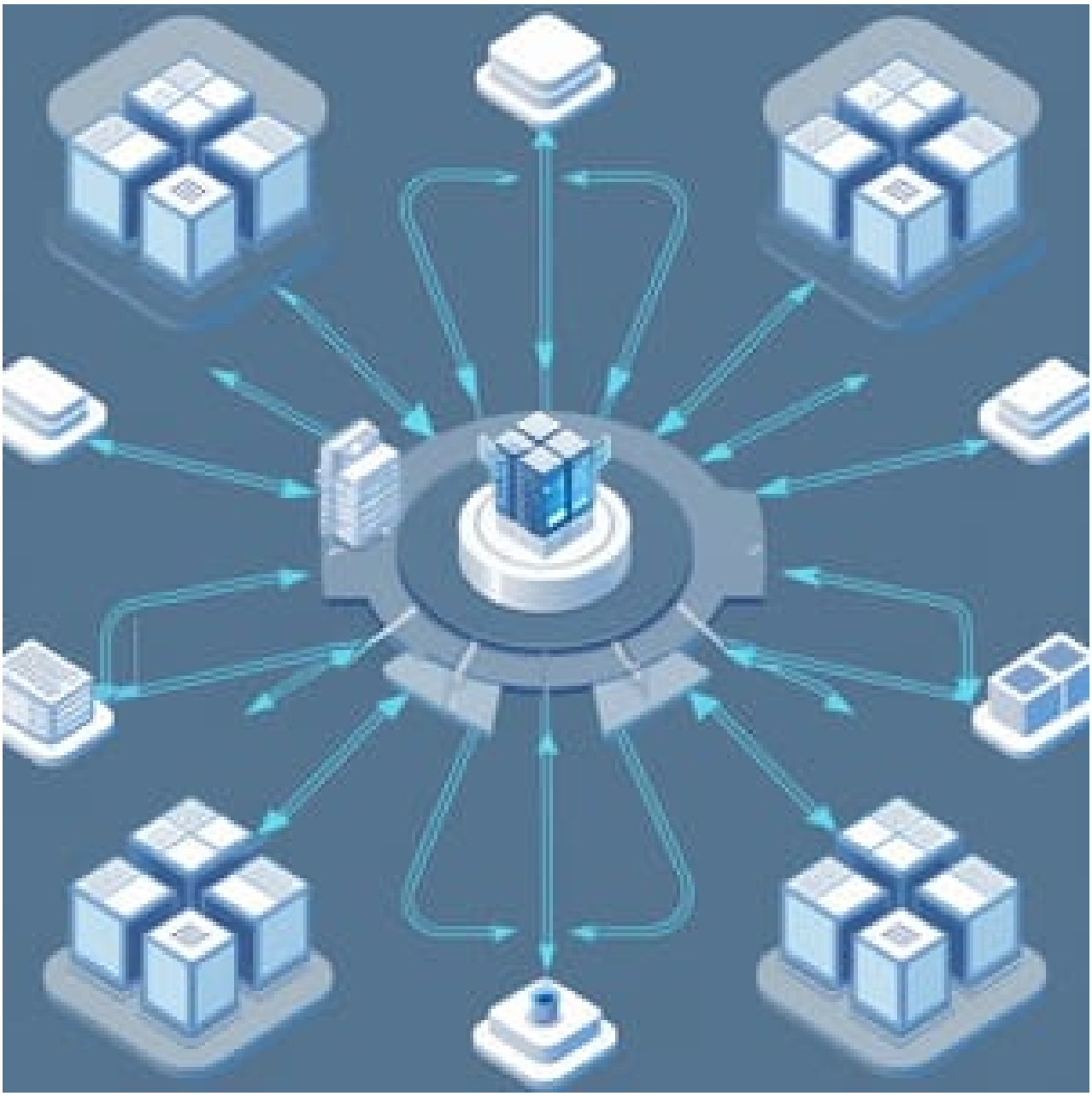
How to Self-Host the External Coordinator (EC)?

mohamedawn.com/how-to-self-host-ec

Mohamed Awnallah August 15, 2024



Get a VM in the Cloud

To successfully host the EC, you need to set up a virtual machine (VM) in the cloud. This process involves choosing a cloud provider, configuring the VM with suitable specifications, and ensuring proper network and security settings. The following steps outline the process to get your VM up and running effectively.

