BACKEND.AI CODE BASE SEMINAR

Sujin Kim

OVERALL ARCHITECTURE

BACKEND.

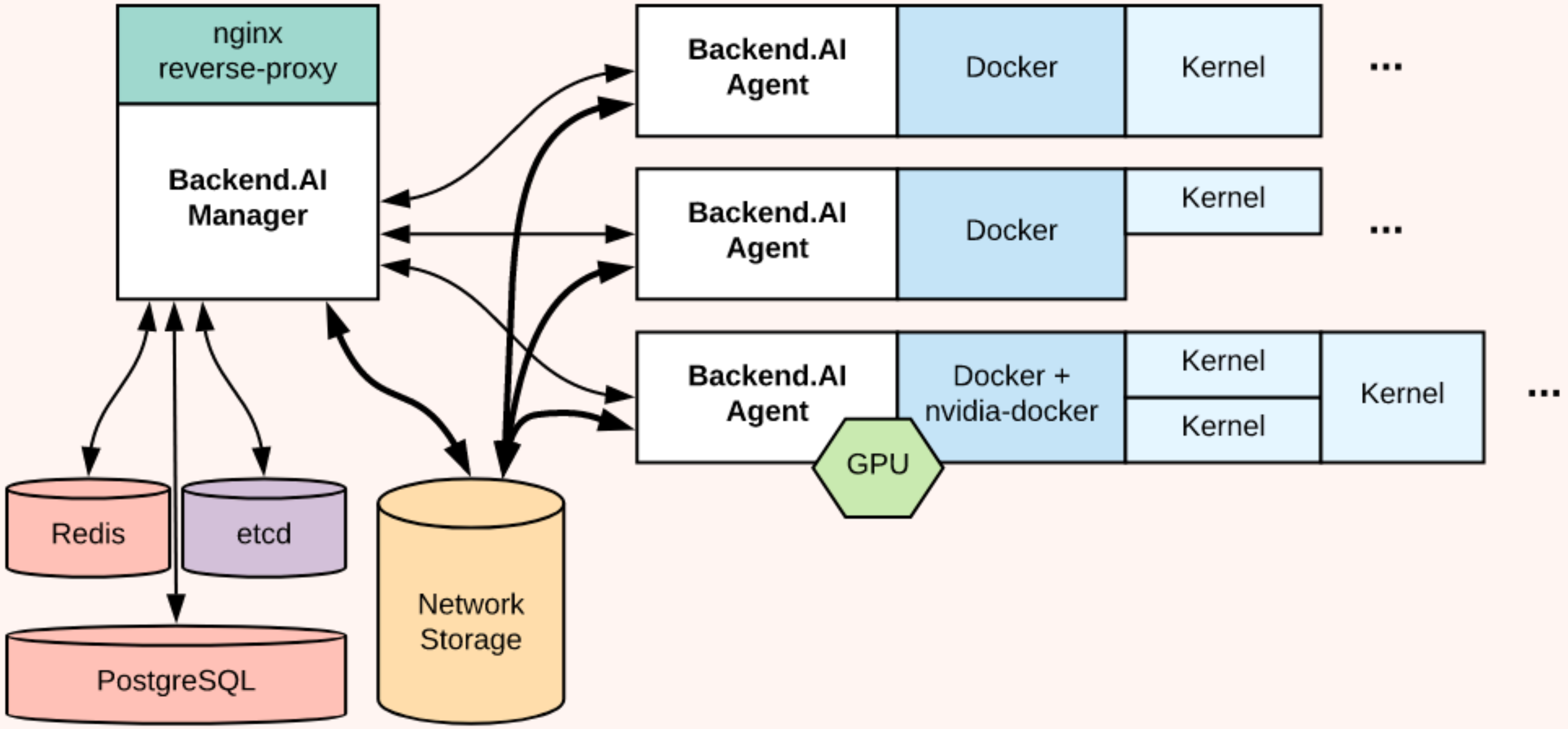
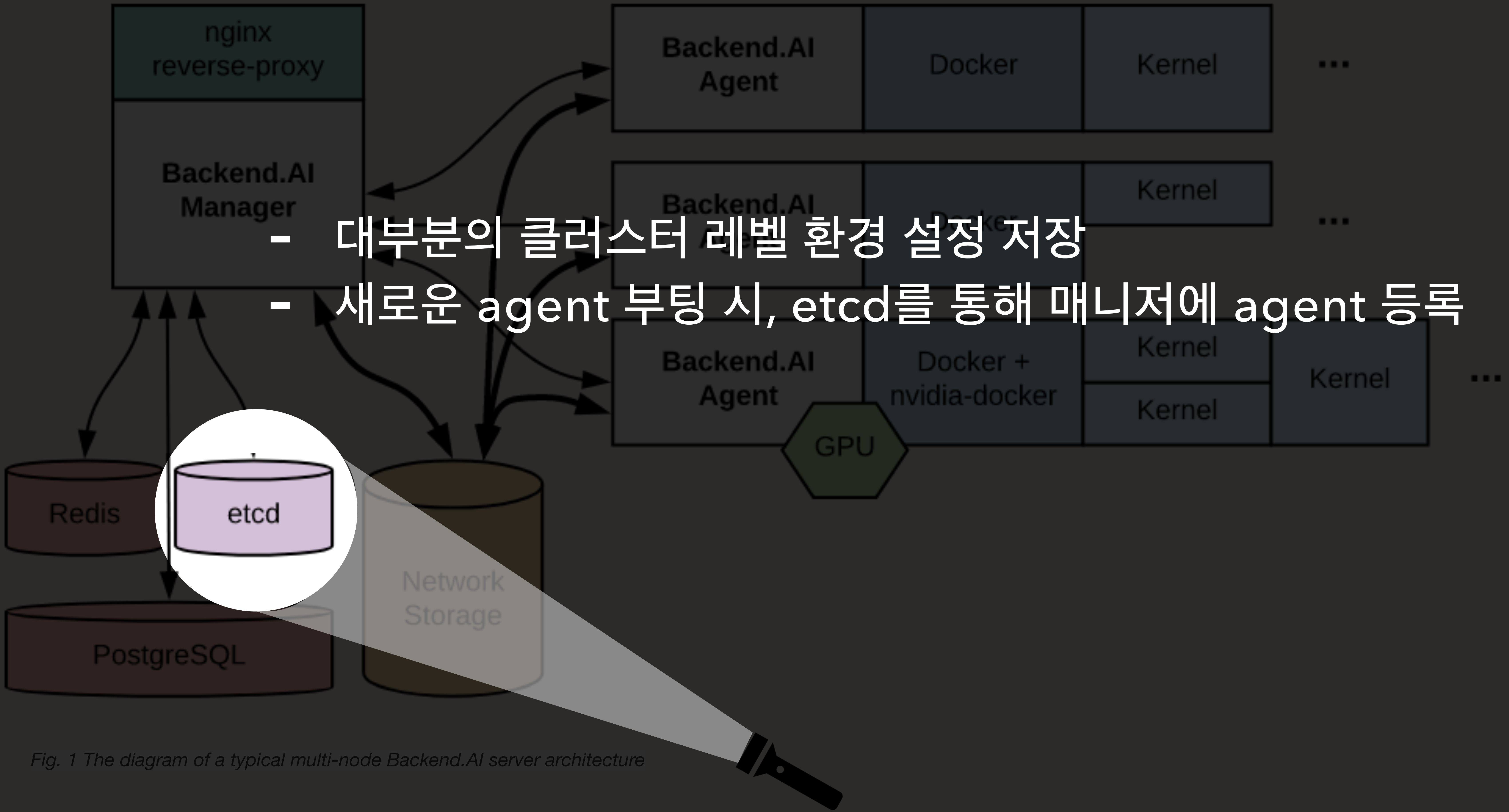


