EdgeX 2.0 Security Overview

References

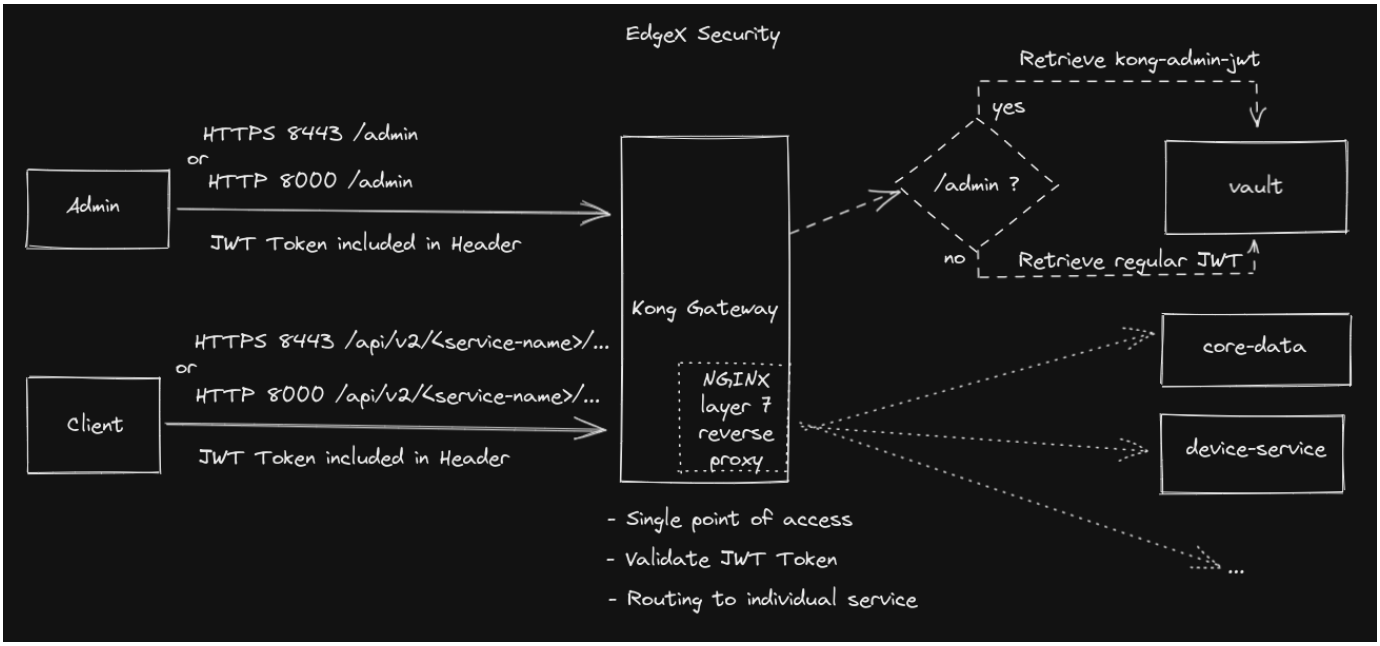
1. EdgeX Go's Security, https://github.com/edgexfoundry/edgex-go/blob/main/SECURITY.md
2. EdgeX Foundry Security API-Gateway Service, https://github.com/edgexfoundry/security-api-gateway
3. EdgeX Security Store, https://docs.edgexfoundry.org/2.0/security/Ch-SecretStore/
4. EdgeX API Gateway, https://docs.edgexfoundry.org/2.0/security/Ch-APIGateway/
5. Kong Datastore, https://konghq.com/faqs/
6. Kong DB-less Mode, https://docs.konghq.com/gateway-oss/2.5.x/db-less-and-declarative-config/
7. EdgeX Foundry Core Snap, https://github.com/edgexfoundry/edgex-go/blob/main/snap/README.md#jwt-tokens

