EnhancedEndpointTracker Documentation

Release 2.0

agccie

Mar 25, 2022

Contents:

CHAPTER 1

Introduction

The EnhancedEndpointTracker is a Cisco ACI application that maintains a database of endpoint events on a per-node basis allowing for unique fabric-wide analysis. The application can be configured to analyze, notify, and automatically remediate various endpoint events. This gives ACI fabric operators better visibility and control over the endpoints in the fabric.

Features include:

- Easy to use GUI with fast type-ahead search for any mac/ipv4/ipv6 endpoint in the fabric
- Move, offsubnet, rapid, and stale endpoint analysis
- Configurable notification options via syslog/email for various events detected by the app
- Capability to clear an endpoint on one or more nodes via GUI or API
- Distributed architecture allowing app to grow based on the size of the fabric
- Fully documented swagger API to allow for easy integration with other tools.

This application is written in python and utilizes a distributed MongoDB database to persist data. It also relies on a custom Flask framework for handling API requests, Redis for messaging between components, and Angular with CiscoUI for the frontend.

CHAPTER 2

Install

The ACI EnhancedEndpointTracker can be installed directly on the APIC as an ACI app or deployed as a standalone application hosted on a baremetal or virtual machine.

2.1 ACI Application

The application can be deployed on the APIC. There are two modes supported, both available on ACI Appcenter.

- mini is backwards compatible with APIC 2.x and 3.x. However, there are memory constraints that limit the supported scale
- full scale application supported on APIC 4.x and above.

After downloading the app, follow the directions for uploading and installing the app on the APIC:

- 2.x Install Video Example
- 2.x Install Instructions
- 3.x Install Instructions

Note: Ensure you select security domain **all** when installing on the APIC

In you are executing the `mini` app, the APIC will enforce a **2G** memory limit and a **10G** disk quota. As a result, it may crash if there are a large number of endpoints or high number events per second. As a best practice, it is recommended to deploy in `full` mode or `standalone` mode if the total number of per-node endpoints exceeds `32k`. You can determine the per-node endpoint count via the following moquery on the APIC:

```
apic# moquery -c epmDb -x query-target=subtree -x target-subtree-class=epmIpEp,
↪epmMacEp,epmRsMacEpToIpEpAtt -x rsp-subtree-include=count
```

If running `mini` mode and it is exceeding the memory limits, you may see the symptoms below:

- Consistent monitor restarts due to “subscriber no longer running”

- Monitor restart due to “worker ‘w0’ not longer active”
- Monitor stuck or restarting during “getting initial endpoint state”

2.2 Standalone Application

The `standalone` app is one that runs on a dedicated host/VM and makes remote connections to the APIC opposed to running as a container on the APIC. For large scale fabrics or development purposes, standalone is the recommended mode to run this application. The standalone app also has a few different deployment options:

- `all-in-one` is a single container with all required processes running. This is similar to `mini` mode executing on the APIC, however the memory and compute restrictions are based on the host device and therefore can support much larger scale. This is the easiest way to deploy the app as it can be started with a single command.
- `cluster` uses a distributed architecture to execute multiple container across one or more nodes. This allows the app to scale with the size of the fabric. This is similar to the `full` mode executing on the APIC but can be deployed in any custom environment that supports container orchestration.

  If you are deploying the cluster with more than one node, ensure there is connectivity between each node in the cluster and the following ports are allowed:

  - **TCP** port **2377** for cluster management
  - **TCP** and **UDP** port **7046** for communication between nodes
  - **UDP** port **4789** for overlay traffic
  - **TCP** port **22** for auto-deployment and setup

2.2.1 All-in-One Mode

To execute in `all-in-one` mode, you need a host with docker installed. See the Docker documentation for installing docker on your host. Once installed, execute the following command to download the EnhancedEndpointTracker docker image and run it:

```
host$ docker run --name ept -p 5000:443 -d agccie/enhancedendpointtracker:latest
```

The command will start an instance of EnhancedEndpointTracker with the web server running on port 5000. Login to the web UI at https://localhost:5000. See the usage section for further details regarding how to use the app.

2.2.2 Cluster Mode - OVA

The EnhancedEndpointTracker app can be deployed in a distributed cluster. Users can deploy in their own cluster or use a prebuilt OVA. This section will focus on the OVA.

Note: Please send an email to aciappcenter-support@external.cisco.com to request a temporary download link for the EnhancedEndpointTracker OVA.

The recommended sizing for the VM is as follows:

- 8 vCPU
- 16G memory
- 75G harddisk, thick provisioned

The OVA contains the following components preinstalled:

- Ubuntu 18.04.2 LTS
- OpenSSH
- Docker CE 18.09.02
- Python 2.7.15rc1
- Network manager
- EnhancedEndpointTracker docker image specific to the version of the OVA
- A copy of the EnhancedEndpointTracker source code located in */opt/cisco/src* directory

To get started with the OVA, perform the following steps:

- *Configure Host Networking*
- *Configure NTP and Timezone*
- *Configure the Cluster and Deploy the Stack*

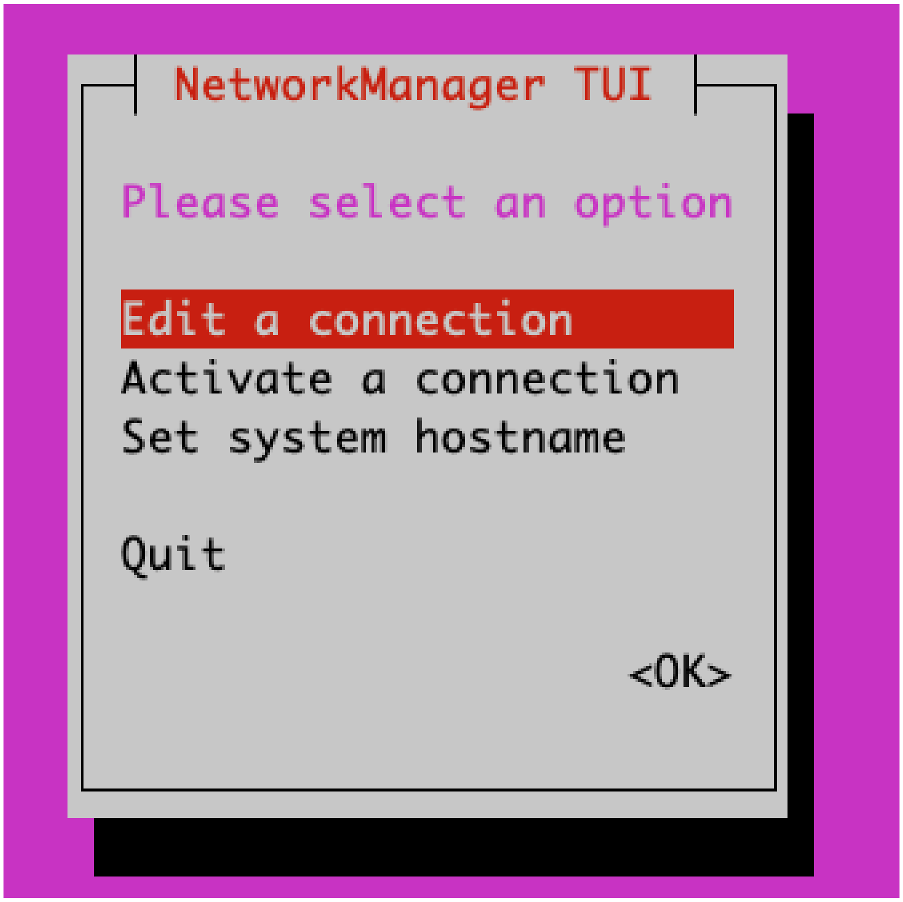
Configure Host Networking

Once the OVA is deployed, access the console with the credentials below. Note, you will be required to change the password on first login.

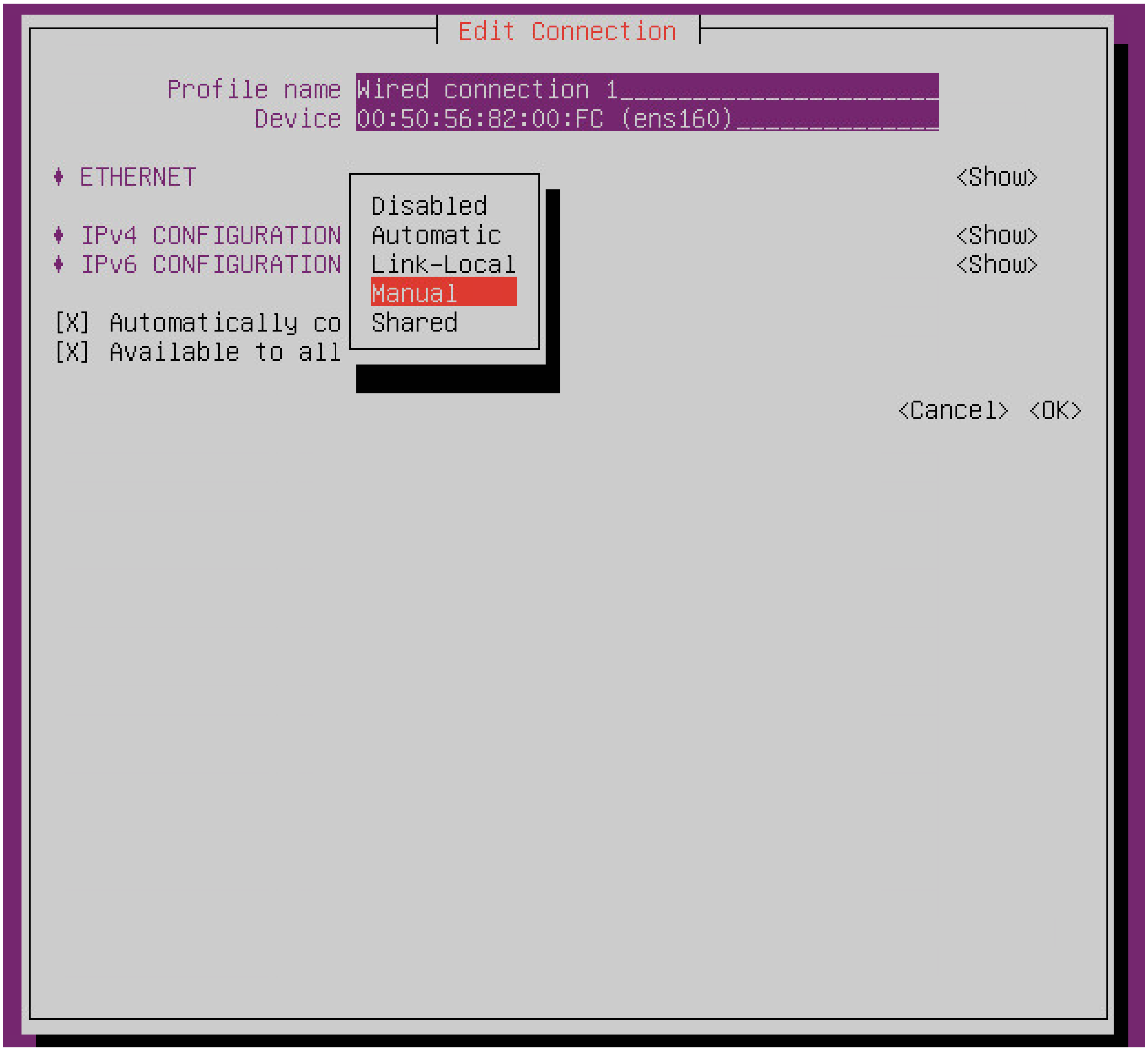
- username: **eptracker**
- password: **cisco**

The OVA is simply a Ubuntu 18.04 install. Users can use any mechanism they prefer to initialize the network. The example below uses network manager TUI which is preinstalled on the host.

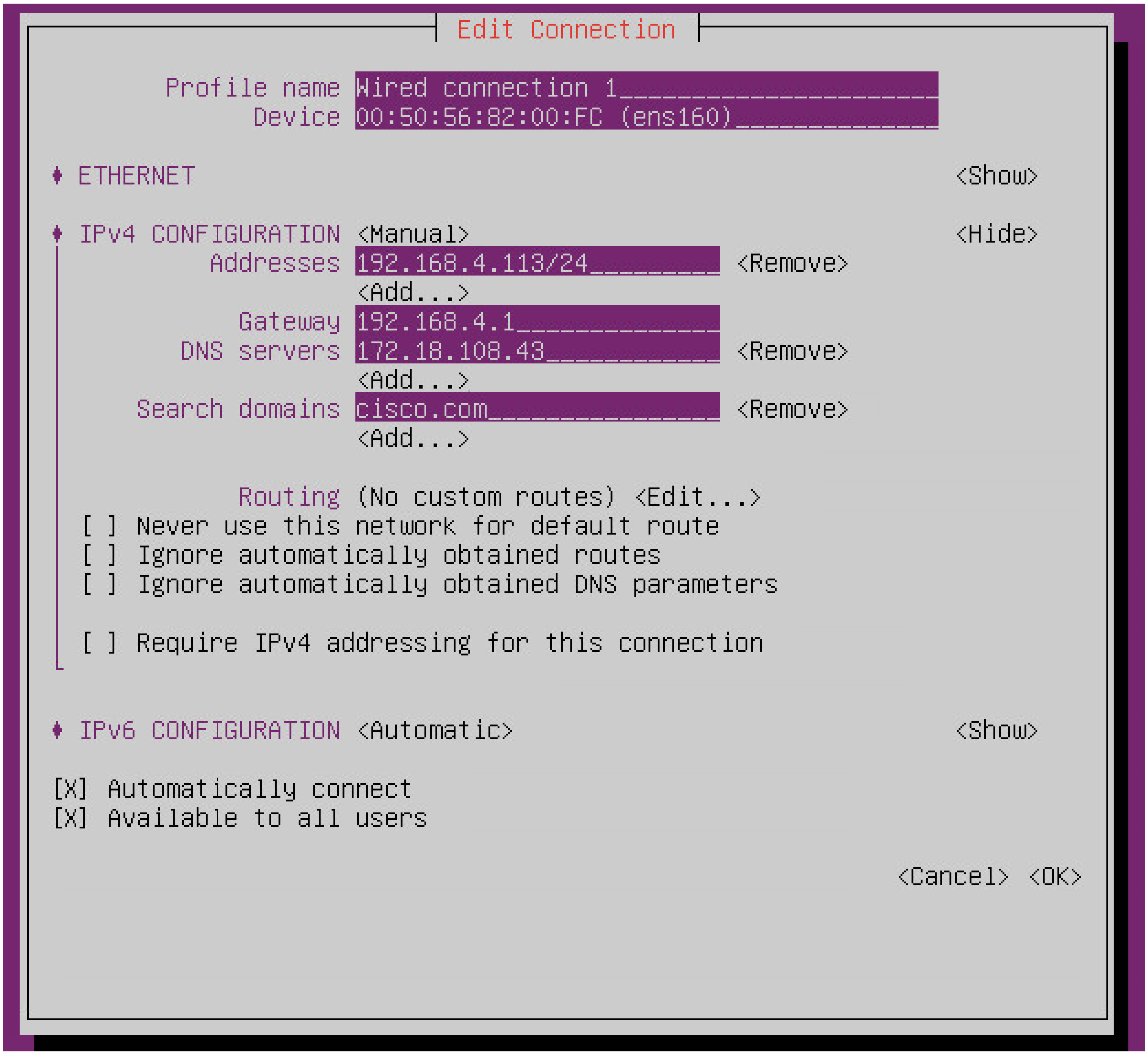
- Enter **sudo nmtui**
- Choose 'Edit a connection'



- Edit the appropriate connection. By default, the connection type is likely **Automatic** (DHCP) but if you need to set a static IP address you will need to change the mode to **Manual** and the set the appropriate info.