Choose a Cloud Provider

Selecting the right cloud provider is crucial for meeting your performance, cost, and geographical needs. Evaluate options such as Hetzner Cloud, AWS, Microsoft Azure, or Google Cloud based on their pricing, features, and support. Choose a provider that aligns with your requirements for running the EC.

Choosing VM Specifications

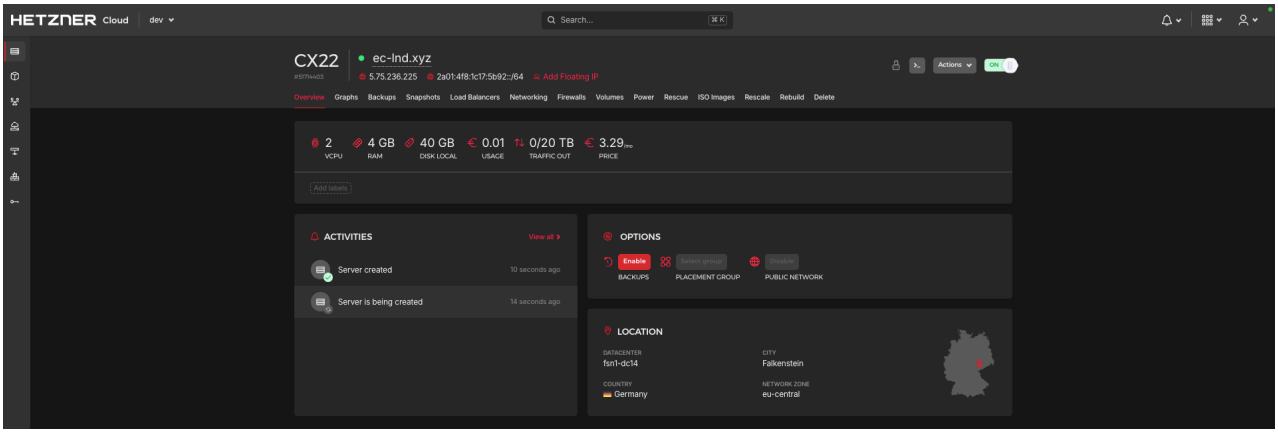
To host the EC effectively, you'll need to select a virtual machine (VM) with adequate resources. For this setup, you have a couple of options depending on your anticipated workload:

1. **Basic Configuration**:
    - **2 vCPUs**
    - **4 GB RAM**
    - **40 GB SSD**

This configuration is suitable for smaller deployments or for initial testing. It’s cost-effective, with an approximate monthly cost of $3.50 on Hetzner Cloud.

2. **Recommended Configuration**:
    - **4 vCPUs**
    - **8 GB RAM**
    - **160 GB SSD**

If you expect moderate to heavy usage, this configuration is more appropriate. It provides better performance and reliability, with an approximate cost of $14 per month on Hetzner Cloud. The choice of configuration should align with the number of clients and interactions expected with the EC.



Basic Configuration Setup

Selecting an Operating System and Network Configuration

For this setup, we’ll use **Ubuntu 22.04 LTS**. It’s a stable and widely supported version of Linux that is compatible with most software and tools you’ll be using.

- **Operating System**: Ubuntu 22.04 LTS
- **IPv4 and IPv6**: Ensure the VM is configured with both IPv4 and IPv6 addresses to handle diverse network requests.
- **SSH Access**: Set up SSH to allow remote access. This is crucial for managing the VM and deploying your EC.