Current issues with EdgeX

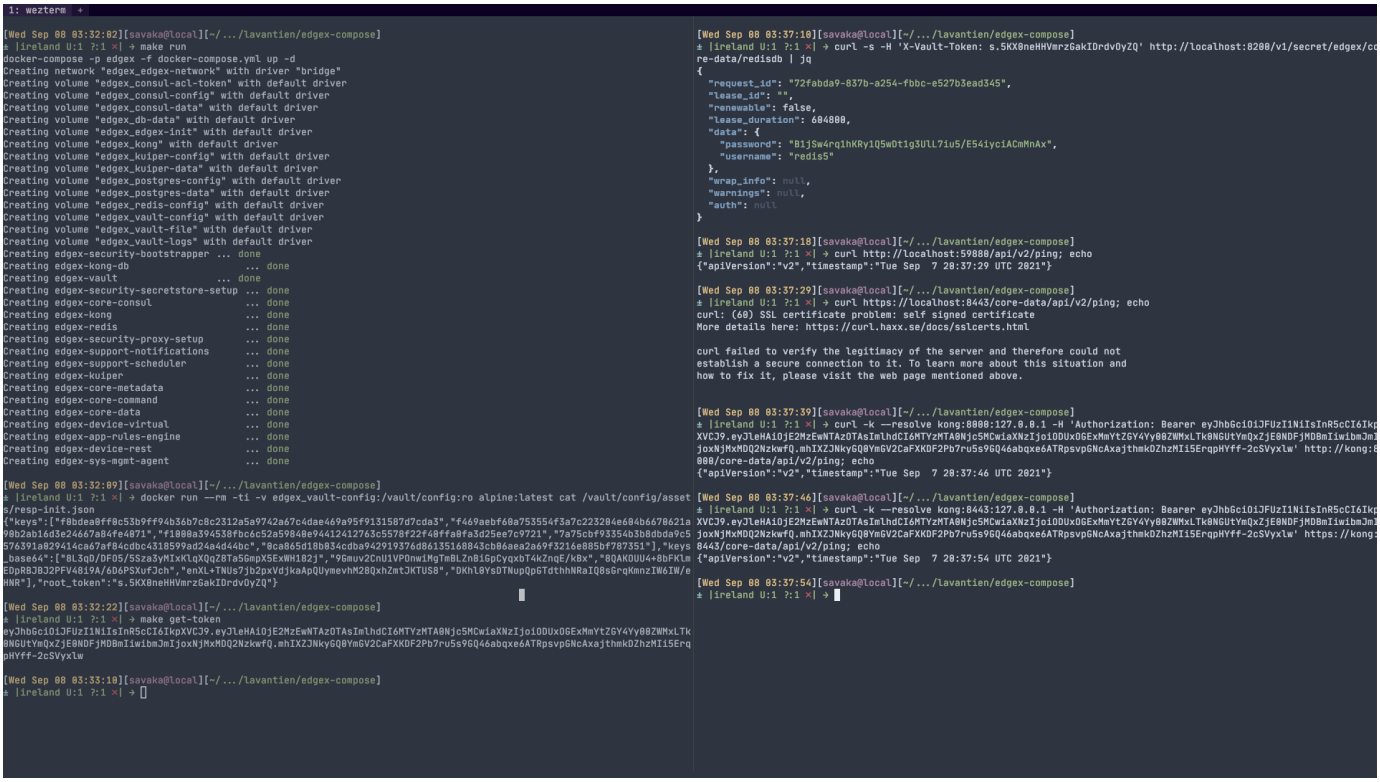
- The TLS materials path doesn't compatible to the official instruction when tested

edgex/pki/tls/edgex-long vs **edgex/ <no pki or kong-related directory exists>**

Diagrams



Targets



There are two major EdgeX security components

- The first is a security store, which is used to provide a safe place to keep the EdgeX secrets
- The second is an API gateway, which is used as a reverse proxy to restrict access to EdgeX REST resources and perform access control related works
- Current features

1. Secret creation, store and retrieve (password, cert, access key etc.)
2. API gateway for other existing EdgeX microservice REST APIs
3. User account creation with optional either OAuth2 or JWT authentication
4. User account with arbitrary Access Control List groups (ACL)

Secret Store

- Utilized to create, store and retrieve secrets relevant to the corresponding microservices
- Implemented in Vault (https://www.vaultproject.io/), with Consule as the secrets engine, store in file system
- Preferred approach is to use `Secret Client` interface in `go-mod-secrets`, each microservices have two methods for accessing the client that is, `StoreSecrets()` and `GetSecrets()`
- For security reason the Vault root token is revoked by default, microservices have to interact with their own partition of the secret store programmatically
- If global access to the secret store is required, it is necessary to obtain a copy of the Vault root token using the below recommended procedure. Note that following this procedure directly contradicts the Vault production hardening guide. **Since the root token cannot be un-revoked, the framework must be started for the first time with root token revokation disabled**
- When starting a secure EdgeX deployment, the sequence is defined by the EdgeX secure bootstrapping Architecture Decision Record. In general the sequence is:

1. Start a bootstrapping component that sequences the framework startup
2. Start the EdgeX secret store component
3. Start the `secretstore-setup` container to initialize the secret store and create shared secrets (such as database passwords) needed by the microservices
4. Start the other stateful components of EdgeX
5. Start the non-stateful EdgeX services
6. Start the `proxy-setup` container to configure the Kong API gateway

- The startup sequence is automatically coordinated to boot in the proper order to ensure correct functioning of EdgeX

Get the Root Token with these procedures

1. Get the docker-compose folder and run the first time for downloading images branch ireland (EdgeX 2.0)

```
$ git clone -b ireland https://github.com/edgexfoundry/edgex-compose
$ cd edgex-compose
$ make run
```

1. Shut down the entire framework and remove the Docker persistent volumes using `make clean` in `edgex-compose` or `docker volume prune` after stopping all the containers. Optionally remove `/tmp/edgex` as well to clean the shared secrets volume
2. Edit `docker-compose.yml` and add an environment variable override for `SECRETSTORE_REVOKEROOTTOKENS`

```
secretstore-setup:
    environment:
      SECRETSTORE_REVOKEROOTTOKENS: "false"
```

1. Start EdgeX using `make run`
2. Reveal the contents of the `resp-init.json` file stored in a Docker volume

```
$ docker run --rm -ti -v edgex_vault-config:/vault/config:ro alpine:
latest cat /vault/config/assets/resp-init.json
```

1. Extract the `root_token` field value from the resulting JSON output

```
{
    "keys": [

"516f1e96fa683af5be5d176c0fe60261796d1f99e715ec81570c7c39c3a56f7c4c",

"a1a6ae67149801038464366b8f0c3b911edc80d0e4ab12d9285051ecb572e6335e",

"5750910d748eacd0e7a47a5fd4311f3be80df7d6d8eeba78614763346d5168cc39",

"41a23879e06cbd663442e196e4c755499490a59db199dc16d735768467ec4bb612",

"bb1207e1ae6676eea967b4f40b8e3d88ff666d416a73d1eea66c279ae7facd6903"
    ],
    "keys_base64": [
        "UW8elvpoOvW+XRdsD+YCYXltH5nnFeyBVwx8OcOlb3xM",
        "oaauZxSYAQOEZDZrjww7kR7cgNDkqxLZKFBR7LVy5jNe",
        "V1CRDXSOrNDnpHpf1DEfO+gN99bY7rp4YUdjNG1RaMw5",
        "QaI4eeBsvWY0QuGW5MdVSZSQpZ2xmdwW1zV2hGfsS7YS",
        "uxIH4a5mdu6pZ7T0C449iP9mbUFqc9Hupmwnmuf6zWkD"
    ],
    "root_token": "s.i9nvh0gTZ5MOJKrrInNF5Jdp"
}
```

1. As an alternative to overriding `SECRETSTORE_REVOKEROOTTOKENS` from the beginning, it is possible to regenerate the root token from the Vault unseal keys in `resp-init.json` using the Vault's documented procedure.
   The EdgeX framework executes this process internally whenever it requires root token capability.
   Note that a token created in this manner will again be revoked the next time EdgeX is restarted if `SECRETSTORE_REVOKEROOTTOKENS` remains set to its default value: all root tokens are revoked every time the framework is started if `SECRETSTORE_REVOKEROOTTOKENS` is `true`

Accessing the running Vault container

```
$ docker exec -it edgex-vault sh -l
```

Login to Vault and get the Root Token, **the Root Token is the only token that has no expiration enforcement rules (Time to Live TTL counter)**

```
edgex-vault:/# vault login s.i9nvh0gTZ5MOJKrrInNF5Jdp
Success! You are now authenticated. The token information displayed
below
is already stored in the token helper. You do NOT need to run "vault
login"
again. Future Vault requests will automatically use this token.

Key                  Value
---                  -----
token                s.i9nvh0gTZ5MOJKrrInNF5Jdp
token_accessor       vZ2Oz4IPWCl5BIQ2qDr7ZGtV
token_duration
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

Validate if the token actually works