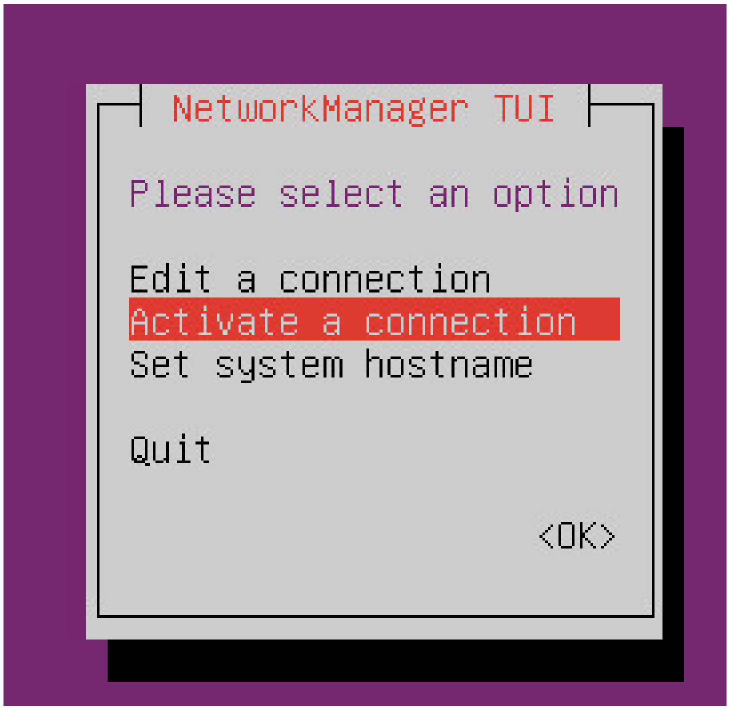


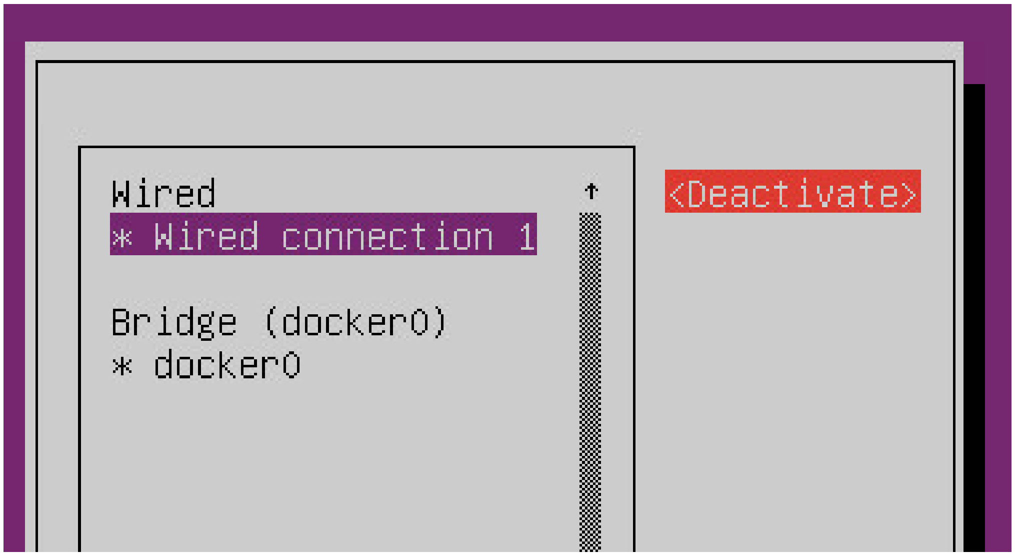
```
                        ┤ Edit Connection ├

          Profile name Wired connection 1______________________
                Device 00:50:56:82:00:FC (ens160)______________

 ♦ ETHERNET                                                    <Show>
                       ┌──────────────┐
                       │ Disabled     │
 ♦ IPv4 CONFIGURATION  │ Automatic    │                        <Show>
 ♦ IPv6 CONFIGURATION  │ Link-Local   │                        <Show>
                       │ Manual       │
 [X] Automatically co  │ Shared       │
 [X] Available to all  └──────────────┘

                                                        <Cancel> <OK>
```



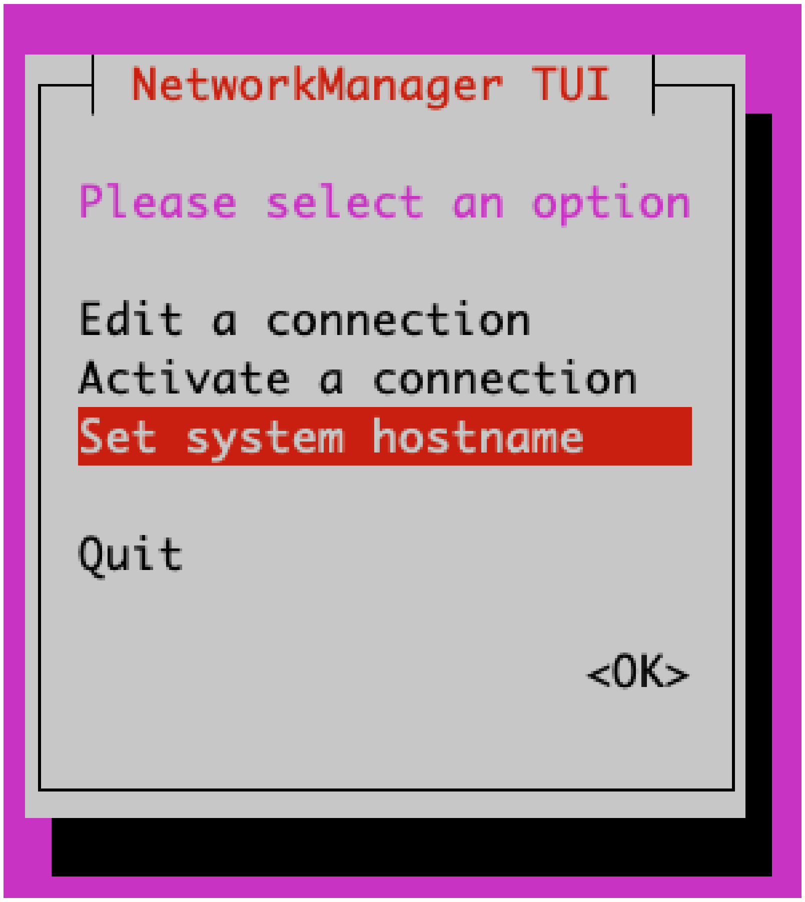
```
                    ┤ Edit Connection ├

       Profile name Wired connection 1________________________
             Device 00:50:56:82:00:FC (ens160)________________

♦ ETHERNET                                                   <Show>

♦ IPv4 CONFIGURATION <Manual>                                <Hide>
│         Addresses 192.168.4.113/24_________ <Remove>
│                   <Add...>
│           Gateway 192.168.4.1______________
│       DNS servers 172.18.108.43____________ <Remove>
│                   <Add...>
│    Search domains cisco.com________________ <Remove>
│                   <Add...>
│
│           Routing (No custom routes) <Edit...>
│ [ ] Never use this network for default route
│ [ ] Ignore automatically obtained routes
│ [ ] Ignore automatically obtained DNS parameters
│
│ [ ] Require IPv4 addressing for this connection
└

♦ IPv6 CONFIGURATION <Automatic>                             <Show>

[X] Automatically connect
[X] Available to all users

                                                     <Cancel> <OK>
```

- To apply the updated configuration, you will need to deactivate and then activate the configured interface.



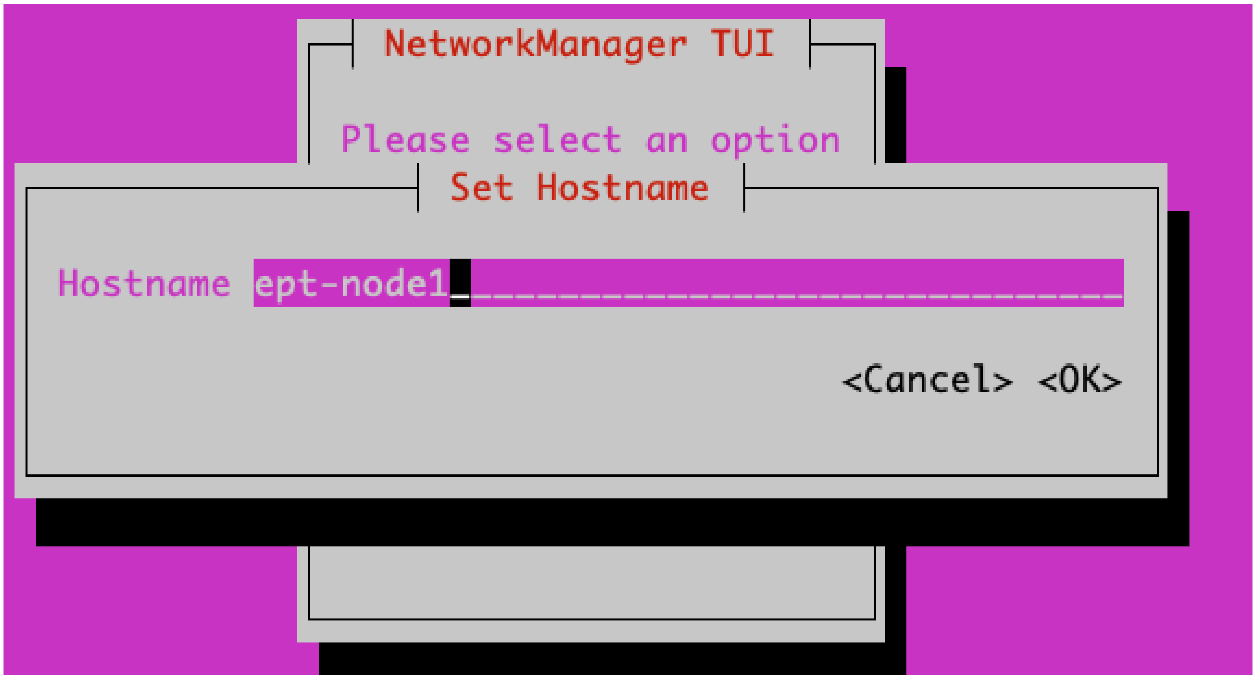


- Ensure you also set the hostname for the host. You will need to logout and log back in to see the hostname updated.



NetworkManager TUI

Please select an option



NetworkManager TUI

Please select an option

Set Hostname

Hostname ept-node1

Configure NTP and Timezone

All timestamps for the app are based on the timestamp of the server itself. If you are running the app on a cluster with more than 1 node or if the time on the host is unreliable, then timestamps for events and analysis may be incorrect. Use **timedatectl** to configure your timezone and the ntp servers.

- List the available options and set to the desired timezone.

```
eptracker@ept-node1:~$ timedatectl list-timezones | grep New
America/New_York
America/North_Dakota/New_SalemA

eptracker@ept-node1:~$ sudo timedatectl set-timezone America/New_York
```

- Use vim or your favorite editor to set the required NTP servers under */etc/systemd/timesyncd.conf*

```
eptracker@ept-node1$ sudo vim /etc/systemd/timesyncd.conf
```

- Uncomment the NTP line and add the appropriate list of NTP servers. For example:

```
eptracker@ept-node1$ cat /etc/systemd/timesyncd.conf | egrep -A 1 Time
[Time]
NTP=172.16.1.1
```

- Restart the ntp process and validate the configuration was successful. **Note**, it may take several a few minutes before ntp synchronizes the clock:

```
eptracker@ept-node1:~$ sudo timedatectl set-ntp true
eptracker@ept-node1:~$ sudo systemctl restart systemd-timesyncd

  eptracker@ept-node1:~$ timedatectl status
                        Local time: Wed 2019-03-13 12:26:50 EDT
                    Universal time: Wed 2019-03-13 16:26:50 UTC
                          RTC time: Wed 2019-03-13 16:26:51
                         Time zone: America/New_York (EDT, -0400)
         System clock synchronized: yes
  systemd-timesyncd.service active: yes  <--------- synchronized
                   RTC in local TZ: no
```

Configure the Cluster and Deploy the Stack

`cluster` mode with the OVA uses docker swarm for the overlay and network orchestration. Even if there is only a single node, the swarm needs to be configured. Before starting, ensure that networking has been configured on all nodes and they are able to communicate on the ports previously listed.

All containers deployed in the stack rely on the `agccie/enhancedendpointtracker:<version>` container. This is available on docker hub and is also available pre-installed on the OVA. **There is no internet requirement to get the app deployed on the OVA.**

- Step 1: Use the `app-deploy` script to initialize the cluster and deploy the app

  The `app-deploy` script performs the following operations

  - Configure the host as a swarm leader
  - Export the manager token to all other nodes and add them to the swarm

  - Add a label called ‘node’ with the appropriate node number to each node in the cluster. The docker compose file uses the node labels to ensure the db shards and replicas are properly distributed.
  - Create the docker compose file based on the desired number of shards, replicas, and workers distributed across the cluster nodes.
  - Deploy the stack.