Accessing the VM via SSH

Once your VM is provisioned with the desired specifications and configured with Ubuntu 22.04, you can access it using SSH. Follow these steps:

Obtain SSH Credentials

During VM creation, you generated a private SSH key (e.g., `id_rsa_ec_hetzner`) and uploaded the corresponding public key to Hetzner.

Access the VM

Open your terminal or command prompt and use the following command to connect to your VM, replacing `root@5.75.236.225` with the appropriate IP address of your VM:

```
$ ssh -i ~/.ssh/id_rsa_ec_hetzner root@5.75.236.225
```

The `-i` flag specifies the path to your private key, used to authenticate your connection. After running the command, you will be prompted for the passphrase if one was set during key generation. Upon successful authentication, you will be logged into your VM as the root user.

Now that you have access to your VM, you can proceed with setting up the necessary environment and software to host the EC.

Setup the VM Environment

After setting up your VM, create a new user with sudo privileges, update the system, and install Docker for container management.

Create a New User with Sudo Privileges

To improve security and manage your VM more effectively, it is best practice to avoid operating as the root user for daily tasks. Instead, create a new user with sudo privileges:

Add a New User

Log into your VM via SSH as the root user and execute the following command to create a new user, replacing `ecuser` with your preferred username:

```
$ sudo adduser ecuser
```

Follow the prompts to set a password and optionally provide additional information for the user.

Grant Sudo Privileges

Add the new user to the sudo group to grant administrative privileges:

```
$ sudo usermod -aG sudo ecuser
```

This command appends the user `ecuser` to the sudo group, allowing them to execute commands with elevated privileges using `sudo`.

Switch to the New User

After creating and configuring the new user, switch to this user to continue with the setup:

```
$ su - ecuser
```

This command will log you in as `ecuser` and start a new shell session with the user's environment settings.

Update Repositories and Packages

To ensure you have the latest security patches and software improvements, keep your system up to date:

Update Package Lists

Run the following command to update the list of available packages and their versions:

```
$ sudo apt-get update
```

Upgrade Installed Packages

To upgrade all installed packages to their latest versions, execute:

```
$ sudo apt-get upgrade -y
```

The `-y` flag automatically confirms the upgrade process, applying updates without additional prompts.

Install Docker

Docker is essential for running the EC in a containerized environment. Here's how to install Docker on your VM:

Install Docker

Docker provides a step-by-step installation guide. You can follow their official documentation for the most current instructions.

Manage Docker as a Non-Root User

By default, Docker commands need to be run with `sudo`. To allow `ecuser` to run Docker commands without `sudo`, create a Docker group and add the user to it:

```
$ sudo groupadd docker
$ sudo usermod -aG docker ecuser
```

Apply the new group membership by logging out and back in, or use the following command:

```
$ newgrp docker
```

Verify Docker Installation

Ensure Docker is correctly installed and running by executing:

```
$ docker run hello-world
```

This command runs a test Docker container and verifies that Docker is set up properly.

Prepare the EC Environment

To prepare the EC environment, start by cloning the Distributed Mission Control for LND (DMC-for-LND) repository, which contains essential code and configuration files. Build the Docker image for the EC from this repository, then configure the `.ec` directory with necessary configuration files and generate self-signed TLS certificates. Finally, run the EC Docker container, verify its operation, and ensure all configurations are correctly applied.

Clone the DMC for LND Repo

To set up the EC, you need to clone the repository containing the necessary code and configuration files:

Clone the Repository

Ensure you are logged in as the user with appropriate permissions (e.g., `ecuser`), then use the following command to clone the repository:

```
$ git clone https://github.com/ziggie1984/Distributed-Mission-Control-for-LND.git
```

Navigate into the cloned directory:

```
$ cd Distributed-Mission-Control-for-LND
```

Build the EC Docker Image

With the repository cloned, you'll need to build the Docker image for the EC:

Build the Docker Image

Ensure Docker is running on your VM and run the following command at the root directory of the `DMC-for-LND` repo:

```
$ docker build -t ec-image .
```

This command creates a Docker image named `ec-image` from the Dockerfile in the specified directory.