Fig. 1 The diagram of a typical multi-node Backend.AI server architecture



- 대부분의 클러스터 레벨 환경 설정 저장
- 새로운 agent 부팅 시, etcd를 통해 매니저에 agent 등록

Fig. 1 The diagram of a typical multi-node Backend.AI server architecture

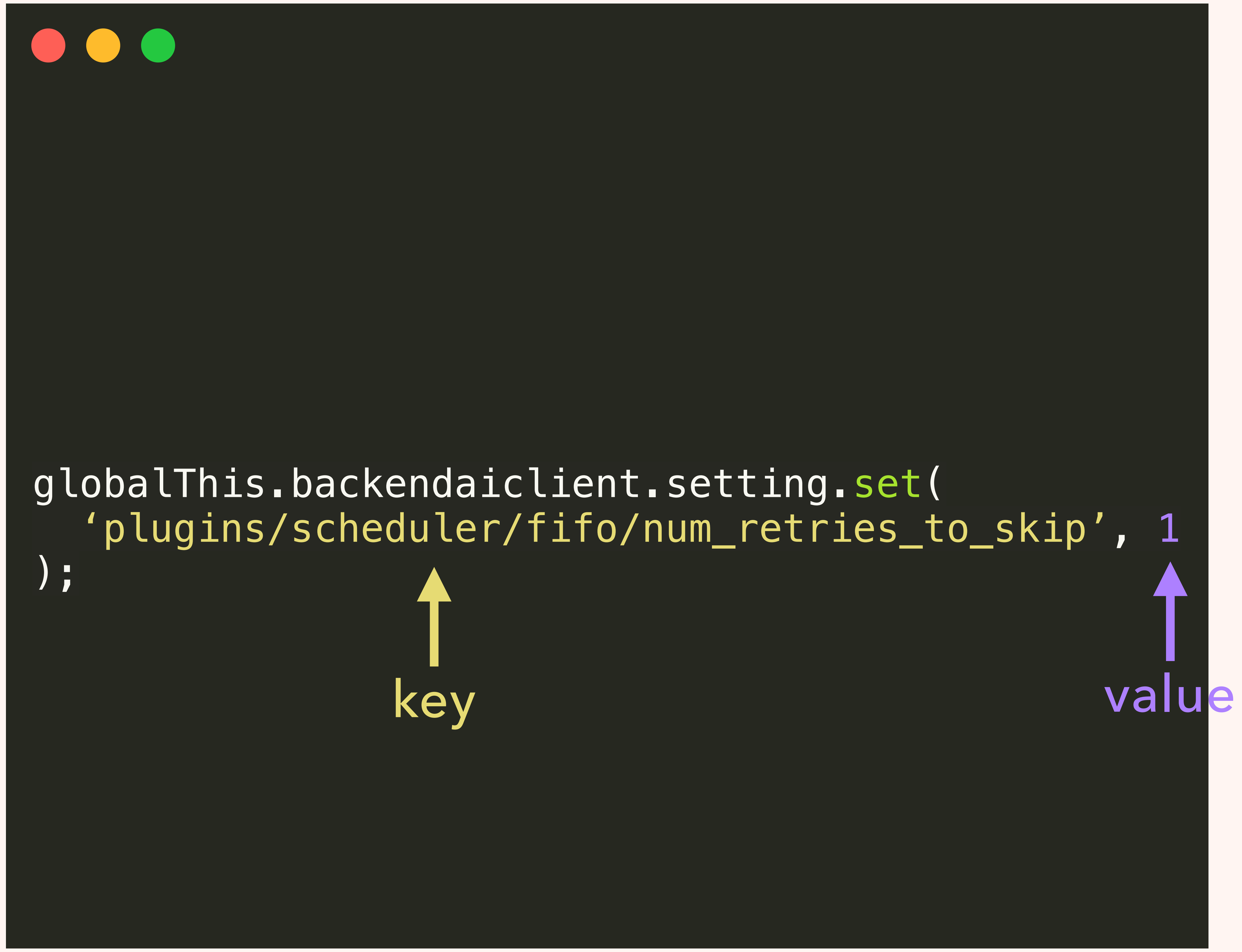
WHAT IS ETCD?