```
edgex-vault:/# vault token lookup
Key                 Value
---                 -----
accessor            vZ2Oz4IPWCl5BIQ2qDr7ZGtV
creation_time       1630990247
creation_ttl        0s
display_name        root
entity_id           n/a
expire_time         <nil>
explicit_max_ttl    0s
id                  s.i9nvh0gTZ5MOJKrrInNF5Jdp
meta                <nil>
num_uses            0
orphan              true
path                auth/token/root
policies            [root]
ttl                 0s
type                service
```

Try lookup for the microservices' credentials, **With the root token, it is possible to modify any Vault setting**

```
edgex-vault:/# vault list secret
Keys
----
edgex/

edgex-vault:/# vault list secret/edgex
Keys
----
app-rules-engine/
core-command/
core-data/
core-metadata/
device-rest/
device-virtual/
security-bootstrapper-redis/
support-notifications/
support-scheduler/

edgex-vault:/# vault list secret/edgex/core-data
Keys
----
redisdb

edgex-vault:/# vault read secret/edgex/core-data/redisdb
Key                 Value
---                 -----
refresh_interval    168h
password            n2uMDpHOCvzmygvJu06Hyk6TMNCl+DKXVYtDAiKgzD3E
username            redis5
```

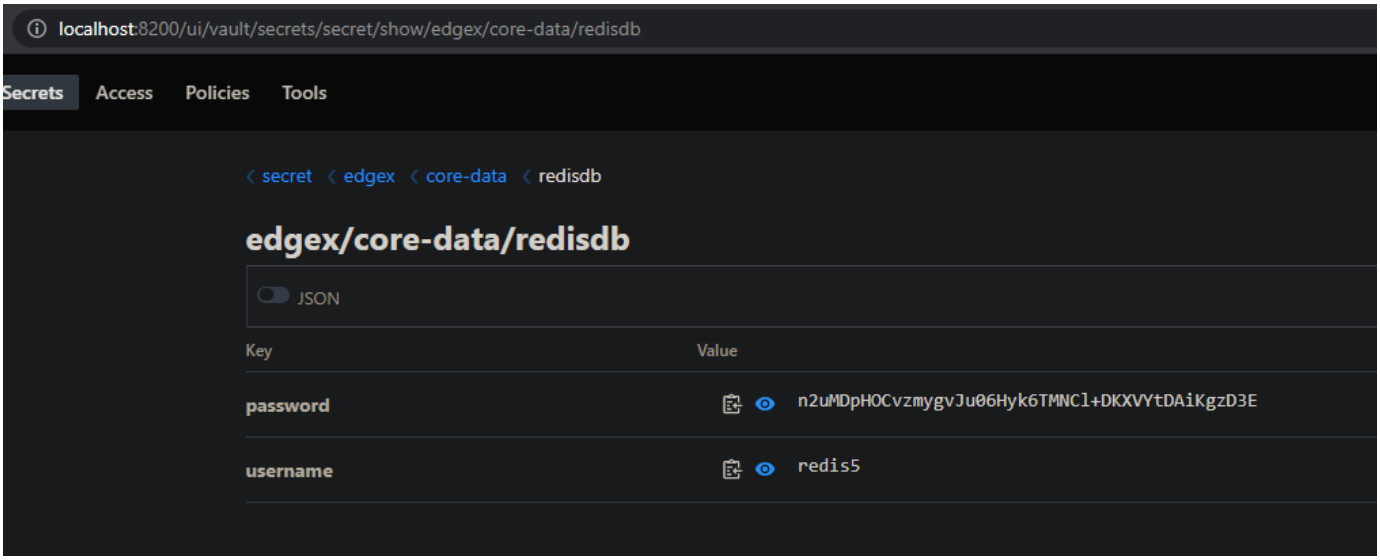
- There is a Vault REST API (https://www.vaultproject.io/api) with equivalent commands to the CLI

```
vault read secret/edgex/core-data/redisdb
```

```
$ curl -s -H 'X-Vault-Token: s.i9nvh0gTZ5MOJKrrInNF5Jdp'
http://localhost:8200/v1/secret/edgex/core-data/redisdb | python -m
json.tool