Configure the .ec Directory

The `.ec` directory contains configuration files necessary for the EC to function correctly:

Create and Configure the .ec Directory

Create a `.ec` directory in your home folder:

```
$ mkdir ~/.ec
```

Copy the `sample-ec.conf` configuration file found in the root directory of the `DMC-for-LND` repo to `~/.ec/ec.conf`:

```
$ cp ./sample-ec.conf ~/.ec/ec.conf
```

Generate and Configure Self-Signed TLS Certificates

To secure communication, you'll need TLS certificates. For testing purposes, self-signed certificates can be used. Follow these steps:

Add IPv4 and IPv6 Addresses to the OpenSSL Configuration

First, add the IPv4 and IPv6 addresses of the `eth0` network interface to the OpenSSL configuration file. This allows external clients to interact with the EC using the shared self-signed certificate.

```
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 5.75.236.225  netmask 255.255.255.255  broadcast 0.0.0.0
    inet6 2a01:4f8:1c17:5b92::1  prefixlen 64  scopeid 0x0<global>
```

Append the IPv4 and IPv6 addresses to the `openssl-self-signed.conf` file in the root directory of `DMC-for-LND` repo:

```
$ echo -e "IP.3   = 5.75.236.225" | tee -a openssl-self-signed.conf
$ echo -e "IP.4   = 2a01:4f8:1c17:5b92::1" | tee -a openssl-self-signed.conf
```

Generate the Private Key and Certificate Signing Request (CSR)

Create the private key for the SSL/TLS certificate:

```
$ openssl ecparam -name prime256v1 -genkey -noout -out tls.key
```

Generate the CSR using the configuration file:

```
$ openssl req -new -key tls.key -out server.csr -config openssl-self-signed.conf
```

Generate the Self-Signed Certificate

Create the self-signed certificate using the CSR and private key:

```
$ openssl x509 -req -in server.csr -days 365 -signkey tls.key -out tls.cert -
extensions req_ext -extfile openssl-self-signed.conf
```

Copy the Certificates to the Configuration Directory

Move the generated certificate and key to the `~/.ec` directory:

```
$ cp tls.cert tls.key ~/.ec
```

Adjust Permissions for Docker Container

Before running the EC container, change the ownership of the `~/.ec` directory to the user ID specified in the Dockerfile (typically 2000):

```
$ sudo chown -R 2000:2000 ~/.ec
```

This step ensures that the Docker container can access the certificate files with the correct permissions.

Note: If running EC without Docker (e.g., using `make` and `ec` directly), the self-signed certificate is generated and configured automatically. These steps are specifically for Docker environments due to their isolated network stack.

Run the EC Docker Container

To start the Docker container for the EC, use the following command:

```
$ docker run -d -p 50050:50050 -p 127.0.0.1:6060:6060 -p 8081:8081 \
-v /home/ecuser/.ec:/home/ecuser/.ec --name ec-daemon-container ec-daemon
```

This command runs the `ec-image` Docker image as a detached container named `ec-daemon-container`, maps necessary ports, and mounts the `~/.ec` directory to `~/.ec` in the container.

Verify the Container is Running

Check the status of your Docker container:

```
$ docker ps
```

Inspect logs for any issues:

```
$ docker logs ec-daemon-container
```

You should see log entries similar to:

```
$ docker logs -f ec-daemon-container
time="2024-08-14T05:17:30Z" level=info msg="Logging setup complete"
time="2024-08-14T05:17:30Z" level=info msg="Database setup complete"
time="2024-08-14T05:17:30Z" level=info msg="TLS configurations loaded"
time="2024-08-14T05:17:30Z" level=info msg="Cleanup routine started to remove
stale mission mission control data from the database on an interval of: 24 hours,
0 minutes, 0 seconds"
time="2024-08-14T05:17:30Z" level=info msg="Running cleanup routine to remove
stale mission control data from the database..."
time="2024-08-14T05:17:30Z" level=info msg="Cleanup routine completed successfully
and 0 pairs were removed"
time="2024-08-14T05:17:30Z" level=info msg="Starting pprof server on
https://localhost:6060"
time="2024-08-14T05:17:30Z" level=info msg="Starting gRPC server on
https://[::]:50050"
time="2024-08-14T05:17:30Z" level=info msg="Starting HTTP/1.1 REST server on
https://[::]:8081"
```

With these steps completed, your self-signed TLS certificates are generated and configured, and the EC Docker container is up and running with the necessary settings.