“etcd는 분산 시스템이나 클러스터가 접근해야 하는 데이터를 일관되고 분산된 방법으로 저장한 키 - 값 저장소이다.”

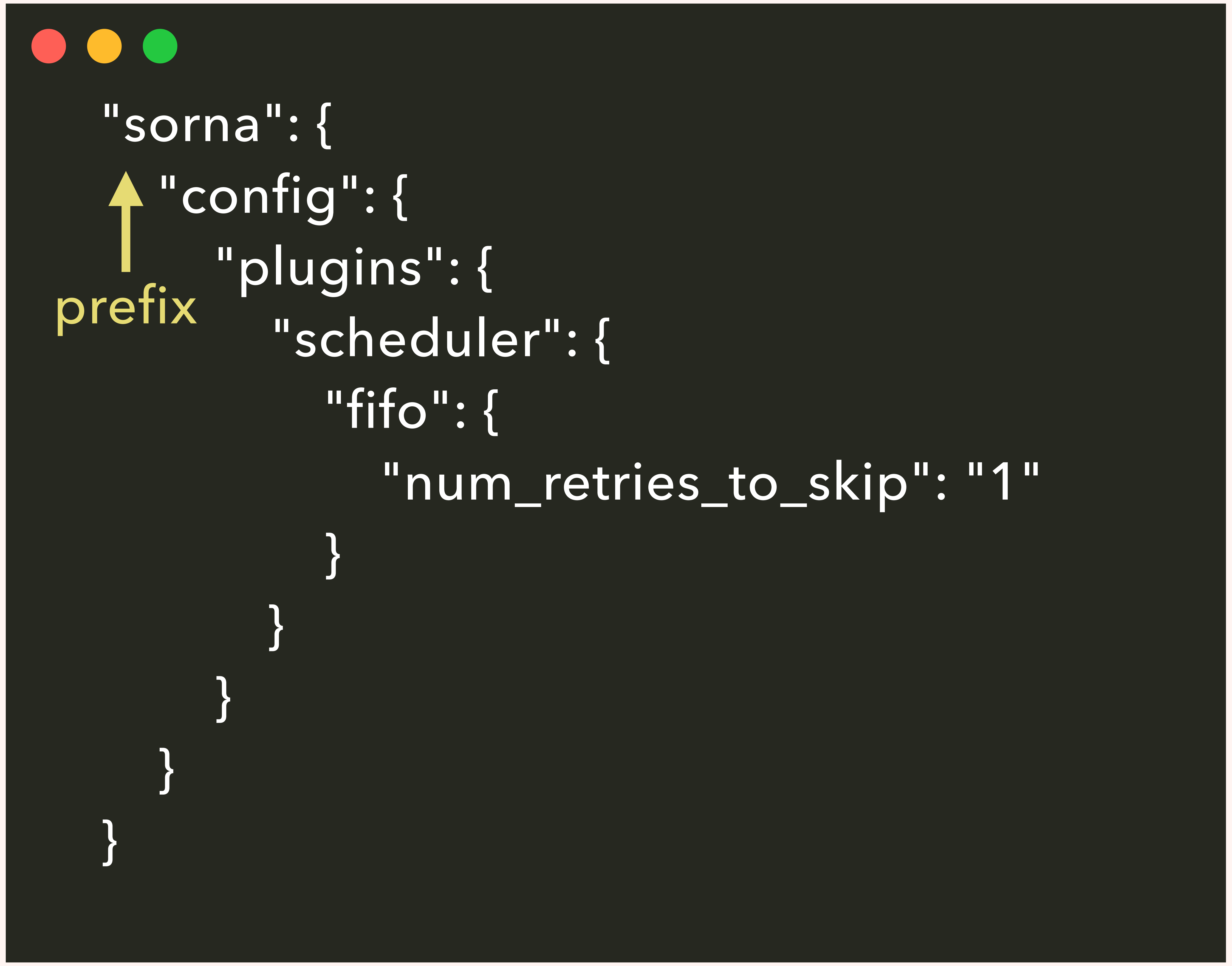
우리는 이 etcd를

“ prefix-based 필터링을 할 수 있는 키 - 값 저장소로 사용”

→ 폴더와 같은 구조로 환경 설정



```
globalThis.backendaiclient.setting.set(
  'plugins/scheduler/fifo/num_retries_to_skip', 1
);
```



```
"sorna": {
  "config": {
    "plugins": {
      "scheduler": {
        "fifo": {
          "num_retries_to_skip": "1"
        }
      }
    }
  }
}
```