{
    "auth": null,
    "data": {
        "password": "n2uMDpHOCvzmygvJu06Hyk6TMNCl+DKXVYtDAiKgzD3E",
        "username": "redis5"
    },
    "lease_duration": 604800,
    "lease_id": "",
    "renewable": false,
    "request_id": "7f506877-6afe-e8bf-9a07-83bf0bd92c16",
    "warnings": null,
    "wrap_info": null
}
```

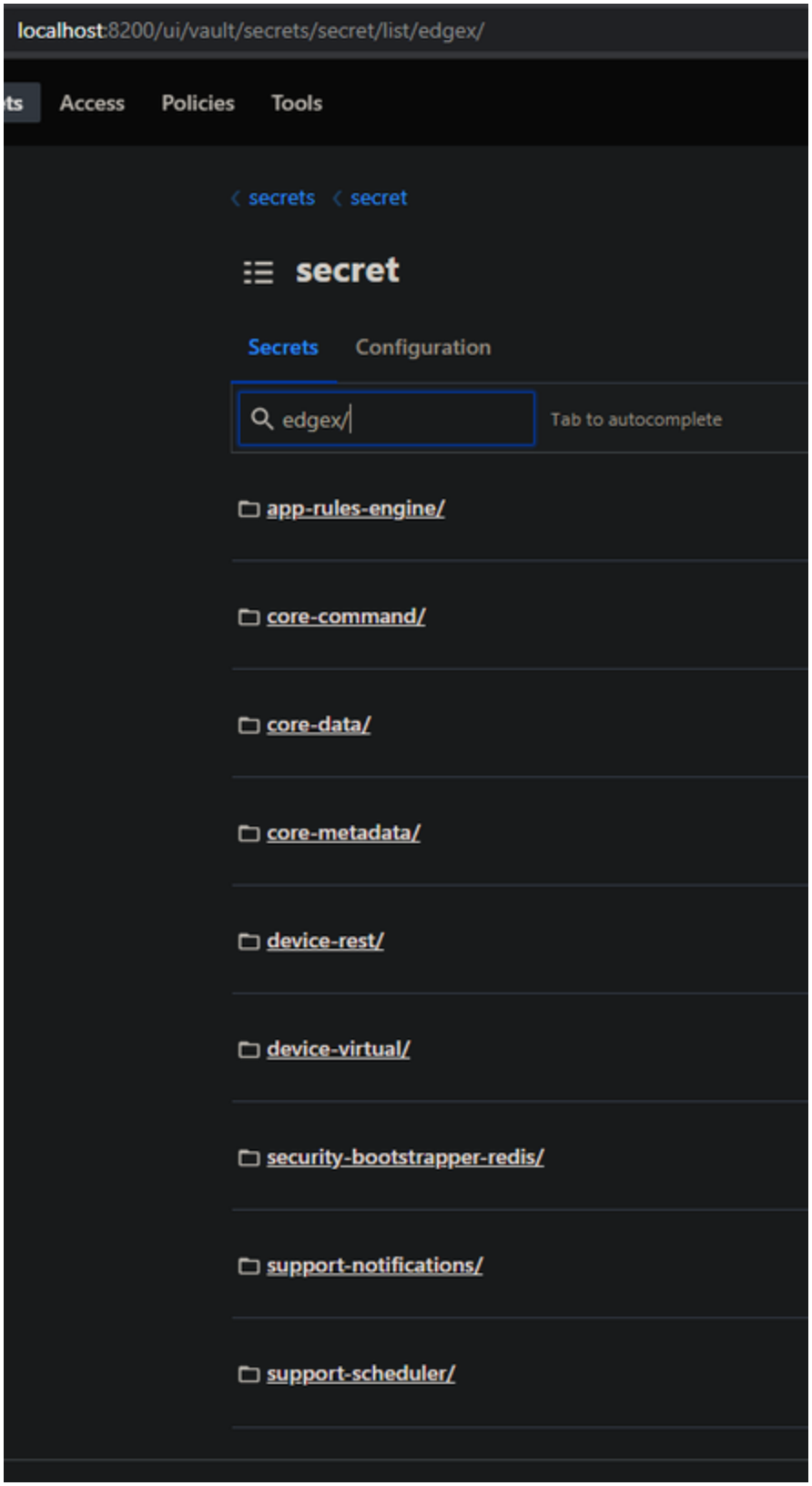
- The Vault Web UI locate at `http://localhost:8200` and is signed-in with the Root Token

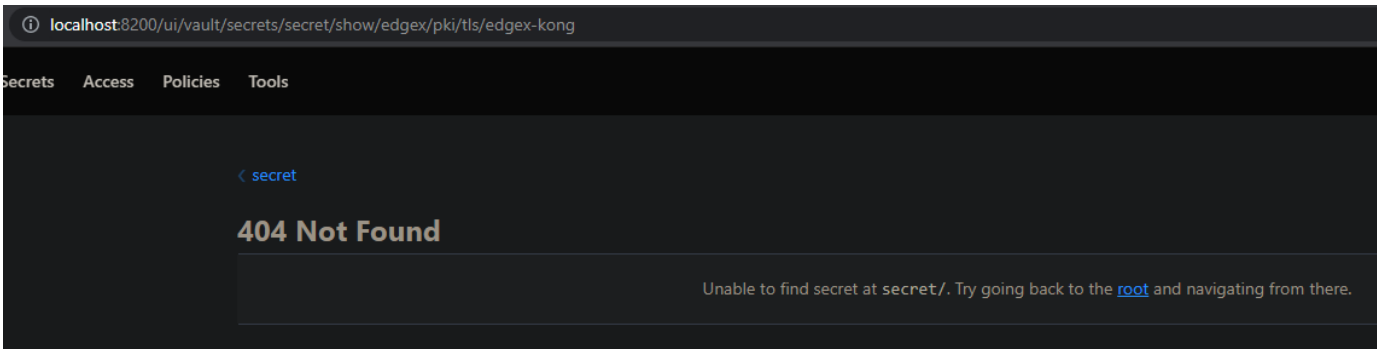


Navigate the API Gateway (Kong) service X.509 TLS materials path **edgex/pki/tls/edgex-kong**

The Vault UI also allows entering Vault CLI commands (see above **1st alternative**) using an embedded console

The PATH doesn't compatible to the official instruction when tested


localhost:8200/ui/vault/secrets/secret/list/edgex/
Access
Policies
Tools
secrets
secret
secret
Secrets
Configuration
edgex/
Tab to autocomplete
app-rules-engine/
core-command/
core-data/
core-metadata/
device-rest/
device-virtual/
security-bootstrapper-redis/
support-notifications/
support-scheduler/



The API Gateway

- The single point of entry for all EdgeX REST traffic, barrier between external clients and the EdgeX microservices preventing unauthorized access to EdgeX REST APIs
- The API gateway accepts client requests, verifies the identity of the clients, redirects the requests to correspondent microservice and relays the results back to the client
- **Internally, no authentication is required for one EdgeX microservice to call another**
- The API Gateway provides two HTTP REST management interfaces

1. The first (insecure) interface is exposed only to `localhost:`, in Docker this insecure interface is bound to the Docker container, and is **not reachable from outside** of the container
2. The second (secure) interface is exposed outside of cluster on an administative URL sub-path, `/admin` and **requires a specifically-crafted JWT to access**
   The management interface offers the means to configure API routing, as well as client authentication and access control
   This configuration is stored in an embedded database