Test the EC Setup

After configuring and starting the EC Docker container, you'll want to verify that everything is working correctly by performing light stress tests and ensuring the EC is functioning as expected. Follow these steps:

Clone the Stress Testing Repository

First, clone the Stress Testing repo containing the stress testing clients:

```
$ git clone https://github.com/mohamedawnallah/Stress-Testing-Distributed-Mission-
Control-EC.git
$ cd Stress-Testing-Distributed-Mission-Control-EC
```

Set Up Python Virtual Environment

Install python virtual environment if not already installed:

```
$ sudo apt install python3.10-venv
```

Create and activate a Python virtual environment to manage dependencies:

```
$ python3.10 -m venv .venv
$ source .venv/bin/activate
```

Install the required packages:

```
$ pip install -r requirements.txt
```

Modify REST Stress Client Configuration for Testing

In the `client_rest.py` script, update the `server_url` to use your EC instance's IPv4 address. In our case replacing `<your_ec_domain>` with `5.75.236.225`:

```
server_url = "https://5.75.236.225:8081"
session = get_insecure_session()
```

Note: Using an insecure session is only recommended for non-production environments as it makes the connection vulnerable to man-in-the-middle attacks.

Run Stress Tests with Insecure Session using HTTP/REST

Execute the stress test script to generate concurrent requests:

```
$ python client_rest.py
```

You should see output similar to:

```
Total Register Requests: 8, Failed Register Requests: 0, Register Failure Rate:
0.0000
Total Query Requests: 4, Failed Query Requests: 0, Query Failure Rate: 0.0000
Mission Control Entries Registered: 24
Mission Control Entries per Register: 3
```

Check the EC Docker container logs to ensure requests are being processed correctly:

```
$ docker logs -f ec-daemon-container
```

You should observe logs indicating that requests were received and processed:

```
time="2024-08-14T05:42:48Z" level=info msg="Received QueryAggregatedMissionControl
request"
time="2024-08-14T05:42:48Z" level=info msg="Received RegisterMissionControl
request with 3 pairs"
...
```

Stress Test with Self-Signed Certificate

To test with the self-signed certificate, first copy the `tls.cert` file from `~/.ec` in the host or in the Docker container to the `Stress-Testing-Distributed-Mission-Control-EC` directory:

```
$ cp ~/.ec/tls.cert ~/Stress-Testing-Distributed-Mission-Control-EC/
```

Update the `client_rest.py` or `client_rpc.py` script to use the self-signed certificate:

```
session = get_self_signed_session(cert="tls.cert")
```

Run the stress test script for REST/HTTP:

```
$ python client_rest.py
```

You should see output like:

```
Total Register Requests: 7, Failed Register Requests: 0, Register Failure Rate:
0.0000
Total Query Requests: 5, Failed Query Requests: 0, Query Failure Rate: 0.0000
Mission Control Entries Registered: 21
Mission Control Entries per Register: 3
```

Similarly you could run the stress test script for gRPC:

```
$ python client_rpc.py
```

This confirms that the TLS handshake with the self-signed certificate succeeded. **Note:** Self-signed certificates are suitable for testing or development but are not recommended for production environments.