The default swarm configuration is in the `/opt/cisco/src/cluster/swarm/swarm_config.yml` file. You can edit this file before deploying the stack to adjust worker count, db scale, adjust which port the web service is deployed, and enable/disable http. Additionally, you can pass in arguments for worker count and db configuration which will override the swarm_config.

```
# example deployment with large scale (default worker/shard/memory is␣
↪sufficent for most setups)
eptracker@ept-node1:~$ app-deploy --deploy --worker 23 --db_shard 3 --db_
↪replica 3 --db_memory 2.0
Number of nodes in cluster [1]: 3

UTC 2019-04-27 13:19:39.251||INFO||loading config file: /opt/cisco/src/
↪cluster/swarm/swarm_config.yml
UTC 2019-04-27 13:19:40.018||INFO||compose file complete: /tmp/compose.yml
UTC 2019-04-27 13:19:41.038||INFO||initializing swarm master

Enter hostname/ip address for node 2: 192.168.4.112     <--- you will be␣
↪prompted for node IP
Enter hostname/ip address for node 3: 192.168.4.113

Enter ssh username [eptracker]:                         <--- you will be␣
↪prompted for credentials
Enter ssh password:

UTC 2019-04-27 13:19:59.752||INFO||Adding worker to cluster (id:2,␣
↪hostname:192.168.4.117)
UTC 2019-04-27 13:20:02.670||INFO||Adding worker to cluster (id:3,␣
↪hostname:192.168.4.118)
UTC 2019-04-27 13:20:04.540||INFO||docker cluster initialized with 3␣
↪node(s)
UTC 2019-04-27 13:20:04.541||INFO||deploying app services, please wait...
UTC 2019-04-27 13:30:07.130||INFO||2 services pending, re-check in 60.0␣
↪seconds
UTC 2019-04-27 13:31:07.483||INFO||app services deployed
UTC 2019-04-27 13:31:22.499||INFO||deployment complete
```

Note: The `app-deploy` script requires that all nodes in the cluster have the same username/password configured. Once the deployment is complete, you can set unique credentials on each node.

Tip: The `app-deploy` script is simply an alias to `/opt/cisco/src/cluster/deploy.py` script with some auto-detection for which version to deploy based on the version of the OVA.

- Step 2: Manager the App via the web-GUI

  After deployment is complete, open a web browser to the IP address of any node in the cluster. Using the example above we could access the app on node-3 via to https://192.168.4.113/. The app can be fully managed from the UI. See the usage section for further details regarding how to use the app.

2.2.3 Cluster Mode Manual

Users may prefer to manually configure the cluster in any environment that supports container orchestration. Deploying each container requires the container image which can be pulled from docker hub or manually built using the Dockerfile on github. Once built, the entry point for the container must be `/home/app/src/Service/start.sh` and appropriate arguments and environmental variables are required.

Manually Deploying Cluster Mode with Docker Swarm

This section provides an example for manually deploying cluster mode with docker swarm. This example uses **ubuntu 18.04** with **docker 18.09.2**. It could be extended to support other docker orchestration environments such as Kubernetes or Nomad. Refer to *Container Arguments* and *Environmental Variables* for more info on required settings for manually deploying a container.

Note: These steps assume a linux host or VM. Using docker swarm to deploy a stack on MACOS or Windows has not been tested and may not work as expected.

- Step 1: Install Docker

  Further instructions for install docker on your docs.docker.com.

```
# update apt and install required packages
sudo apt-get update && sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common

# add Docker's official GPC KEY and setup the stable docker repository
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) \
    stable"

# update apt and install docker
sudo apt-get update
sudo apt-get install \
    docker-ce=5:18.09.2~3-0~ubuntu-bionic \
    docker-ce-cli=5:18.09.2~3-0~ubuntu-bionic containerd.io

# add your username to docker group to run docker commands without root (required logout)
sudo usermod -aG docker eptracker
```

- Step 2: Install python and pull automation scripts

  This step is only required if you are using the provided automation scripts to automate deployment of the cluster and service stack. If you are using your own docker orchestration, then this step can be skipped.

```
# install python and python-pip
sudo apt install git python python-pip
```

(continues on next page)

(continued from previous page)

```
# pull the source code in dedicated directory and change ownership to
↪'eptracker'. Ensure
# you substitute the username with your username.
sudo mkdir -p /opt/cisco/src
sudo chown eptracker:eptracker /opt/cisco/src -R
git clone https://github.com/agccie/ACI-EnhancedEndpointTracker.git /opt/
↪cisco/src

# install build python requirements
sudo pip install -r /opt/cisco/src/build/requirements.txt
```

Note: If you installed python and build requirements you can automate all remaining steps. I.e., you can configure the swarm AND create the compose file AND deploy the full stack with a single command. Refer to *Configure the Cluster and Deploy the Stack* for more info

```
python /opt/cisco/src/cluster/deploy.py --deploy
```

- Step 3: Configure the Docker Swarm

  Docker Swarm consist of one or more managers and one or more workers. For redundancy there should be multiple manager processes. The manager process can also be used to run containers or just for monitoring/managing the swarm. In this example, we will deploy on only three nodes which will all be managers. Note you can skip this step if you used the deploy script in Step 2.

```
# intialize node-1 as the swarm master with 10 year certificate
eptracker@ag-docker1:~$ docker swarm init --cert-expiry 87600h0m0s
Swarm initialized: current node (s6pbhtb34ttvv7f1k35df855l) is now a␣
↪manager.
<snip>

# get the manager token to use for other managers in the cluster
eptracker@ag-docker1:~$ docker swarm join-token manager
To add a manager to this swarm, run the following command:
    docker swarm join --token SWMTKN-1-
↪4ef1xrfmosdecb5i4ckm6t4v1rdr95wkbdej4nla0d35mr3i8x-
↪aad8vucl9lfjs65x3whe23upg 192.168.2.78:2377

# assuming docker has been installed on node-2 and node-3, add them to␣
↪the cluster as managers
eptracker@ag-docker2:~$ docker swarm join --token SWMTKN-1-
↪4ef1xrfmosdecb5i4ckm6t4v1rdr95wkbdej4nla0d35mr3i8x-
↪aad8vucl9lfjs65x3whe23upg 192.168.2.78:2377
This node joined a swarm as a manager.

eptracker@ag-docker3:~$ docker swarm join --token SWMTKN-1-
↪4ef1xrfmosdecb5i4ckm6t4v1rdr95wkbdej4nla0d35mr3i8x-
↪aad8vucl9lfjs65x3whe23upg 192.168.2.78:2377
This node joined a swarm as a manager.
```

  Now that the swarm is initialized, verify that all nodes are available and are active.

```
eptracker@ag-docker1:~$ docker node ls
ID                              HOSTNAME            STATUS              ␣
↪AVAILABILITY        MANAGER STATUS      ENGINE VERSION
```

(continues on next page)

(continued from previous page)

```
s6pbhtb34ttvv7f1k35df855l *   ag-docker1          Ready               ␣
↪Active              Leader              18.09.2
5flk9lvtppjoopugcp0ineo8l     ag-docker2          Ready               ␣
↪Active              Reachable           18.09.2
oqcg9okajvgm2l0x74bqsh043     ag-docker3          Ready               ␣
↪Active              Reachable           18.09.2
```

The compose file used in this example will pin various db components to different nodes in the cluster using a docker placement constraint. For this functionality to be successful, we need add appropriate node labels to each node in the cluster. This can be executed on any master node in the swarm.

```
# set the node label for each node in the swarm
eptracker@ag-docker1:~$ docker node update --label-add node=1 ag-docker1
eptracker@ag-docker1:~$ docker node update --label-add node=2 ag-docker2
eptracker@ag-docker1:~$ docker node update --label-add node=3 ag-docker3

# validate the node label is present
eptracker@ag-docker1:~$ docker node inspect ag-docker1 --format '{{ .Spec.
↪Labels }}'
map[node:1]
eptracker@ag-docker1:~$ docker node inspect ag-docker2 --format '{{ .Spec.
↪Labels }}'
map[node:2]
eptracker@ag-docker1:~$ docker node inspect ag-docker3 --format '{{ .Spec.
↪Labels }}'
map[node:3]
```

- Step 3: Create the compose file to start the stack

  In this example will use the swarm_config.yml referenced in *Configure the Cluster and Deploy the Stack* combined with the automation scripts to create the compose file. Again, refer to *Container Arguments* and *Environmental Variables* for required settings if you are manually creating a swarm configuration file.

```
# use --help for more options. Use --version for specific version else␣
↪latest image is used.
eptracker@ag-docker1:~$ python /opt/cisco/src/cluster/deploy.py --config␣
↪[--version 2.0.12]
eptracker@ag-docker1:~$ python /opt/cisco/src/cluster/deploy.py --config
Number of nodes in cluster [1]: 3
EST 2019-02-28 18:08:07.029||INFO||loading config file: /opt/cisco/src/
↪cluster/swarm/swarm_config.yml
EST 2019-02-28 18:08:07.135||INFO||compose file complete: /tmp/compose.yml

# verify compose file is present
eptracker@ag-docker1:~$ more /tmp/compose.yml
        networks:
          default:
            ipam:
              config:
              - subnet: 192.0.2.0/24
        services:
          db:
            command: '/home/app/src/Service/start.sh -r db -l '
            deploy:
              mode: global
```

(continues on next page)

(continued from previous page)

```
            environment:
            - DB_CFG_SRV=cfg/db_cfg_0:27019,db_cfg_1:27019,db_cfg_2:27019
            - DB_RS_SHARD_0=sh0/db_sh_0_0:27017,db_sh_0_1:27017,db_sh_0_
↪2:27017
            - DB_RS_SHARD_1=sh1/db_sh_1_0:27017,db_sh_1_1:27017,db_sh_1_
↪2:27017
            - DB_RS_SHARD_2=sh2/db_sh_2_0:27017,db_sh_2_1:27017,db_sh_2_
↪2:27017
            - DB_SHARD_COUNT=3
            - HOSTED_PLATFORM=SWARM
            - MONGO_HOST=db
            - MONGO_PORT=27017
            - REDIS_HOST=redis
            - REDIS_PORT=6379
            - LOCAL_REPLICA=0
            - LOCAL_SHARD=0
            - LOCAL_PORT=27017
            - DB_MEMORY=2.0
            - DB_ROLE=mongos
            image: agccie/enhancedendpointtracker:latest
            logging:
              driver: json-file
              options:
                max-buffer-size: 1m
                max-file: '10'
                max-size: 50m
                mode: non-blocking
            volumes:
            - db-log:/home/app/log
<snip>
```

- Step 4: Deploy the stack and verify

  The final step is to deploy the stack and verify all services are operational. This can be done on any master node. The syntax for the command is *docker stack deploy -c <compose file> <stack name>*

```
# deploy the stack
eptracker@ag-docker1:~$ docker stack deploy -c /tmp/compose.yml ept
Creating network ept_default
Creating service ept_db_sh_1_0
Creating service ept_db
Creating service ept_web
Creating service ept_redis
<snip>
```

  Next verify that all required services are running. From the output below we can see the number of configured replicas for each service, the number successfully running, and the external exposed ports. We expect 1/1 for most replicas and the ept_web service exposed on port 80 and port 443.

```
# stack is running with 27 services
eptracker@ag-docker3:~$ docker stack ls
NAME                SERVICES            ORCHESTRATOR
ept                 27                  Swarm

# verify all services are running
eptracker@ag-docker1:~$ docker service ls
ID                  NAME                MODE                REPLICAS     ␣
↪       IMAGE                                   PORTS
```

(continues on next page)

(continued from previous page)

```
1y1vauo1yahi        ept_db              global              3/3                ␣
↪       agccie/enhancedendpointtracker:latest
2r53aefqghyf        ept_db_cfg_1        replicated          1/1                ␣
↪       agccie/enhancedendpointtracker:latest
m7ryoimptzbt        ept_db_cfg_2        replicated          1/1                ␣
↪       agccie/enhancedendpointtracker:latest
vkqz5h2np5bt        ept_db_sh_0_0       replicated          1/1                ␣
↪       agccie/enhancedendpointtracker:latest
ofd174ixmeem        ept_web             replicated          1/1                ␣
↪       agccie/enhancedendpointtracker:latest   *:80->80/tcp, *:443->443/
↪tcp
<snip>

# further inspection to determine which node a specific service is running
eptracker@ag-docker1:~$ docker service ps ept_mgr
ID                  NAME                IMAGE                              ␣
↪       NODE                DESIRED STATE       CURRENT STATE            ␣
↪ERROR               PORTS
yvq6uunapsh1        ept_mgr.1           agccie/
↪enhancedendpointtracker:latest   ag-docker2          Running            ␣
↪ Running 5 minutes ago
```

The application stack has successfully been deployed.

Container Arguments

This section lists the available arguments to the `/home/app/src/Service/start.sh` startup script which executed by default when starting the container.

-r `role`

The role for the container to execute. There are several different roles required for the app to execute correctly. See *Components* for more details. The allowed rows as follows:

`all-in-one` (default role) all-in-one starts all required processes within the same container. This can be combined with `count` option to adjust the number of workers. This mode runs a single instance of mongo with no sharding support.

`web` web role will run the apache web process on port 80 and 443 with a self-signed certificate. Additional docker arguments can be included to expose these ports on whatever external ports are required.

`redis` will run a single instance of redis on **REDIS_PORT** which defaults to 6379

`db` runs a single instance of mongo v3.6.10. There are several **required** environmental variables. If not provided the container will restart.

- **DB_ROLE**
- **DB_SHARD_COUNT**
- **DB_CFG_SRV**
- **DB_MEMORY**
- **LOCAL_PORT**
- **LOCAL_REPLICA**
- **LOCAL_SHARD**

`mgr` runs an instance of manager process. There should only be a single instance of manager running for the entire application. The manager is also responsible for initializing the db cluster and therefore requires the following environment variables previously defined within `db` role:

- **DB_CFG_SRV**
- **DB_SHARD_COUNT**

`watcher` runs single instance of the watcher process with provided `identity`. watcher will also start exim4 process used for sending email notifications, if configured.