CONFIGURATION SCHEMA ON ETCD

etcd의 환경설정 스키마

Global

↑

Scaling group ↑ 존재할 경우 override

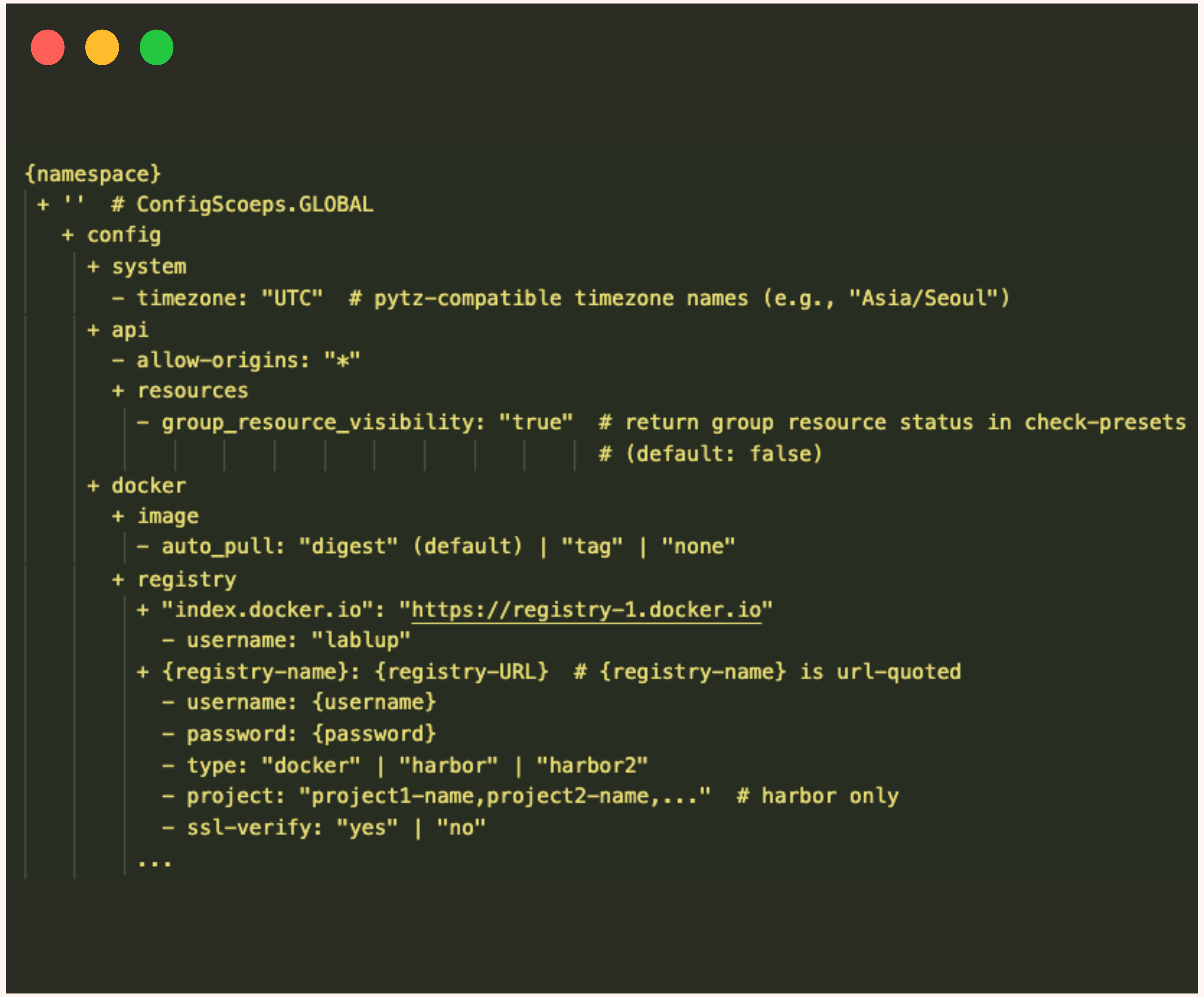
↑

Node

etcd의 환경설정 스키마

/manager/config.py

Global

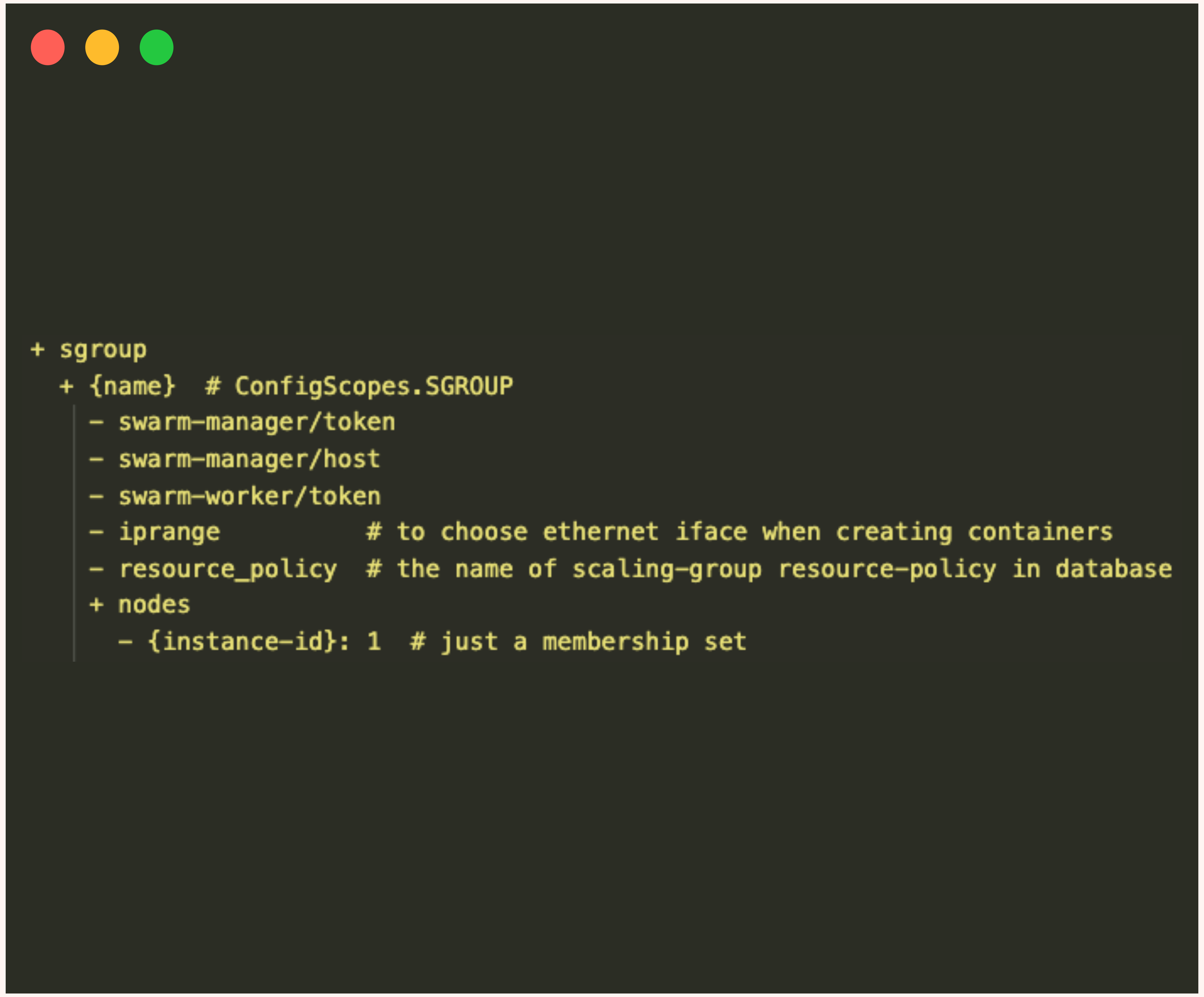


```
{namespace}
 + ''  # ConfigScoeps.GLOBAL
   + config
     + system
       - timezone: "UTC"  # pytz-compatible timezone names (e.g., "Asia/Seoul")
     + api
       - allow-origins: "*"
       + resources
         - group_resource_visibility: "true"  # return group resource status in check-presets
                                              # (default: false)
     + docker
       + image
         - auto_pull: "digest" (default) | "tag" | "none"
       + registry
         + "index.docker.io": "https://registry-1.docker.io"