Configure Trusted SSL/TLS Certificates

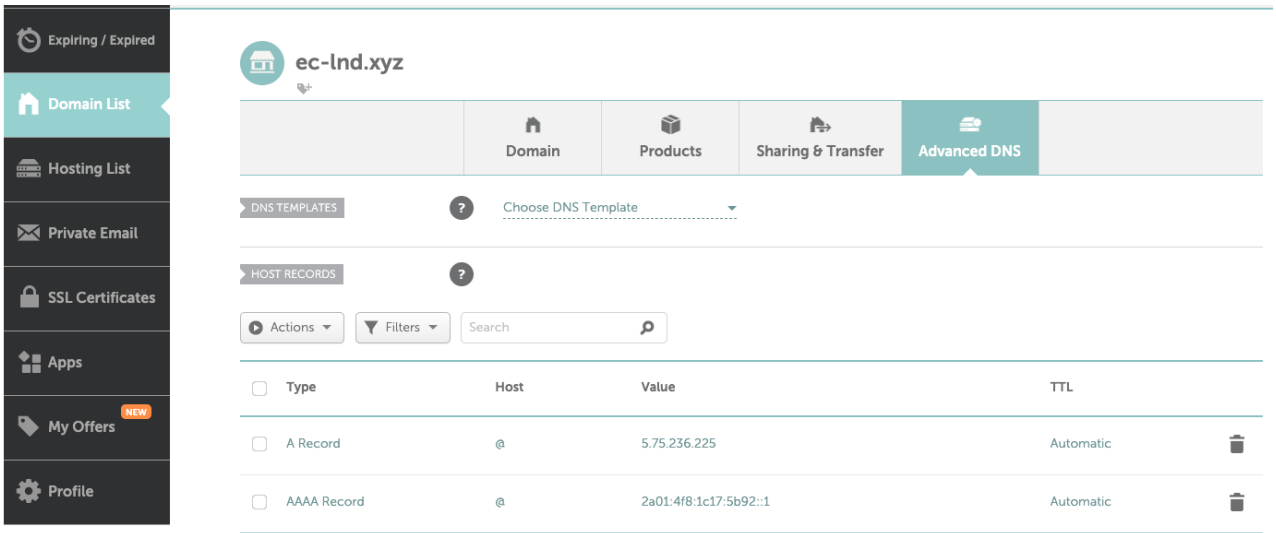
To secure your EC instance with trusted SSL/TLS certificates from a Certificate Authority (CA) like Let's Encrypt, follow these steps. This process includes registering a domain, configuring DNS records, installing Certbot, and updating your Docker container to use the new certificates.

Register a Domain

Choose a domain name and register it with a domain registrar of your choice (e.g., Namecheap, GoDaddy, Google Domains).

Configure DNS Records

Log in to your domain registrar's DNS management console. Create an `A` record pointing your domain to the public IPv4 address of your VM. Set the record type to `A`, name it `@` (or use your domain name), and enter `5.75.236.225` as the value, which is your VM's public IP. Optionally, create an `AAAA` record for IPv6 if applicable. Set the record type to `AAAA`, name it `@` (or use your domain name), and enter `2a01:4f8:1c17:5b92::1` as the value, which is your VM's public IPv6 address.



Note: DNS record propagation may take some time.

To verify DNS records, use the following commands:

Check A Record:

```
$ dig A ec-lnd.xyz
```

Example Output:

```
; <<>> DiG 9.18.28-0ubuntu0.22.04.1-Ubuntu <<>> A ec-lnd.xyz
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 51258
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;ec-lnd.xyz.                    IN      A

;; ANSWER SECTION:
ec-lnd.xyz.             1779    IN      A       5.75.236.225

;; AUTHORITY SECTION:
ec-lnd.xyz.             1779    IN      NS      dns1.registrar-servers.com.
ec-lnd.xyz.             1779    IN      NS      dns2.registrar-servers.com.

;; Query time: 0 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Wed Aug 14 12:36:55 UTC 2024
;; MSG SIZE  rcvd: 114
```