`worker` runs one or more instances of worker process. The worker process uses `count` option to set the number of worker instances running within the container. The `identity` assigned to each worker is relative to the initial `identity` provided. For example, if an id of 5 is assigned to the worker and a count of 3 is provided, then there will be three workers started in the container with id's 5, 6, and 7.

It is recommended to use `-c 1` when executing the worker role.

-i `identity` unique integer `identity` required for `mgr`, `watcher`, and `worker` components.

Note: Ensure that there are no overlapping `identities` per role. A duplicate id will result in race conditions that can trigger invalid analysis.

-c `count` count is an integer for the number of workers to run within a single container. This is applicable to `all-in-one` and `worker` roles only.

-l `log-rotation` enables log rotation within the container. If an external component is managing log rotation or you are using stdout for all logging then this is not required.

Warning: the application can perform extensive logging. If there is no component performing the log rotation then **-l** should be provided.

Note: all logs are saved to `/home/app/log` or a sub folder within this directory.

-s `stdout` enables all logging to stdout. Note that stdout is not currently supported with `web` role.

Environmental Variables

There are several required environmental variables depending on which `role` the container is executing.

HOSTED_PLATFORM Should be statically set to *SWARM*. This is required for proper creation of various config instance files. This should be set on all containers.

MONGO_HOST the hostname of the `db` role with **DB_ROLE** = *mongos*. This should be set on all containers.

MONGO_PORT the **LOCAL_PORT** number of the `db` role with **DB_ROLE** = *mongos*. This should be set on all containers.

REDIS_HOST the hostname of the redis role container. This should be set on all containers.

REDIS_PORT the port number where redis is exposed. This should be set on all containers.

DB_ROLE The role can be `mongos`, `configsvr`, or `shardsvr`. The application requires at least one instance of each. If running as configsvr, the replica set name is statically configured as **cfg**. If running as a sharsvr, the replcia set is statically configured as 'sh$LOCAL_SHARD' where shard number starts at 0.

DB_SHARD_COUNT the number of db shards. This is used by mgr process during db init.

DB_CFG_SRV used by mongos instance to connect to configsvr replica set. This will be in the format `cfg/<configsvr0-hostname:configsvr0-port, ...>`. For example, if there is a replica set of 3 config servers each exposed on port 27019 with hostname db_cfg_0, db_cfg_1, db_cfg_2, then DB_CFG_SVR should be set to `cfg/db_cfg_0:27019,db_cfg_1:27019,db_cfg_2:27019`

DB_MEMORY Amount of memory this instance of mongo is allowed to use. This is measured in GB and can be a float. For example, 1.5 would limit mongo instance to 1.5 GB of memory.

LOCAL_PORT local tcp port to expose running instance of mongo

LOCAL_REPLICA replica number for this mongo instance. **LOCAL_REPLICA** should be set to 0 for mongos role. The configsvr and shardsvr are each deployed in replica sets so each instance will have a **LOCAL_REPLICA** starting at 0.

LOCAL_SHARD shard number for shardsvr instance. For mongos and configsvr this should be set to 0.

CHAPTER 3

Usage

This section assumes EnhancedEndpointTracker has already been installed. Please refer to the *Install* for more details regarding installation steps. This page will cover general usage of the application. Note that some sections are specific to restricted to *standalone* mode only.

3.1 User Accounts

Note: This section is only applicable to *standalone* mode. In *app* mode the APIC handles all authentication and authorization requests.

Once the application is installed, the default credentials are as follows:

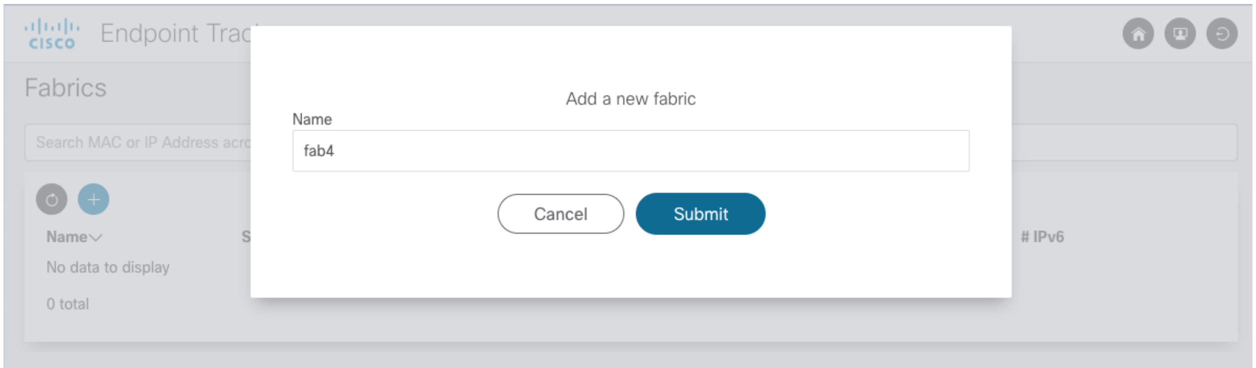
- **username**: admin
- **password**: cisco

You can click the Users 🖥 icon at the top right of the page to modify credentials along with adding and modifying users. The app supports an **admin** role which has full read and write access along with a read-only **user** role. Refer to api for more details regarding required role for each operation.

3.2 Fabrics

Note: This section is only applicable to *standalone* mode. In *app* mode the fabric is auto discovered when the app is installed. Only one fabric will be monitored.

Click the Home ⌂ icon to access the home page and add fabrics to monitor. Multiple fabrics can be monitored by a single app and you can search for an endpoint from the Home page across all fabrics. Simply click the Add icon and enter the fabric name. Once submitted you will be redirected to the *Settings* page to further configure various monitoring options.



Add a new fabric

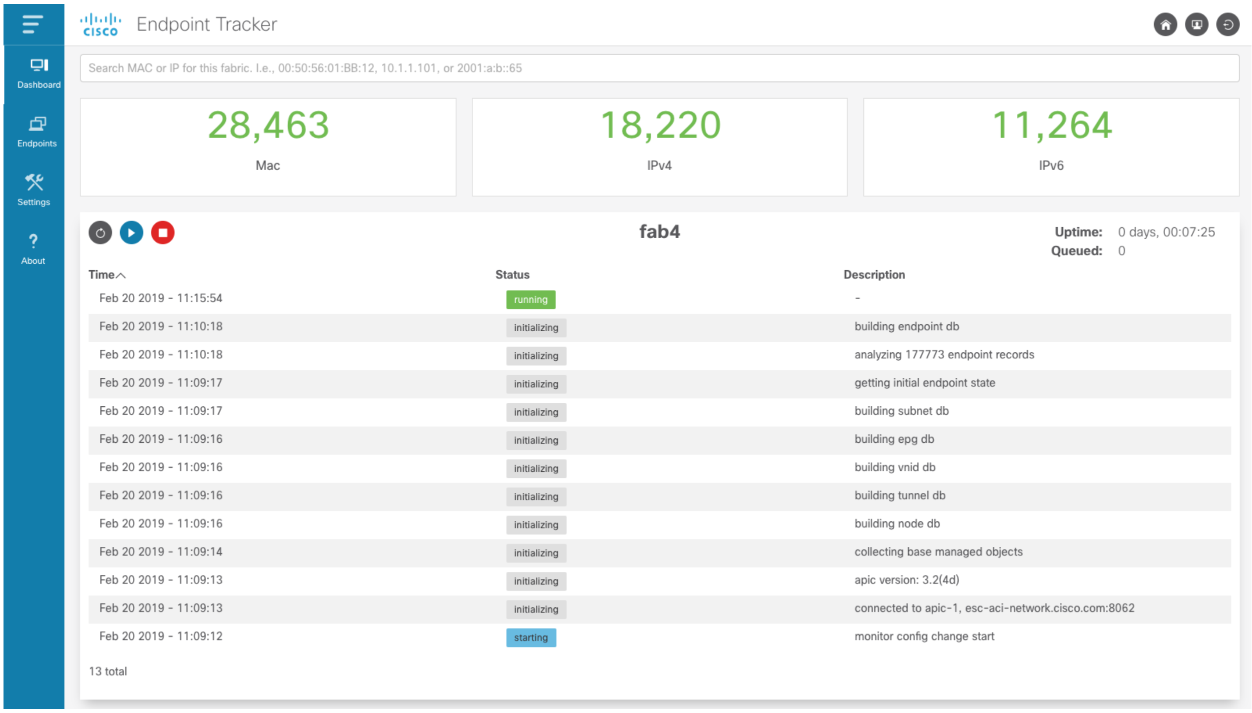
Name

fab4

3.3 Dashboard

The dashboard page provides an overview of the current status of the fabric monitor. You can see an overall count for the total number of active endpoints, a history of the fabric monitor events, along with the uptime of the monitor. Operators can control the monitor via the Start ▶ and Stop ■ icons.

Tip: Loopback addresses and pervasive SVIs are tracked but are not considered as active endpoints within the fabric and do not count toward the total active count. Additionally, if an endpoint has been deleted the records remain within the database and can be searched/viewed but will not count toward the total active count.


Endpoint Tracker
Dashboard
Endpoints
Settings
About
Search MAC or IP for this fabric. I.e., 00:50:56:01:BB:12, 10.1.1.101, or 2001:a:b::65

28,463	18,220	11,264
Mac	IPv4	IPv6

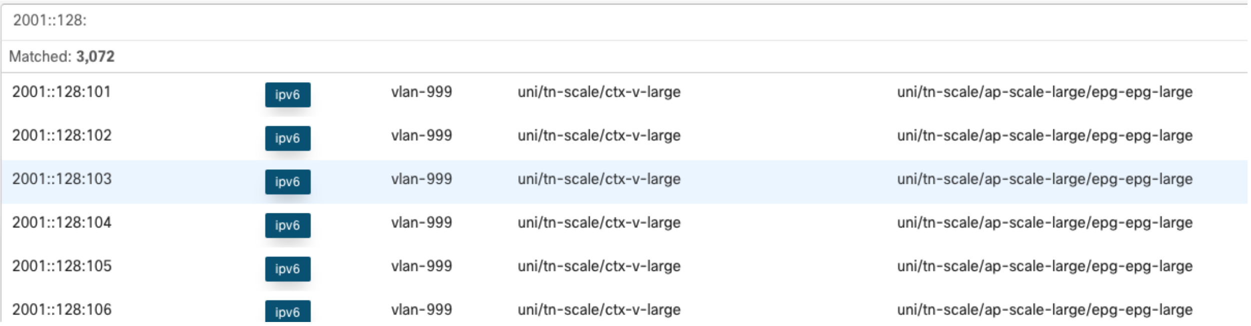
fab4

Uptime: 0 days, 00:07:25
Queued: 0

Time	Status	Description
Feb 20 2019 - 11:15:54	running	-
Feb 20 2019 - 11:10:18	initializing	building endpoint db
Feb 20 2019 - 11:10:18	initializing	analyzing 177773 endpoint records
Feb 20 2019 - 11:09:17	initializing	getting initial endpoint state
Feb 20 2019 - 11:09:17	initializing	building subnet db
Feb 20 2019 - 11:09:16	initializing	building epg db
Feb 20 2019 - 11:09:16	initializing	building vnid db
Feb 20 2019 - 11:09:16	initializing	building tunnel db
Feb 20 2019 - 11:09:16	initializing	building node db
Feb 20 2019 - 11:09:14	initializing	collecting base managed objects
Feb 20 2019 - 11:09:13	initializing	apic version: 3.2(4d)
Feb 20 2019 - 11:09:13	initializing	connected to apic-1, esc-aci-network.cisco.com:8062
Feb 20 2019 - 11:09:12	starting	monitor config change start

13 total

Operators can quickly find an endpoint of interest by typing in the corresponding MAC, IPv4, or IPv6 address in the search bar at the top of the page. The search returns the number of endpoints matched along with the full address, endpoint type, encapsulation, VRF name, and EPG name where the endpoint is learned.



2001::128:

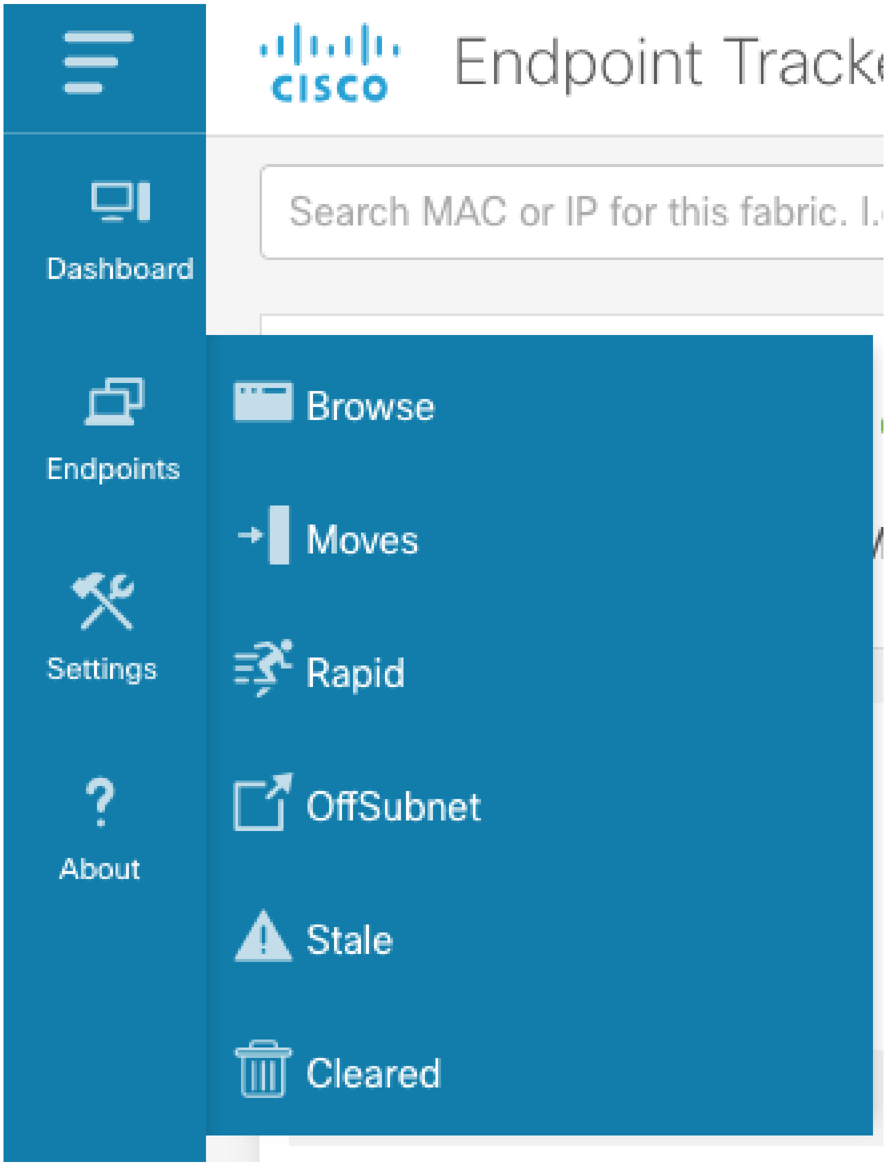
Matched: **3,072**

2001::128:101	ipv6	vlan-999	uni/tn-scale/ctx-v-large	uni/tn-scale/ap-scale-large/epg-epg-large
2001::128:102	ipv6	vlan-999	uni/tn-scale/ctx-v-large	uni/tn-scale/ap-scale-large/epg-epg-large
2001::128:103	ipv6	vlan-999	uni/tn-scale/ctx-v-large	uni/tn-scale/ap-scale-large/epg-epg-large
2001::128:104	ipv6	vlan-999	uni/tn-scale/ctx-v-large	uni/tn-scale/ap-scale-large/epg-epg-large
2001::128:105	ipv6	vlan-999	uni/tn-scale/ctx-v-large	uni/tn-scale/ap-scale-large/epg-epg-large
2001::128:106	ipv6	vlan-999	uni/tn-scale/ctx-v-large	uni/tn-scale/ap-scale-large/epg-epg-large

If the fabric is not running or `eptManager` is not active, then an appropriate message is displayed at the top of the Dashboard page.