           - username: "lablup"
         + {registry-name}: {registry-URL}  # {registry-name} is url-quoted
           - username: {username}
           - password: {password}
           - type: "docker" | "harbor" | "harbor2"
           - project: "project1-name,project2-name,..."  # harbor only
           - ssl-verify: "yes" | "no"
         ...
```

etcd의 환경설정 스키마

/manager/config.py

Scaling Group

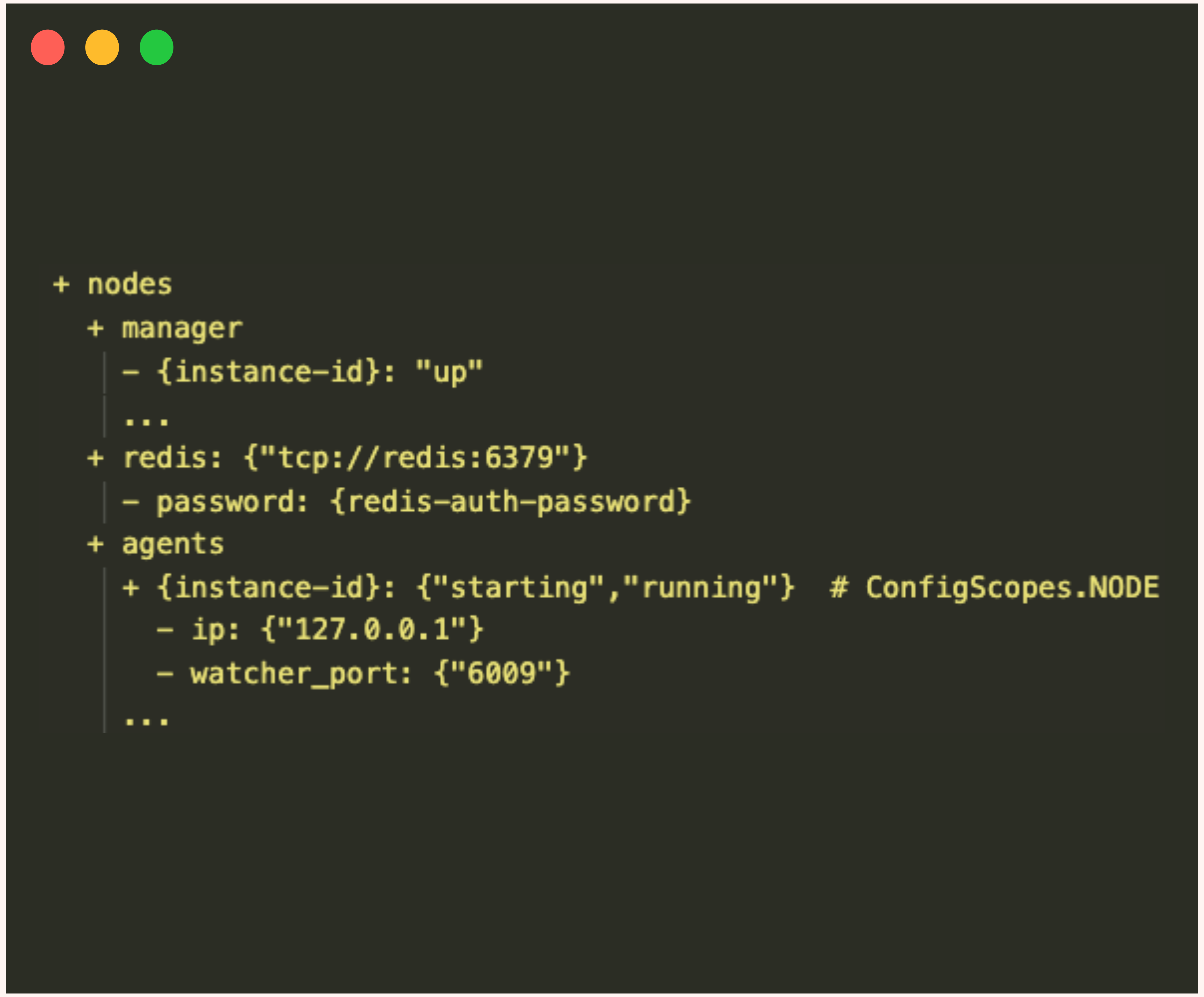


```
+ sgroup
  + {name}  # ConfigScopes.SGROUP
    - swarm-manager/token
    - swarm-manager/host
    - swarm-worker/token
    - iprange          # to choose ethernet iface when creating containers
    - resource_policy  # the name of scaling-group resource-policy in database
    + nodes
      - {instance-id}: 1  # just a membership set
```