Check AAAA Record:

```
$ dig AAAA ec-lnd.xyz
```

Example Output:

```
; <<>> DiG 9.18.28-0ubuntu0.22.04.1-Ubuntu <<>> AAAA ec-lnd.xyz
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24449
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;ec-lnd.xyz.                    IN      AAAA

;; ANSWER SECTION:
ec-lnd.xyz.             1712    IN      AAAA    2a01:4f8:1c17:5b92::1

;; AUTHORITY SECTION:
ec-lnd.xyz.             1712    IN      NS      dns2.registrar-servers.com.
ec-lnd.xyz.             1712    IN      NS      dns1.registrar-servers.com.

;; Query time: 0 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Wed Aug 14 12:38:07 UTC 2024
;; MSG SIZE  rcvd: 126
```

Install and Configure Certbot

First, install Certbot by running the following command:

```
$ sudo apt install certbot
```

Use Certbot to obtain a certificate for your domain with the command:

```
$ sudo certbot certonly --standalone -d ec-lnd.xyz
```

Add a cron job to automatically renew the certificate. This job will run twice daily at noon and midnight:

```
$ 0 0,12 * * * certbot renew --quiet
```

Update EC Configuration for Trusted Certificates

Certbot stores your certificates in `/etc/letsencrypt/live/ec-lnd.xyz/`. Copy these certificates to your EC configuration directory using the following commands:

```
$ cp /etc/letsencrypt/live/ec-lnd.xyz/fullchain.pem ~/.ec/tls.cert
$ cp /etc/letsencrypt/live/ec-lnd.xyz/privkey.pem ~/.ec/tls.key
```

Adjust ownership of the configuration directory:

```
sudo chown -R 2000:2000 ~/.ec/third_party_tls/
```

Update the `~/.ec/ec.conf` file with the following fields:

```
third_party_tls_cert_file = tls.cert
third_party_tls_key_file = tls.key
tls_domain_name = ec-lnd.xyz
```

Test with Let's Encrypt Certificates

Restart the Docker container to apply the new certificate:

```
$ docker restart ec-daemon-container
```

Update the `client_rest.py` script in the Stress Testing repository to use the new certificate. Set the `server_url` to your domain and use the trusted CA session:

```
server_url = "https://ec-lnd.xyz:8081"
session = get_trusted_ca_session()
```

Run the stress test script to ensure successful communication through the EC.

Additionally, test with `client_rpc.py` from the Stress Testing repository by setting:

```
server_url = "ec-lnd.xyz:50050"
channel = get_trusted_ca_channel(server_url)
```

This confirms that the EC is properly configured and operational with trusted Let's Encrypt certificates.

Integrate with LND

To integrate your EC instance with the LND daemon using the Distributed Mission Control for LND repository, follow these steps to register LND mission control data with EC, verify the integration, and update both EC and LND with the new data.

Prerequisites

Ensure your LND daemon is running in regtest mode. For setup instructions, refer to the LND installation documentation (Recommended) or the Bootstrap Lightning Network repository. Verify that your LND node has mission control data available for registration with EC by executing the following command:

```
$ lncli --network=regtest querymc
```

Sample Output:

```
{
    "pairs":  [
        {
            "node_from":
"03e422e044b72fd4397cdf4a58d4368eeba106e361dd37407da4975c85c368a450",
            "node_to":
"03dbd084631519fcaccfd3a72cad4a65f42d808b86b7e362629aa221d097cb0f71",
            "history":  {
                "fail_time":  "0",
                "fail_amt_sat":  "0",
                "fail_amt_msat":  "0",
                "success_time":  "1723675118",
                "success_amt_sat":  "250",
                "success_amt_msat":  "250000"
            }
        }
    ]
}
```


Install Python Client Dependencies

To get started, first install the necessary Python client dependencies by running:

```
$ python install_python_client_dependencies.py
```

Then, activate the virtual environment:

```
$ source .ecrpc_client/bin/activate
```