- KONG (https://konghq.com/) is the product underlying the API gateway. The EdgeX community has added code to initialize the KONG environment, set up service routes for EdgeX microservices, and add various authentication/authorization mechanisms including JWT authentication and ACL

Kong Datastore

Kong uses an external datastore to store its configuration such as registered APIs, Consumers and Plugins. Plugins themselves can store every bit of information they need to be persisted, for example rate-limiting data or Consumer credentials

Kong maintains a cache of this data so that there is no need for a database roundtrip while proxying requests, which would critically impact performance. This cache is invalidated by the inter-node communication when calls to the Admin API are made. As such, it is discouraged to manipulate Kong's datastore directly, since your nodes cache won't be properly invalidated

This architecture allows Kong to scale horizontally by simply adding new nodes that will connect to the same datastore and maintain their own cache. As a result, Kong gateway and plugin performance is sub-millisecond for most use-cases

PostgreSQL is a good candidate for single instance or centralized setups due to its relative simplicity and strong performance. Many cloud providers can host and scale PostgreSQL instances

Kong DB-less Mode

Kong 1.1 added the capability to run Kong without a database, using only in-memory storage for entities: we call this **DB-less mode**. When running Kong DB-less, the configuration of entities is done in a second configuration file, in YAML or JSON, using **declarative configuration**

Drawbacks:

1. Memory Cache Requirements
2. No Central Database Coordination
3. Read-Only Admin API
4. Plugin Compatibility

Start the API Gateway

From the docker-compose file (https://github.com/edgexfoundry/edgex-compose/tree/ireland) the API Gateway will use the secure version by default, download the non `no-secty`

The API Gateway is provided by the `kong` service. The `proxy-setup` service is a one-shot service that configures the proxy and then terminates. `proxy-setup` docker image also contains the `secrets-config` utility to assist in common configuration tasks