etcd의 환경설정 스키마

/manager/config.py

- **Node**



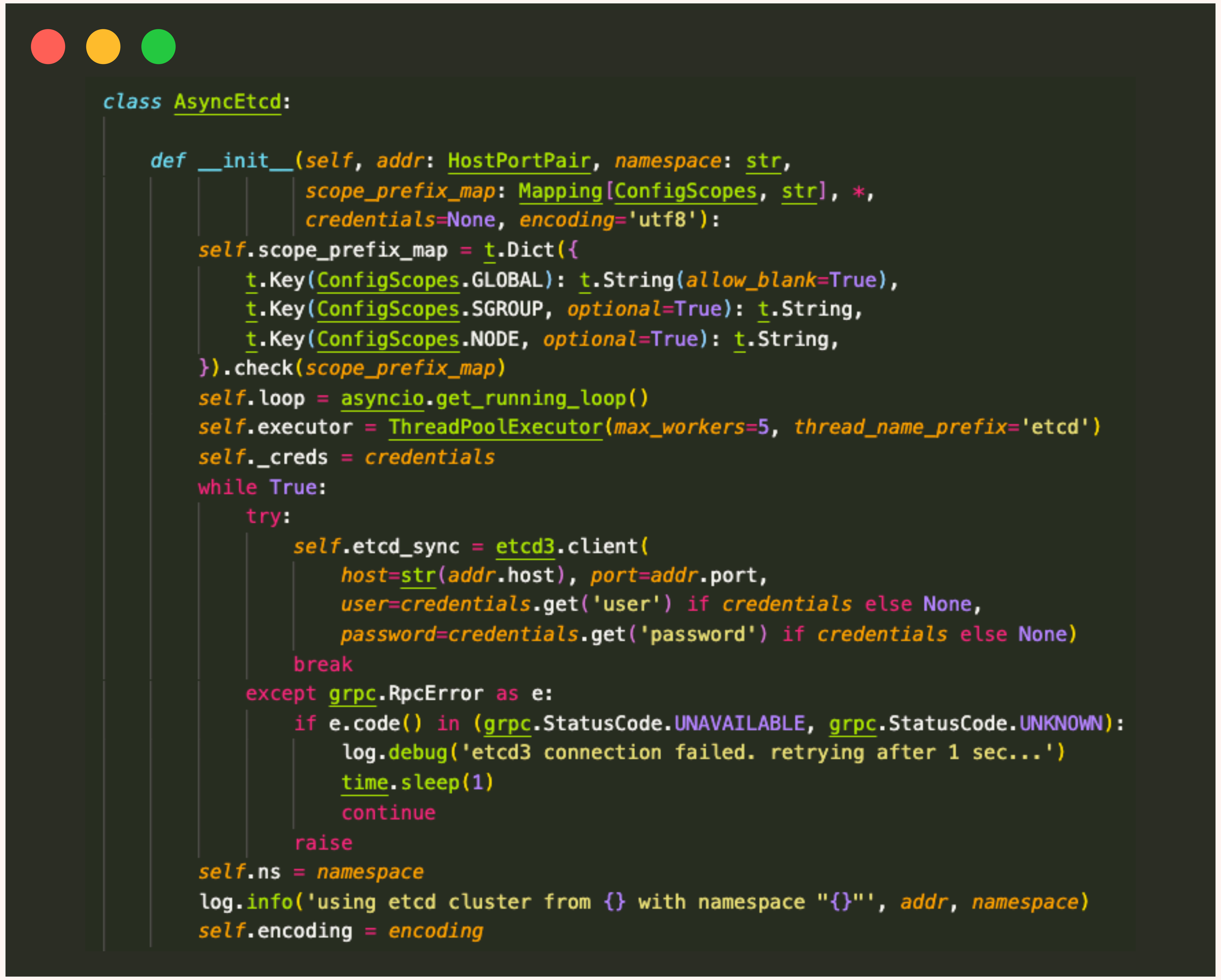
```
+ nodes
  + manager
  | - {instance-id}: "up"
  | ...
  + redis: {"tcp://redis:6379"}
  | - password: {redis-auth-password}
  + agents
  | + {instance-id}: {"starting","running"}  # ConfigScopes.NODE
  |   - ip: {"127.0.0.1"}
  |   - watcher_port: {"6009"}
  | ...
```

HOW DOES IT WORKS?