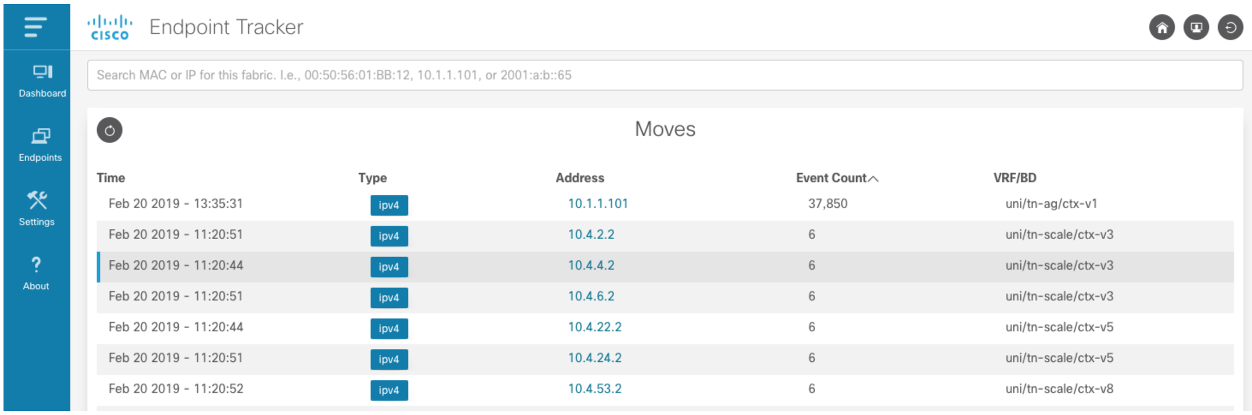
3.4 Endpoints

There are several analyses performed and as a result several endpoint tables that can be viewed. Each page supports various filtering, a total count for number of endpoints matching filter, flexible sorting on attributes of interest, and resizable columns.



Browse allows operators to view the **Current Endpoint State** of for the fabric. Use this page to walk through all active endpoints, filter on currently offsubnet, stale, or rapid endpoints.

Moves shows all endpoints that have moved within the fabric. The table is sorted by most recent event but can also be sorted based on available column. It is extremely useful to sorted based on move event count which will show any endpoints that are unstable in the fabric. In the example below we can see that `10.1.1.101` has moved `37k` times which indicates we may have a misconfiguration that needs to be addressed.



Search MAC or IP for this fabric. I.e., 00:50:56:01:BB:12, 10.1.1.101, or 2001:a:b::65

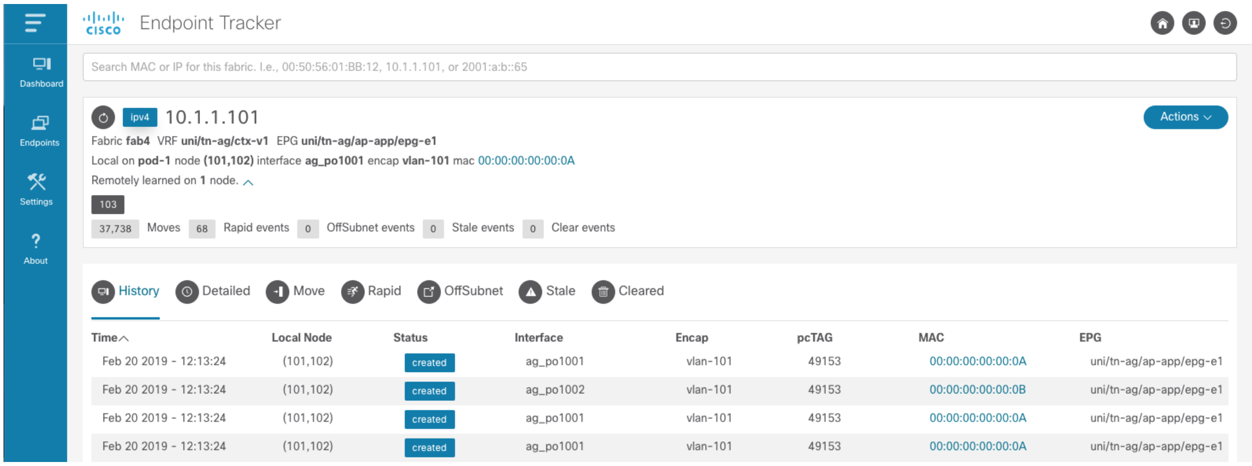
Moves

Time	Type	Address	Event Count	VRF/BD
Feb 20 2019 - 13:35:31	ipv4	10.1.1.101	37,850	uni/tn-ag/ctx-v1
Feb 20 2019 - 11:20:51	ipv4	10.4.2.2	6	uni/tn-scale/ctx-v3
Feb 20 2019 - 11:20:44	ipv4	10.4.4.2	6	uni/tn-scale/ctx-v3
Feb 20 2019 - 11:20:51	ipv4	10.4.6.2	6	uni/tn-scale/ctx-v3
Feb 20 2019 - 11:20:44	ipv4	10.4.22.2	6	uni/tn-scale/ctx-v5
Feb 20 2019 - 11:20:51	ipv4	10.4.24.2	6	uni/tn-scale/ctx-v5
Feb 20 2019 - 11:20:52	ipv4	10.4.53.2	6	uni/tn-scale/ctx-v8

There are additional pages for **Rapid**, **Offsubnet**, **Stale**, and **Cleared** records. Each of these pages are historical records for past detection events. Similar to **Moves**, operators can sort and page through results as needed.

3.4.1 Endpoint Details

The power of the EnhancedEndpointTracker app is the **Endpoint Detail** page. This allows operators to see the current state of endpoint within the fabric along with the historical records of what has happened to the endpoint in the past. On the Overview section, the current state of the endpoint is listed. As seen in the example below, this includes the current location of the endpoint in the fabric including VRF, EPG, pod, node, interface, encap, and rewrite information. All nodes where the endpoint is remotely learned (XR) is also available. Also, a summary count of each event type that has occurred for this endpoint is displayed. If an endpoint is currently rapid, offsubnet, or stale it will be highlighted along with the list of affected nodes.



Search MAC or IP for this fabric. I.e., 00:50:56:01:BB:12, 10.1.1.101, or 2001:a:b::65

ipv4 10.1.1.101

Fabric **fab4** VRF **uni/tn-ag/ctx-v1** EPG **uni/tn-ag/ap-app/epg-e1**

Local on **pod-1** node **(101,102)** interface **ag_po1001** encap **vlan-101** mac 00:00:00:00:00:0A

Remotely learned on **1** node.

103

37,738 Moves 68 Rapid events 0 OffSubnet events 0 Stale events 0 Clear events

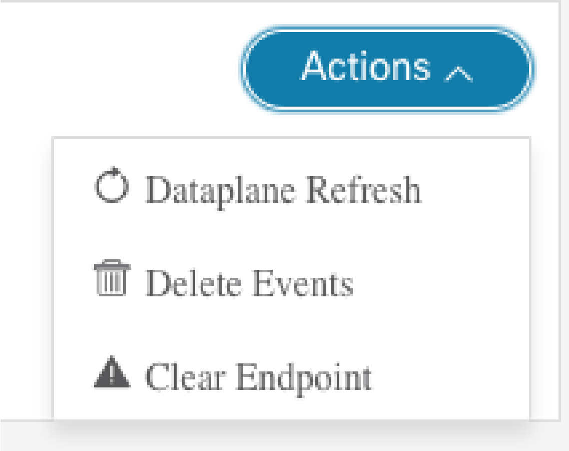
Time	Local Node	Status	Interface	Encap	pcTAG	MAC	EPG
Feb 20 2019 - 12:13:24	(101,102)	created	ag_po1001	vlan-101	49153	00:00:00:00:00:0A	uni/tn-ag/ap-app/epg-e1
Feb 20 2019 - 12:13:24	(101,102)	created	ag_po1002	vlan-101	49153	00:00:00:00:00:0B	uni/tn-ag/ap-app/epg-e1
Feb 20 2019 - 12:13:24	(101,102)	created	ag_po1001	vlan-101	49153	00:00:00:00:00:0A	uni/tn-ag/ap-app/epg-e1
Feb 20 2019 - 12:13:24	(101,102)	created	ag_po1001	vlan-101	49153	00:00:00:00:00:0A	uni/tn-ag/ap-app/epg-e1

Tip: All columns are resizable. Simply click to the left of the column name and drag the column to make it wider.

- **History** displays the local learn events for the endpoint in the fabric along with delete events
- **Detailed** is a per-node history of events that has occurred for this endpoint. It provides an additional search bar to filter on a specific attribute such as node, epg, pcTag, etc... This is extremely helpful for experienced operators who need to know the state and history of the endpoint on a specific node.
- **Move** displays the move events for this endpoint. Each row has the source and destination for the move.
- **Rapid** displays the rapid events detected for this endpoint. The timestamp when the endpoint was flagged as rapid along with the total number of epm events at that instance and the calculated rate of events are also displayed

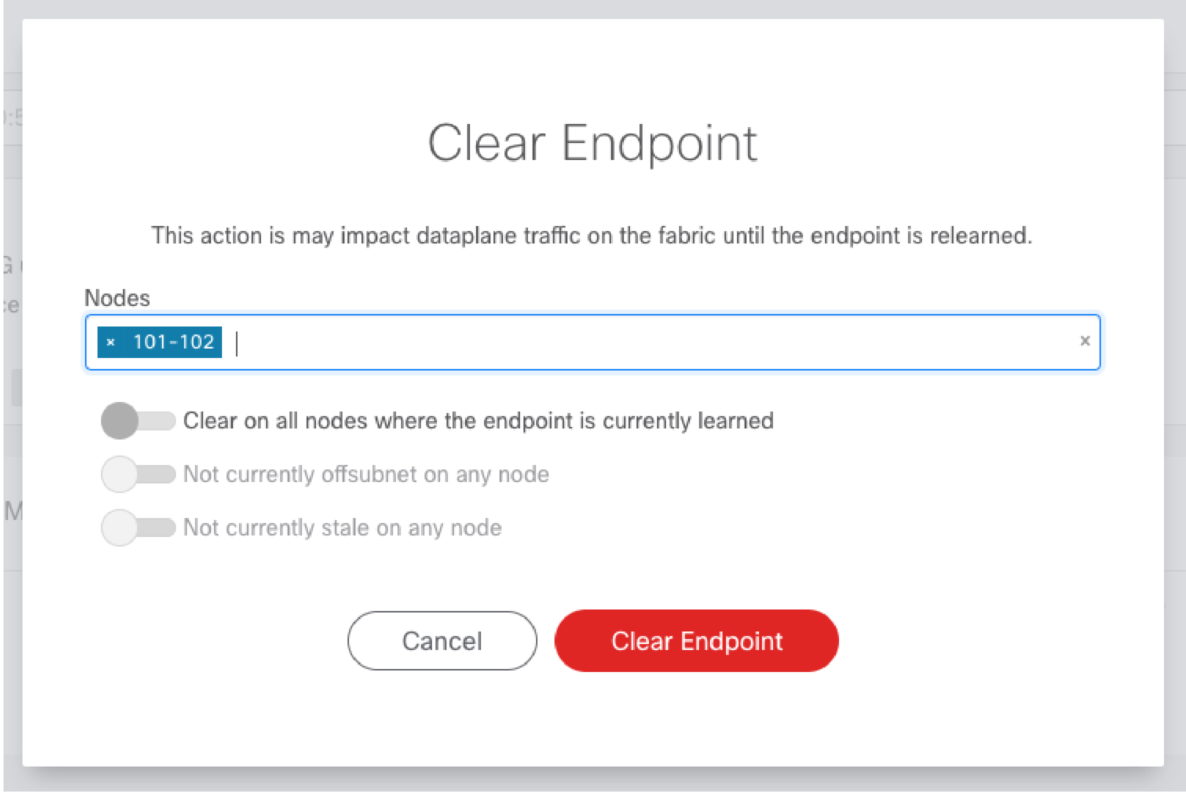
- **OffSubnet** displays offsubnet events detected for this endpoint. The affected node, interface, encap, and EPG are also displayed. It's common that a misconfigured endpoint fails subnet check on the ingress leaf but still triggers a remote learn on another node. For this reason, the remote node column is available so operators know which leaf the offsubnet endpoint originated from.
- **Stale** displays stale events detected for this endpoint. Similar to the other tables, the affected node, interface, encap, EPG, and remote node are captured. Generally, a stale endpoint is a remote learn pointing to an incorrect leaf. This table includes the expected remote node at the time the stale event was detected.
- **Cleared** displays the timestamp and reason an endpoint was cleared from the fabric by this app.