Register LND's Mission Control Data with EC

Configure Client for Registration

Set the `LND_REST_HOST`, `LND_DIR`, and `EC_REST_HOST` variables to match your setup:

```
LND_REST_HOST = 'localhost:8080'
LND_DIR = "/home/ecuser/.lnd"
EC_REST_HOST = "https://ec-lnd.xyz:8081"
```

Register Mission Control Data with EC

In `client_rest.py`, comment out the `import_mission_control_data_from_ec_to_my_lnd` function call since you are testing the registration of data from LND to EC. Then, execute the script:

```
$ python client_rest.py
```

Expected Output:

```
1 of your LND Mission Control pairs registered into EC 🎉
```

To confirm that the data was received and processed correctly, check the EC logs:

```
$ docker logs -f ec-daemon-container
```

Sample Log Output:

```
time="2024-08-14T22:49:16Z" level=info msg="Received RegisterMissionControl
request with 1 pairs"
time="2024-08-14T22:49:16Z" level=info msg="1 pairs were processed and stored
successfully"
```

Import Mission Control Data from EC to LND

In `client_rest.py`, comment out the `register_my_lnd_mission_control_data_with_ec` function call since you are testing the importing of data from EC to LND. Then, execute the script:

```
$ python client_rest.py
```

Expected Output:

```
37 EC Mission Control pairs imported into your LND 🎉
```

Check the mission control data in LND to ensure it has been imported correctly:

```
$ lncli --network=regtest querymc
```

Sample Output:

```
{
    "pairs":  [
        {
            "node_from":
"036913a11ea397c80ff034e258593991b6ec51a137064d15a9ac4afb25a03cd90c",
            "node_to":
"02cd3ada00ede17b1a1be61184f827556f86d24283ab6b1f3652bf115665cfdbee",
            "history":  {
                "fail_time":  "1723425237",
                "fail_amt_sat":  "2024",
                "fail_amt_msat":  "2024000",
                "success_time":  "1723483321",
                "success_amt_sat":  "643",
                "success_amt_msat":  "643000"
            }
        },
        ...
    ]
}
```

Alternative Method: Using RPC Client

You can also perform the same operations using the `client_rpc.py` script:

```
$ python client_rpc.py
```

By following these steps, you will successfully integrate your EC instance with the LND daemon, allowing you to register and import mission control data between the two systems.

Keep the nodes connected, the channels open, and the mission control data flowing! ⚡📊

Read Also

Summer of Bitcoin 2024: Distributed Mission Control Data for LND Pathfinding Enhancement

This summer, my focus will be on a groundbreaking project that aims to enhance the Lightning Network Daemon (LND) pathfinding using Mission Control (MC) data. Here's a brief overview of what this project entails and its potential impact on the Lightning Network (LN). Project Synopsis The primary goal of

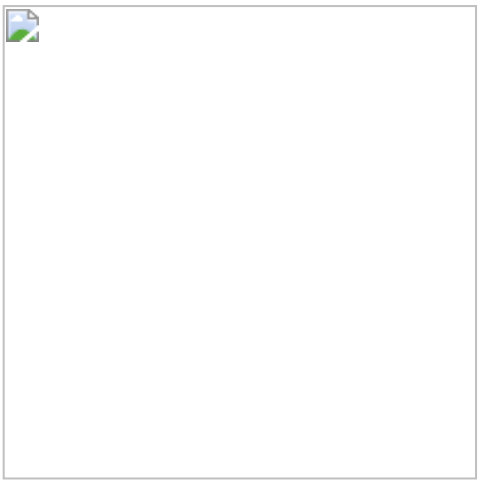
Mohamed Tech and LifeMohamed Awnallah

Additional Resources

Distributed-Mission-Control-for-LND/docs/HOW_TO_DEPLOY_EC.md at main · ziggie1984/Distributed-Mission-Control-for-LND

This project creates an external coordination service which aggregates Mission Control Data from different lightning nodes and makes the data available via an API.. - ziggie1984/Distributed-Missio…

GitHubziggie1984



Mohamed Awnallah

Aug 15, 2024
Summer of Bitcoin 2024 🌞