초기화

/common/etcd.py

- AsyncEtcd class 의 생성자

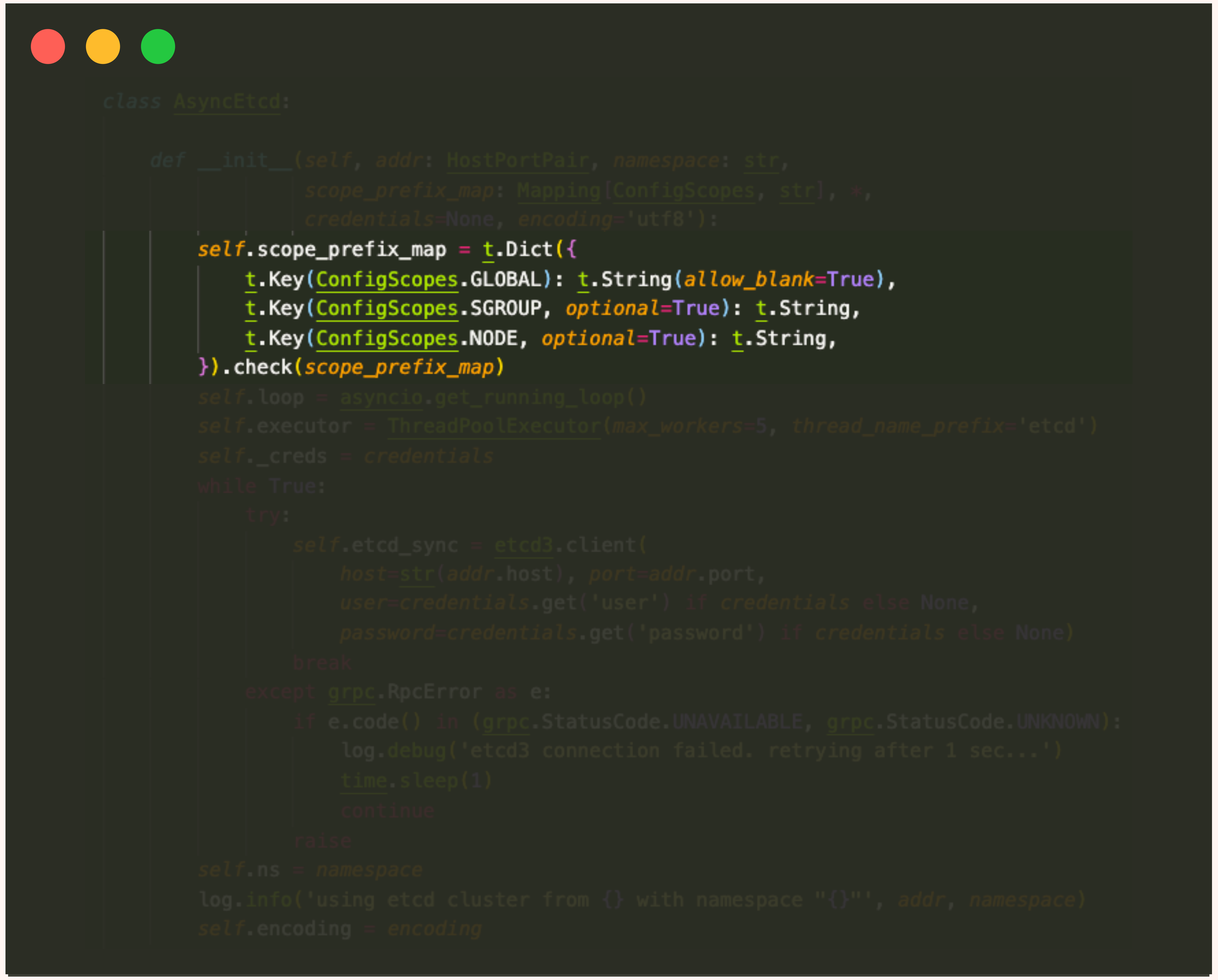


```python
class AsyncEtcd:

    def __init__(self, addr: HostPortPair, namespace: str,
                 scope_prefix_map: Mapping[ConfigScopes, str], *,
                 credentials=None, encoding='utf8'):
        self.scope_prefix_map = t.Dict({
            t.Key(ConfigScopes.GLOBAL): t.String(allow_blank=True),
            t.Key(ConfigScopes.SGROUP, optional=True): t.String,
            t.Key(ConfigScopes.NODE, optional=True): t.String,
        }).check(scope_prefix_map)
        self.loop = asyncio.get_running_loop()
        self.executor = ThreadPoolExecutor(max_workers=5, thread_name_prefix='etcd')
        self._creds = credentials
        while True:
            try:
                self.etcd_sync = etcd3.client(
                    host=str(addr.host), port=addr.port,
                    user=credentials.get('user') if credentials else None,
                    password=credentials.get('password') if credentials else None)
                break
            except grpc.RpcError as e:
                if e.code() in (grpc.StatusCode.UNAVAILABLE, grpc.StatusCode.UNKNOWN):
                    log.debug('etcd3 connection failed. retrying after 1 sec...')
                    time.sleep(1)
                    continue
                raise
        self.ns = namespace
        log.info('using etcd cluster from {} with namespace "{}"', addr, namespace)
        self.encoding = encoding
```