There are a few actions that can be performed on an endpoint.



- **Dataplane Refresh** will query the APIC for the most recent state of the endpoint in the fabric and update the app database. This is used as a sanity check to ensure that the state of the endpoint reported in the app is 100% correct. There is no impact to this operation but it does require that the fabric monitor is actively running.
- **Delete Events** will delete the endpoint information from the app database. It has no impact on the fabric. This is a useful mechanism to delete historical information on endpoints you no longer care about. After the entry is removed from the app, a refresh is also triggered to ensure the app stays in sync with the fabric. Therefore, you may notice that after the delete the endpoint is immediately relearned by the app.
- **Clear Endpoint** allows the operator to clear an endpoint from the fabric on one or more nodes. This operation requires SSH credentials are configured under the fabric *Settings*. When clearing an endpoint, operators can provided a list of nodes. Operators can also use the available toggles:
  - **Clear on All Active Nodes** will clear the endpoint on all nodes that currently have the endpoint learned. This includes both local learns and remote learns
  - **Clear on All Offsubnet Nodes** will clear the endpoint on all nodes that have currently learned the endpoint offsubnet
  - **Clear on All Stale Nodes** will clear the endpoint on all nodes that are currently stale.

  > **Warning:** Clearing the endpoint will trigger an EPM delete in the fabric. This can cause a traffic impact to the endpoint until it is relearned in the fabric.



Clear Endpoint

This action is may impact dataplane traffic on the fabric until the endpoint is relearned.

Nodes

× 101-102

Clear on all nodes where the endpoint is currently learned
Not currently offsubnet on any node
Not currently stale on any node
Cancel
Clear Endpoint

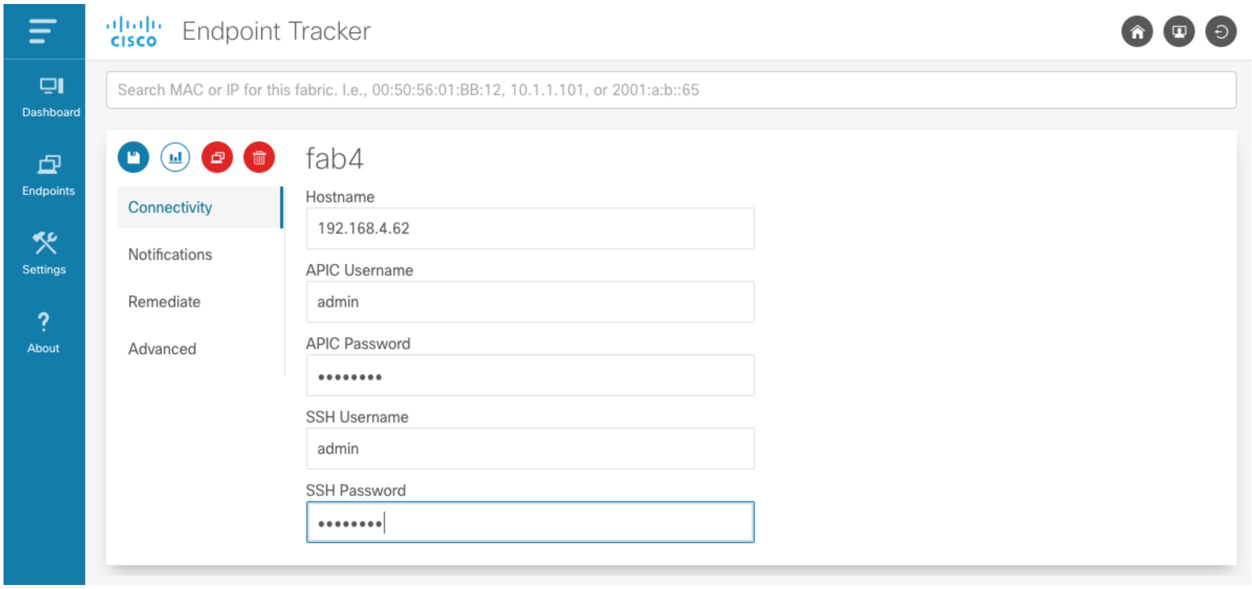
3.5 Settings

The settings section is divided into multiple sub-sections. Ensure you click the Save 💾 icon to apply your changes. If you update the connectivity settings you will need to restart the fabric monitor for them to be applied.

3.5.1 Connectivity

Note: APIC hostname and API credentials are auto detected in *app* mode and cannot be changed.

Connectivity settings containing hostname and APIC credentials are required to access the fabric. An APIC username with **admin role** on the **all** security domain is required. SSH credentials are only required if clear endpoint functionality is required. For ssh, a user with **admin** read role is required. When settings are saved the credentials are checked and an error is displayed if the credentials are invalid



fab4

Hostname: 192.168.4.62

APIC Username: admin

APIC Password: ••••••••

SSH Username: admin

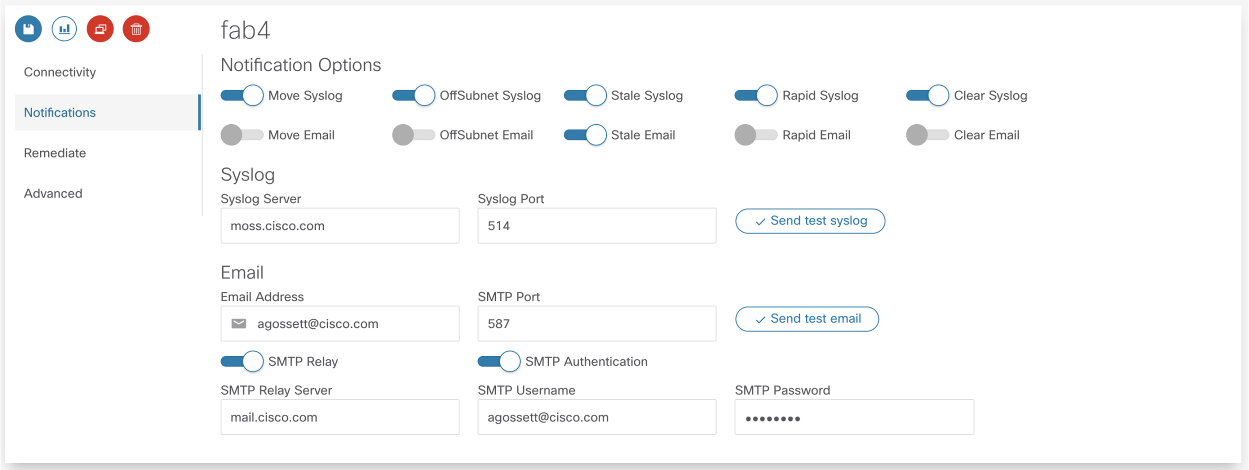
SSH Password: ••••••••

3.5.2 Notifications

There are very flexible notification options. Users can choose to be notified via syslog and email for each of the analysis and detection mechanisms available. **Once you've saved the settings** you can test both syslog and email servers by clicking the *Send test syslog* and *send test email* buttons, respectively.

By default email notifications are sent directly to the mail server corresponding to configured email address. By default SMTP messages are sent on port **587** with TLS encryption but can be configured for standard port **25** or any required custom port. Many public mail exchanges will not accept email directly from unknown hosts. You can configure email notifications to be sent through an SMTP relay along with required SMTP authentication.

In the example below, syslog notifications are generated for all events and an email is sent if a stale endpoint is detected.



fab4

Notification Options

Move Syslog: on; OffSubnet Syslog: on; Stale Syslog: on; Rapid Syslog: on; Clear Syslog: on

Move Email: off; OffSubnet Email: off; Stale Email: on; Rapid Email: off; Clear Email: off

Syslog

Syslog Server: moss.cisco.com

Syslog Port: 514

Email

Email Address: agossett@cisco.com

SMTP Port: 587

SMTP Relay: on

SMTP Authentication: on

SMTP Relay Server: mail.cisco.com

SMTP Username: agossett@cisco.com

SMTP Password: ••••••••

Syslog/Email Requirements

Syslogs and Email notifications are sent from the `eptWatcher` process. There are one or more DNS lookups performed before the message is sent. The following ports need to be allowed:

Syslog

- DNS lookup (**UDP** port **53**) for A-record of syslog server
- Syslog frame (**UDP** port **514** or custom configured port)

Email

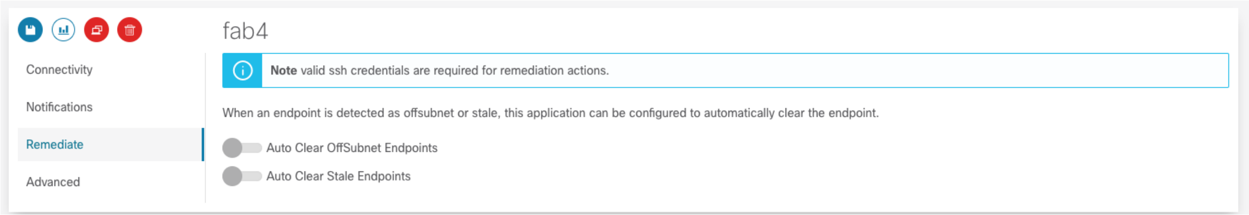
- DNS lookup (**UDP** port **53**) for MX-record of email servers to reach configured email domain
- DNS lookup for corresponding A-record of each returned email server
- SMTP (**TCP** port **587** or custom configured port) connection to send the email to the selected email server

Note: When executing in app mode, the container is executed on the APIC and the source IP of syslog/email notifications will be translated to the APIC inband or out-of-band address. When running in standalone mode, the IP will be from the docker host.

3.5.3 Remediate

The application can be configured to automatically remediate offsubnet or stale endpoints. By default, auto-remediation is disabled.

Remediation is performed by opening an ssh connection to the affected leaf and issuing an epm clear command to delete the endpoint. Ssh credentials are required to perform this action. When an endpoint is cleared there is an entry logged to the Remediation database and a notification may be sent. Fabric operators have full visibility into what clear events have occurred.



Note valid ssh credentials are required for remediation actions.

When an endpoint is detected as offsubnet or stale, this application can be configured to automatically clear the endpoint.

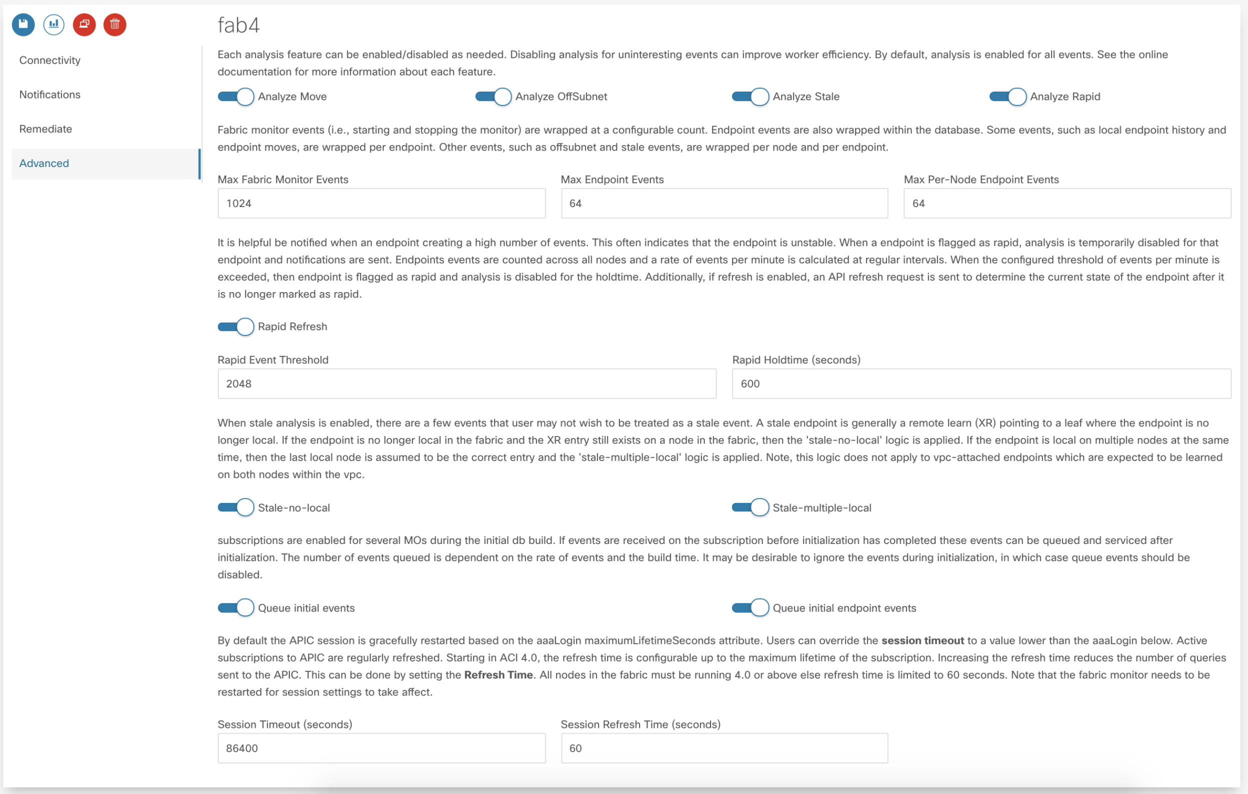
○ Auto Clear OffSubnet Endpoints

○ Auto Clear Stale Endpoints

3.5.4 Advanced

There are several nerd knobs available to affect how the app is running. The default settings are suitable for most deployments. However, there may be scenarios where this settings need to be updated. The following settings are available within the **Advanced** section.

If you are using the API then these settings would apply to **fabric** and **eptSettings** objects.



fab4

Each analysis feature can be enabled/disabled as needed. Disabling analysis for uninteresting events can improve worker efficiency. By default, analysis is enabled for all events. See the online documentation for more information about each feature.

Analyze Move: on
Analyze OffSubnet: on
Analyze Stale: on
Analyze Rapid: on

Fabric monitor events (i.e., starting and stopping the monitor) are wrapped at a configurable count. Endpoint events are also wrapped within the database. Some events, such as local endpoint history and endpoint moves, are wrapped per endpoint. Other events, such as offsubnet and stale events, are wrapped per node and per endpoint.