```
$ make run
```

Configuring API Gateway

The API gateway will generate a default self-signed TLS certificate that is used for external communication

the X.509 PEM-encoded certificate is assumed to be called `cert.pem` and the unencrypted PEM-encoded private key is called `key.pem`

Do not use an encrypted private key as the API gateway will hang on startup in order to prompt for a password

If there is a need to install a custom certificate, with the external DNS name of the API gateway is `edge001.example.com`, the API gateway also requires client to support Server Name Identification (SNI)

```
$ docker-compose -p edgex -f docker-compose.yml run --rm -v `pwd`:/host:
ro --entrypoint /edgex/secrets-config edgex-proxy proxy tls --incert
/host/cert.pem --inkey /host/key.pem --snis edge001.example.com
```

Verify it

```
$ echo "GET /" | openssl s_client -showcerts -servername edge001.
example.com -connect 127.0.0.1:8443
```

Configuration of JWT Authentication for API Gateway

When using JWT Authentication, the [KongAuth] section needs to be specified in the configuration file as shown below. This is the default

The configuration file is <EdgeX Go source code folder>/cmd/security-proxy-setup/res/configuration.toml

```
[KongAuth]
Name = "jwt"
```

The "oauth2" authentication method has been removed in EdgeX 2.0 as JWT-based authentication is resistant to brute-force attacks and does not require storage of a secret in the Kong database

Configuration of Adding Microservices Routes for API Gateway

Adding new proxy Kong routes are now possibly achieved via the environment variable, `ADD_PROXY_ROUTE`, of service `edgex-proxy` in the docker-compose file

```
proxy-setup:
      ...
    environment:
      ...
      ADD_PROXY_ROUTE: "edgex-core-data.http://edgex-core-data:59880"
      ...

...

data:
   ...
   container_name: edgex-core-data
   hostname: edgex-core-data
   ports:
    - 127.0.0.1:5563:5563/tcp
    - 127.0.0.1:59880:59880/tcp
   ...
```

The value of `ADD_PROXY_ROUTE` takes a comma-separated list of one or more (at least one) paired additional service name and URL for which to create proxy Kong routes, **case insensitive**

`<RoutePrefix>.<TargetRouteURL>`

It is the docker network hostname of the service that want to add the proxy Kong route in the docker-compose file if running from docker-compose

Once `ADD_PROXY_ROUTE` is configured and composed-up successfully, the proxy route then can be accessed the app's REST API via Kong as h`ttps://localhost:8443/core-data/api/v2/...` in the same way of accessing the EdgeX service. Need an access token obtained using the steps below

Using the API Gateway

If the EdgeX API gateway is not in use, a client can access and use any REST API provided by the EdgeX microservices by sending an HTTP request to the service endpoint

```
$ curl http://localhost:59880/api/v2/ping
{"apiVersion":"v2","timestamp":"Wed Sep  1 08:11:25 UTC 2021"}
```

Once the API gateway is started and initialized successfully, and all the common ports for EdgeX microservices are blocked by disabling the exposed external ports of the EdgeX microservices through updating the docker compose file

```
$ curl https://localhost:8443/core-data/api/v2/ping

# If doesn't have any certificate
curl: (77) schannel: next InitializeSecurityContext failed:
SEC_E_UNTRUSTED_ROOT (0x80090325) - The certificate chain was issued by
an authority that is not trusted.

# If already have self signed certificate
curl: (60) SSL certificate problem: self signed certificate
More details here: https://curl.haxx.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could
not
establish a secure connection to it. To learn more about this situation
and
how to fix it, please visit the web page mentioned above.
```

The differents

1. `http` is switched to `https` as we enable the SSL/TLS for secure communication. This applies to any client side request
2. The API gateway will eventually relay the request to the Core Data service if the client is authorized. Note that for Kong to serve the proper TLS certificate, **a DNS host name must be used as SNI does not support IP addresses**. This is a standards-compliant behavior, not a limitation of Kong
3. The port switched from 48080 to 8443, which is the default SSL/TLS port for API gateway (versus the micro service port). This applies to any client side request
4. The `/core-data/` path in the URL is used to identify which EdgeX micro service the request is routed to. As each EdgeX micro service has a dedicated service port open that accepts incoming requests, there is a mapping table kept by the API gateway that maps paths to micro service ports