Max Fabric Monitor Events: 1024
Max Endpoint Events: 64
Max Per-Node Endpoint Events: 64

It is helpful be notified when an endpoint creating a high number of events. This often indicates that the endpoint is unstable. When a endpoint is flagged as rapid, analysis is temporarily disabled for that endpoint and notifications are sent. Endpoints events are counted across all nodes and a rate of events per minute is calculated at regular intervals. When the configured threshold of events per minute is exceeded, then endpoint is flagged as rapid and analysis is disabled for the holdtime. Additionally, if refresh is enabled, an API refresh request is sent to determine the current state of the endpoint after it is no longer marked as rapid.

Rapid Refresh: on

Rapid Event Threshold: 2048
Rapid Holdtime (seconds): 600

When stale analysis is enabled, there are a few events that user may not wish to be treated as a stale event. A stale endpoint is generally a remote learn (XR) pointing to a leaf where the endpoint is no longer local. If the endpoint is no longer local in the fabric and the XR entry still exists on a node in the fabric, then the 'stale-no-local' logic is applied. If the endpoint is local on multiple nodes at the same time, then the last local node is assumed to be the correct entry and the 'stale-multiple-local' logic is applied. Note, this logic does not apply to vpc-attached endpoints which are expected to be learned on both nodes within the vpc.

Stale-no-local: on
Stale-multiple-local: on

subscriptions are enabled for several MOs during the initial db build. If events are received on the subscription before initialization has completed these events can be queued and serviced after initialization. The number of events queued is dependent on the rate of events and the build time. It may be desirable to ignore the events during initialization, in which case queue events should be disabled.

Queue initial events: on
Queue initial endpoint events: on

By default the APIC session is gracefully restarted based on the aaaLogin maximumLifetimeSeconds attribute. Users can override the **session timeout** to a value lower than the aaaLogin below. Active subscriptions to APIC are regularly refreshed. Starting in ACI 4.0, the refresh time is configurable up to the maximum lifetime of the subscription. Increasing the refresh time reduces the number of queries sent to the APIC. This can be done by setting the **Refresh Time**. All nodes in the fabric must be running 4.0 or above else refresh time is limited to 60 seconds. Note that the fabric monitor needs to be restarted for session settings to take affect.

Session Timeout (seconds): 86400
Session Refresh Time (seconds): 60

Enable/Disable Specific Analysis

Each analysis feature can be enabled/disabled as needed. Disabling analysis for uninteresting events can improve worker efficiency. The following analysis is available:

- **Move** tracks move events within the fabric. When a move is detected, an event is added to the **eptMove** table and an optional notification is sent.
- **Offsubnet** analysis keeps track of all configured fvSubnet/fvIpAttr objects in the fabric and corresponding mapping to fvAEPg/fvBD. When an IP learn occurs, the vrf VNID and pcTag from the endpoint are used to derive the originating EPG and corresponding BD. The IP is checked against all subnets configured for that BD and if it doesn't match it the endpoint is flagged as offsubnet. This mechanism can be used for both local (PL/VL) learns along with remove (XR) learns. When an offsubnet endpoint is detected, an event is added to the **eptOffsubnet** table and the **is_offsubnet** flag is set for the endpoint in the **eptEndpoint** table.
- **Stale** analysis tracks where in the fabric an endpoint is locally learned. When a new learn is seen on a node, it is cross referenced to where it is expected to be learned. If the learn does not point to the expected leaf (or leaf-pair in the case of vPC) then the endpoint is flagged as stale. Note, stale analysis also includes the appropriate logic to handle bounce/bounce-to-proxy scenarios. When a stale endpoint is detected, an event is added to the **eptStale** table and the **is_stale** flag is set for the endpoint in the **eptEndpoint** table.
- **Rapid** analysis is a mechanism that counts the number of events received across all nodes for a single endpoint. If the number of events per minute exceed the configured threshold, then the endpoint is flagged as rapid and further events from this endpoint are ignored until the rapid hold-down timer have expired. Rapid analysis helps operators quickly determine if an endpoint is unstable in the fabric. Additionally, it protects the app from processing excessive events from an unstable endpoint. When a rapid endpoint is detected, an event is added to the **eptRapid** table and the **is_rapid** flag is set for the endpoint in the **eptEndpoint** table.

Event Count

Fabric monitor events (i.e., starting and stopping the monitor) are wrapped at a configurable count. Endpoint events are also wrapped within the database. Some events, such as local endpoint history and endpoint moves, are wrapped per endpoint. Other events, such as offsubnet and stale events, are wrapped per node and per endpoint. Users can set the following thresholds:

- **Max Fabric Monitor Events** the maximum number of fabric monitor events to keep. If the threshold is exceeded then older events are discarded.
- **Max Endpoint Events** the maximum number of endpoint events to keep. This applies to several endpoint tables such as **eptEndpoint**, **eptMove**, and **eptRapid**.
- **Max Per-Node Endpoint Events** the maximum number of per-node endpoint events to keep. This applies to all endpoint tables that are keyed per node. This includes **eptHistory**, **eptOffsubnet**, **eptStale**, and **eptRemediate**.

Rapid Endpoint Paramaters

It is helpful be notified when an endpoint is creating a high number of events. This often indicates that the endpoint is unstable. When a endpoint is flagged as rapid, analysis is temporarily disabled for that endpoint and notifications are sent. Endpoints events are counted across all nodes and a rate of events per minute is calculated at regular intervals. When the configured threshold of events per minute is exceeded, the endpoint is flagged as rapid and analysis is disabled for the holdtime. If refresh is enabled, an API refresh request is sent to determine the current state of the endpoint after it is no longer marked as rapid.

- **Rapid Event Threshold** number of events per minute before an endpoint is marked as rapid.
- **Rapid Holdtime** is the number of seconds to ignore new events from an endpoint marked as rapid.
- **Rapid Refresh**, when an endpoint is no longer rapid the state of db is out of sync from the fabric. When endabled, a refresh is triggered to determine the current state of the previously rapid endpoint.

Stale Analysis

When stale analysis is enabled, there are a few events that user may not wish to be treated as a stale event. A stale endpoint is generally a remote learn (XR) pointing to a leaf where the endpoint is no longer local. If the endpoint is no longer local in the fabric and the XR entry still exists on a node in the fabric, then the 'stale-no-local' logic is applied. If the endpoint is local on multiple nodes at the same time, then the last local node is assumed to be the correct entry and the 'stale-multiple-local' logic is applied. Note, this logic does not apply to vpc-attached endpoints which are expected to be learned on both nodes within the vpc.

- **Stale-no-local** enable stale-no-local detection
- **Stale-multiple-local** enable stale-multiple-local detection

Startup Event Queueing

This app heavily leverages subscriptions for keeping the app db in sync with the APIC. When the fabric monitor is started it needs to build the initial db state and setup appropriate MO subscriptions. It is possible that changes are occurring during the initial build. To capture these events, the subscriptions are started before the MO builds. After the build completes, any event received is then analyzed.The number of events queued is dependent on the rate of events and the build time. It may be desirable to ignore the events during initialization, in which case queue events can be disabled.

- **Queue initial events** enables queueing of all standard MO events during build
- **Queue initial endpoint events** enables queuing of all EPM events during endpoint build

Subscription Heartbeats

The subscription process monitors the health of the websocket and manages login token and subscription refreshes. It also polls the APIC at a regular interval to ensure that the nginx process is reachable and responsive. There is a query sent with a **hearbeat timeout** once per **heartbeat interval**. If there are consecutive heartbeat failures, up to the configured **heartbeat retry** count, then the APIC is marked as unreachable and the subscription process will restart. Set the **heartbeat interval** to 0 to disable heartbeat functionality.

Note: The fabric monitor needs to be restarted for heartbeat settings to take affect.

- **Heartbeat Retries** maximum number of successive heartbeat failures before APIC connection is declared unusable and subscription thread is closed
- **Heartbeat Interval** interval in seconds to perform heartbeat query to ensure APIC connection is available and responsive. Set to 0 to disable heartbeat.
- **Heartbeat Timeout** timeout in seconds for a single heartbeat query to complete

Session Handling

By default the APIC session is gracefully restarted based on the aaaLogin maximumLifetimeSeconds attribute. Users can override the session timeout to a value lower than the aaaLogin lifetime by setting a limit on the session time. Starting in ACI 4.0, the refresh time is configurable up to the maximum lifetime of the subscription. Increasing the refresh time reduces the number of queries sent to the APIC. This can be done by setting the Refresh Time. All nodes in the fabric must be running 4.0 or above else refresh time is limited to 60 seconds.

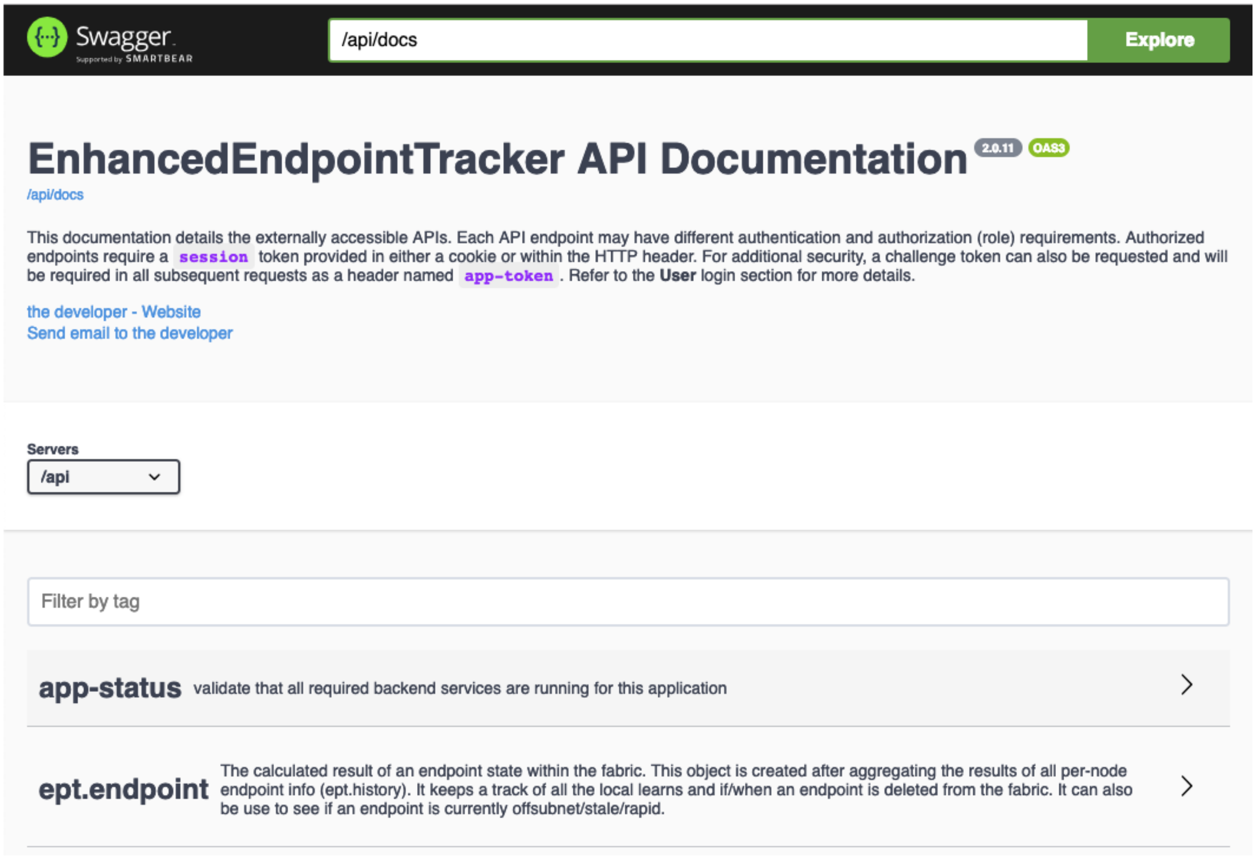
Note: The fabric monitor needs to be restarted for session settings to take affect.

- **Session Timeout** maximum time in seconds before new login and websocket is started for APIC session
- **Subscription Refresh Time** time in seconds between subscription refreshes.

CHAPTER 4

API

The EnhancedEndpointTracker app has a fully documented swagger-ui conforming to the open api 3.0 standard. You can access the swagger docs running on the app at https://APP_IP/docs.



EnhancedEndpointTracker API Documentation 2.0.11 OAS3

/api/docs

This documentation details the externally accessible APIs. Each API endpoint may have different authentication and authorization (role) requirements. Authorized endpoints require a `session` token provided in either a cookie or within the HTTP header. For additional security, a challenge token can also be requested and will be required in all subsequent requests as a header named `app-token`. Refer to the **User** login section for more details.

the developer - Website
Send email to the developer

Servers
/api

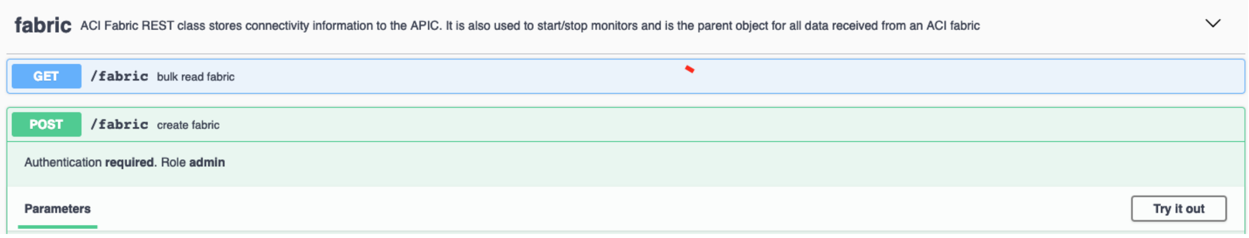
Filter by tag

app-status validate that all required backend services are running for this application

ept.endpoint The calculated result of an endpoint state within the fabric. This object is created after aggregating the results of all per-node endpoint info (ept.history). It keeps a track of all the local learns and if/when an endpoint is deleted from the fabric. It can also be use to see if an endpoint is currently offsubnet/stale/rapid.

Note: The swagger-ui page is only available when running in `standalone` mode.

The API provides paging, advanced filtering and sorting, and attribute projections. The full syntax is documented within the swagger docs. The docs also include authentication and authorization requirements per REST endpoint. For example, to create a fabric the user must be authenticated and have an `admin` role.



fabric ACI Fabric REST class stores connectivity information to the APIC. It is also used to start/stop monitors and is the parent object for all data received from an ACI fabric

GET `/fabric` bulk read fabric

POST `/fabric` create fabric

Authentication **required**. Role **admin**

Parameters

4.1 Getting Started with API

To begin you need to login and get a session identifier. Subsequent API requests must provide the session id in a cookie named `session` or an HTTP header named `session`. To login, provide a username and password to the user login API. Note, this is also documented within the swagger-ui docs under the `user` object. The example below uses curl and assumes that an instance for the app is running with HTTPS service on localhost:5000.

```
host$ curl -skX POST "https://localhost:5001/api/user/login" \
      --cookie-jar cookie.txt --cookie cookie.txt \
      -H "Content-Type: application/json" \
      -d "{\"password\":\"cisco\", \"username\":\"admin\"}"
result:
     {"session":"tBoERKg8qXbldyRZ/tkn4/9I8PUapQHDYbMepzSw1b6ZEW1NKur0JyscDv9b80Nf/
↪pZB8U4Q6megY8B++a320Q==","success":true,"token":""}
```

On successful login the web-server will reply with both the session id and an HTTP Set-Cookie header with the same value. This allows users to POST to the login url and receive the session cookie in JSON reply and within a cookie. Subsequent requests only need to reference the cookie for authentication.

For example, to get the list of configured fabrics referencing the existing cookie:

```
host$ curl -skX GET "https://localhost:5000/api/fabric?include=fabric,apic_hostname" \
      --cookie-jar cookie.txt --cookie cookie.txt \
      -H "Content-Type: application/json"
result:
{"count":1,"objects":[{"fabric":{"apic_hostname":"esc-aci-network.cisco.com:8062","dn
↪":"/uni/fb-fab4","fabric":"fab4"}}]}
```

4.2 API Access on the APIC

The APIC restricts stateful applications to use **GET** and **POST** methods only along with enforcing static URL endpoints. Since this app uses RESTful **GET**, **POST**, **PATCH**, and **DELETE** operations along with dynamic endpoints, a `proxy` mechanism was implemented to tunnel all requests through **POST** to a single URL on the APIC. This allows for seemless migration of the source code for both frontend and backend components when running on the APIC or running in standalone mode. For the proxy operation, the following three fields are required in each POST:

- **method** is the original HTTP method intended for the app (**GET**, **POST**, **PATCH**, or **DELETE**)
- **url** is the original url such as `/api/fabric`
- **data** is any required data sent in a **POST** or **PATCH** request that needs to be proxied to the backend services.

The user must also have admin read access on the APIC and use the APIC aaaLogin api to acquire a token for accessing the app API. The token must be included in all requests an HTTP header named `DevCookie`.

- If running in `mini` mode, use the following url for all request:

  https://APIC_IP/appcenter/Cisco/EnhancedEndpointTrackerMini/proxy.json

- **If running in `full` mode, use the following url for all requests:** http://APIC_IP/appcenter/Cisco/EnhancedEndpointTracker/proxy.json

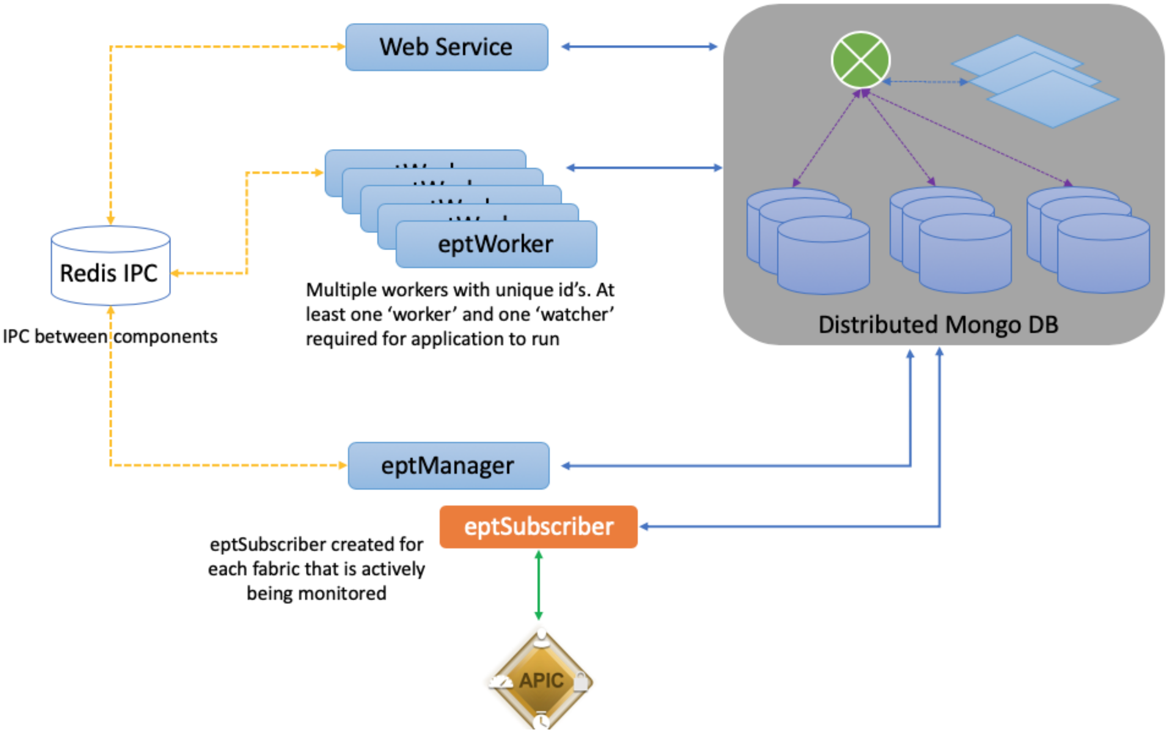
An example using curl on the APIC CLI:

```
# login to the APIC with appropriate admin credentials
apic:~> export token=`curl -skX POST "https://127.0.0.1/api/aaaLogin.json" \
 -H "Content-Type: application/json" \
 -d '{"aaaUser":{"attributes":{"name":"username", "pwd":"password"}}}' \
     | python -m json.tool | grep token | cut -d'"' -f4`

# verify that a token was acquired
apic:~> echo $token
akoAAAAAAAAAAAAAAAAAAOWxfZ8iEOFKQRpFiNNT1w2qXUFV8Gt2PyJ43FG8lGi2gu//
↪zEOU8lpWx1LNS1BG49mY6XjaXeI9m9RtgWhzEGlWWIJ7RgFBW3SOnUlbHs0kj8Xcsj0ZOxanBdWwA3c5TWDys7wpGbxVlz926M
↪cF5heh8sck7oTK4pcnu2pn7f4WDULXJ4gEA5rMWiYgtrSTiG+oeclkt4v

# read the auto-discovered fabric objects
curl --header "DevCookie: $token" \
     --header "Content-Type: application/json" \
     -skX POST "https://127.0.0.1/appcenter/Cisco/EnhancedEndpointTrackerMini/proxy.
↪json" \
     -d '{
       "method": "GET",
       "url":"/api/fabric?include=fabric,apic_hostname",
       "data": {}
     }'
result:
{"count":1,"objects":[{"fabric":{"apic_hostname":"https://172.17.0.1","dn":"/uni/fb-
↪esc-aci-fab4","fabric":"esc-aci-fab4"}}]}
```

CHAPTER 5

Components

The EnhancedEndpointTracker app is composed of several components that can be deployed as either multiple processes running within the same container in `mini` or `all-in-one` mode or as separate containers distributed over multiple nodes in `full` or `cluster` modes.



Web Service
eptWorker
Multiple workers with unique id's. At least one 'worker' and one 'watcher' required for application to run
Redis IPC
IPC between components
Distributed Mongo DB
eptManager
eptSubscriber
eptSubscriber created for each fabric that is actively being monitored
APIC

5.1 mongoDB

mongoDB `3.6` is used for persistent storage of data. In `mini` mode this is a single mongo process with journaling disabled and wireTiger cache size limit to 1.5G memory. In `cluster` mode, it runs as a distributed database utilizing mongos, configsvr in replica set, and multiple shards configured in replica sets. Sharding is enabled for a subset of collections, generally based on endpoint address.

5.2 redisDB

redisDB is in an memory key-store database. It is used as a fast IPC between components. There are two main messaging implementations in this app.

- Publish/ Subscribe mechanism allows for a component to publish a message on a channel that is received by one more subscribers
- Message queue via rpush and blpop with support for queue prioritization.

5.3 WebService

Python Flask and Apache are used for the web service.

5.4 eptManager

`eptManager` is a python process that is responsible for starting, stopping, and monitoring `eptSubscriber` proceses along with tracking the status of all available `eptWorker` processes. It is also responsible queuing and distributing all work that is dispatched to worker processes. There is only a single instance of `eptManager` deployed within the app.

5.5 eptSubscriber

`eptSubscriber` is a python process responsible for communication with the APIC. It collects the initial state from the APIC and stores into the db. It establishes and monitors a websocket to the APIC with subscriptions for all necessary MOs and ensures the db is in sync with the APIC. `eptSubscriber` process also subscribes to all epm events and dispatches each event to `eptManager` which will enqueue to an appropriate `eptWorker` process to analyze the event. There is a single `eptSubscriber` process running for each configured fabric. This process is always a subprocess running in the same container as `eptManager`.

The following objects are collected and monitored by the subscriber process:

- datetimeFormat
- epmIpEp
- epmMacEp
- epmRsMacEpToIpEpAtt
- fabricAutoGEp
- fabricExplicitGEp
- fabricNode
- fabricProtPol

- fvAEPg
- fvBD
- fvCtx
- fvIpAttr
- fvRsBd
- fvSubnet
- fvSvcBD
- l3extExtEncapAllocator
- l3extInstP
- l3extOut
- l3extRsEctx
- mgmtInB
- mgmtRsMgmtBD
- pcAggrIf
- pcRsMbrIfs
- tunnelIf
- vnsEPpInfo
- vnsLIfCtx
- vnsRsEPpInfoToBD
- vnsRsLIfCtxToBD
- vpcRsVpcConf

5.6 eptWorker

> There is a configurable number of `eptWorker` processes that can be executed. Each `eptWorker` must have a unique id and will be deployed with a role of either a **worker** or **watcher** process. `eptManager` requires at least one active `eptWorker` for all roles before it can start any fabric monitors. The `eptWorker` **worker** process performs the bulk of the computations for the app. It receives epm events and performs move/offsubnet/stale/rapid analysis and stores the results into the db. If an endpoint is flagged by one of the analyses, a message is sent to `eptManager` to enqueue to an `eptWorker` **watcher** process. The **watcher** will perform the configure notifications along with executing rechecks to prevent incorrect detection of transitory events.

The full source code for the Flask web-service implementation and all ept components is available on Github.

CHAPTER 6

Releases

This page will track information about each new release along with new features and any known issues.

6.1 Version 2.1.2a

Released Sep 12 2019

- fix for Docker swarm node IP discovery
- change default worker count from 10 to 23 for swarm
- ensure web service is closing redis connections for each pub/sub message
- ensure all messages are sent on support qnum (regression from queue count reduction fixes)
- force username for ssh remote login procedures to ensure custom domain is maintained

6.2 Version 2.1.2

Released Aug 20 2019

- fix for #46 fabricNode parsing broken on recent build breaks all eptNode tables
- update worker hash calculation for better load-balancing of work in scale setups
- use pub/sub redis messaging for broadcast between subscriber and workers
- reduce queue count and queue stat objects when using pub/sub broadcasts
- address timeouts on techsupport collection in multi-node standalone cluster
- reduce memory utilization on startup by streaming class queries
- include sort on all concrete objects initializations to prevent out-of-order results during paging

- use local user directory instead of global tmp for compose and deploy logs to prevent access problems for multiuser deployments
- additional user validation checks during docker swarm deployment
- increase manager timeout on app-status to support scale setups
- fix mongo cursor timeouts due to docker routed-mesh mongos load-balancing in multi-node standalone cluster
- propagate session/subscriber errors into fabric events so user can quickly isolate subscription restart errors
- added UI and backend support for configurable heartbeat interval, timeout, and retry count
- improve queue cleanup on fabric restart
- logrotate updates for apache logs
- UI polling service fix to prevent rapid requests on 400/500 errors
- UI optimize fabric status and manager polling mechanisms
- UI fix for cascading polling events on refresh

6.3 Version 2.1.1

Released Mar 21 2019

- Configurable SMTP settings including custom SMTP port, TLS encryption, and support for SMTP relay with optional authentication
- Configurable session timeout
- Configurable Refresh time for subscriptions. Staring in ACI 4.0 the subscription refresh time can be extended up to 10 hours but will be limited to 60 seconds in older versions. The app will auto-detect the version of code and apply a max of 60 seconds if APIC or switch is <4.0. Previously this was static at 30 seconds.
- Moved managed object event handling from subscriber process to dedicated worker. This addresses issues such as high rate of tunnelIf events during initial build that can cause subscriber process to hang.
- Cross reference fabricNode and topSystem to ensure inactive nodes are included in initial build. This resolves a bug TEP calculation for vpcs if one node in the vpc is offline when app starts
- Disable accurate queue-length calculation on UI as it impacts manager process under scale
- Updates to app deployment scripts to allow user to pass in all arguments via command-line in addition to editing cluster yaml file
- Trigger fabric restart on worker heartbeat failure and new worker detection
- Removed exim4 dependencies and rely on native python smtplib for email notifications

6.4 Version 2.0.x

Released Feb 22 2019

Initial 2.0 release. This was a complete rewrite with focus on scalability while maintaining and augmenting the feature set from the 1.x versions. Features include:

- Support for full endpoint scale fabric
- Easy to use GUI with fast type-ahead search for any mac/ipv4/ipv6 endpoint in the fabric

- Full details on current state of an endpoint in the fabric along with historical information
- Move, offsubnet, rapid, and stale endpoint analysis
- Configurable notification options via syslog/email for various events detected by the app
- Capability to clear an endpoint on one or more nodes via GUI or API
- Distributed architecture allowing app to grow based on the size of the fabric
- Fully documented swagger API to allow for easy integration with other tools
- Flexible deployment options include APIC and APIC-mini apps along with standalone all-in-one container and standalone docker-swarm cluster.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search