```
Microservice Host Name              Port number        Partial URL
edgex-core-data                      59880               core-data
edgex-core-metadata                  59881               core-metadata
edgex-core-command                  59882              core-command
edgex-support-notifications          59860               support-
notifications
edgex-support-scheduler              59861               support-
scheduler
edgex-kuiper                       59720               rules-engine
device-virtual                       59900               device-virtual
```

API Gateway TLS certificate setup

By default Kong is configured with an EdgeX signed TLS certificate. Client validation of this certificate requires the root CA certificate from the EdgeX instance. This file (`ca.pem`) can be copied from directory `$SNAP_DATA/secrets/ca`.

It is also possible to install your own TLS certificate to be used by the gateway. The steps to do so are as follows:

Start by provisioning a TLS certificate to use. One way to do so for testing purposes is to use the edgeca snap:

```
$ sudo snap install edgeca
$ edgeca gencsr --cn localhost --csr csrfile --key csrkeyfile
$ edgeca gencert -o localhost.cert -i csrfile -k localhost.key
```

Then install the certificate:

```
$ sudo snap set edgexfoundry env.security-proxy.tls-certificate="$(cat
localhost.cert)"
$ sudo snap set edgexfoundry env.security-proxy.tls-private-key="$(cat
localhost.key)"
```

This sample certificate is signed by the EdgeCA root CA, so by specifying the Root CA certificate for validation then a connection can now be made using your new certificate:

```
$ curl -v --cacert /var/snap/edgeca/current/CA.pem -X GET
https://localhost:8443/coredata/api/v1/ping? -H "Authorization: Bearer
$TOKEN"
```

Creating Access Token for API Gateway Authentication

The API gateway is configured to require authentiation prior to passing a request to a back-end microservice

JWT authentication is based on a public/private keypair, where the public key is registered with the API gateway, and the private key is kept secret. This method does not require exposing any secret to the API gateway and allows JWTs to be generated offline

Before using the JWT authentiation method, it is necessary to create a public/private keypair.

Then, generate and save a unique ID that will be used in any issued JWTs to look up the public key to be used for validation. Also we need the JWT used to authenticate to Kong--this JWT was written to host-based secrets area when the framework was started.

Register a user for that key

Lastly, generate a valid JWT. Any JWT library should work, but secrets-config provides a convenient utility

All of the above steps can be done with

```
$ make get-token
<JWT>
```

The command will output a long alphanumeric sequence of the format

`<alphanumeric> '.' <alphanumeric> '.' <alphanumeric>`

The access token is used in the Authorization header of the request (see details below)

Refresh Token? Note that the `get-token` Makefile target by default generates a public/private keypair in a temporary folder that is **deleted** after the token is created. ***For production usage, one would want to keep this keypair around and use it to generate future access tokens***.

To de-authorize or delete the user

```
$ docker-compose -p edgex -f docker-compose.yml run --rm -u "$UID" --
entrypoint "/edgex/secrets-config" proxy-setup -- proxy deluser --user
_SOME_USERNAME_ --jwt "$KONGJWT"
```

Using API Gateway to Proxy Existing EdgeX Microservices

```
$ token=`make get-token`
$ curl -k -H "Authorization: Bearer $token" https://localhost:8443/core-
data/api/v2/ping
```

(Note the above `--resolve` line forces "kong" to resolve to 127.0.0.1. This is just for illustrative purposes to force SNI. In practice, the TLS certificate would be registered under the external host name)

Or using HTTP via 8000 port

```
$ token=`make get-token`
$ curl -k -H "Authorization: Bearer $token" http://localhost:8000/core-
data/api/v2/ping
```

Summary

1. --resolve tells curl to resolve https://kong:8443 to the loopback address. This will cause curl to use the hostname `kong` as the SNI, but connect to the specified IP address to make the connection. -k tells curl to ignore certificate errors. This is for demonstration purposes. In production, a known certificate that the client trusts be installed on the proxy and this parameter omitted
2. --H "host: edgex" is used to indicate that the request is for EdgeX domain as the API gateway could be used to take requests for different domains
3. Use the https versus http protocol identifier for SSL/TLS secure communication
4. The service port 8443 is the default TLS service port of API gateway
5. Use the URL path "coredata" to indicate which EdgeX microservice the request is routed to
6. Use header of -H "Authorization: Bearer \<access-token>" to specify the access token associated with the client that was generated when the client was added
7. `https` seems not to work on a Mac machine.