Initialize

/common/etcd.py

- AsyncEtcd class 의 생성자
  - scope_prefix_map
    - prefix의 scope 저장
    - trafaret을 사용해 딕셔너리 타입 검사



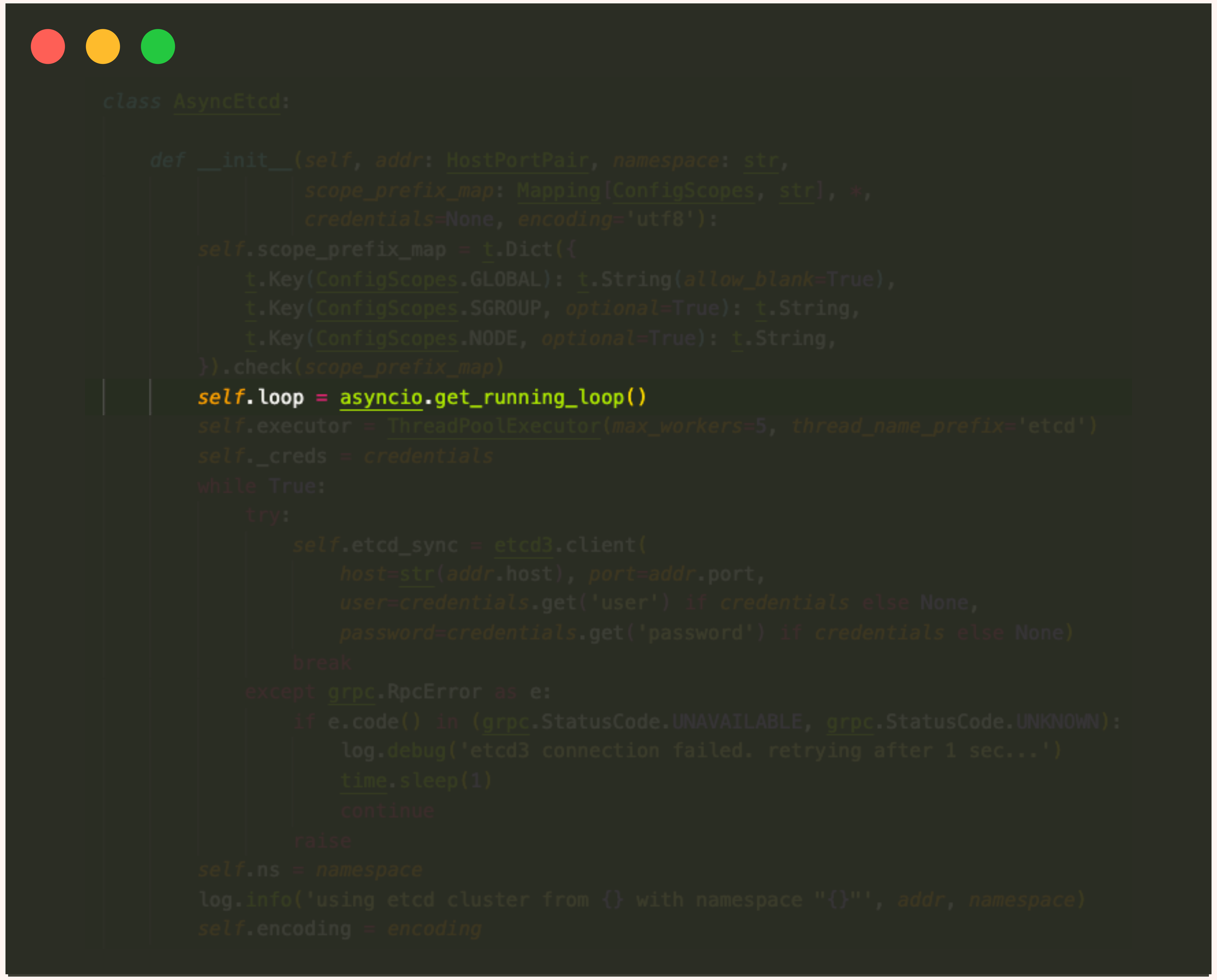
```python
class AsyncEtcd:

    def __init__(self, addr: HostPortPair, namespace: str,
                 scope_prefix_map: Mapping[ConfigScopes, str], *,
                 credentials=None, encoding='utf8'):
        self.scope_prefix_map = t.Dict({
            t.Key(ConfigScopes.GLOBAL): t.String(allow_blank=True),
            t.Key(ConfigScopes.SGROUP, optional=True): t.String,
            t.Key(ConfigScopes.NODE, optional=True): t.String,
        }).check(scope_prefix_map)
        self.loop = asyncio.get_running_loop()
        self.executor = ThreadPoolExecutor(max_workers=5, thread_name_prefix='etcd')
        self._creds = credentials
        while True:
            try:
                self.etcd_sync = etcd3.client(
                    host=str(addr.host), port=addr.port,
                    user=credentials.get('user') if credentials else None,
                    password=credentials.get('password') if credentials else None)
                break
            except grpc.RpcError as e:
                if e.code() in (grpc.StatusCode.UNAVAILABLE, grpc.StatusCode.UNKNOWN):
                    log.debug('etcd3 connection failed. retrying after 1 sec...')
                    time.sleep(1)
                    continue
                raise
        self.ns = namespace
        log.info('using etcd cluster from {} with namespace "{}"', addr, namespace)
        self.encoding = encoding
```

Initialize

/common/etcd.py

- **AsyncEtcd class 의 생성자**
  - **loop**

    : 현재 OS 스레드에서 작동중인 이벤트 루프 반환



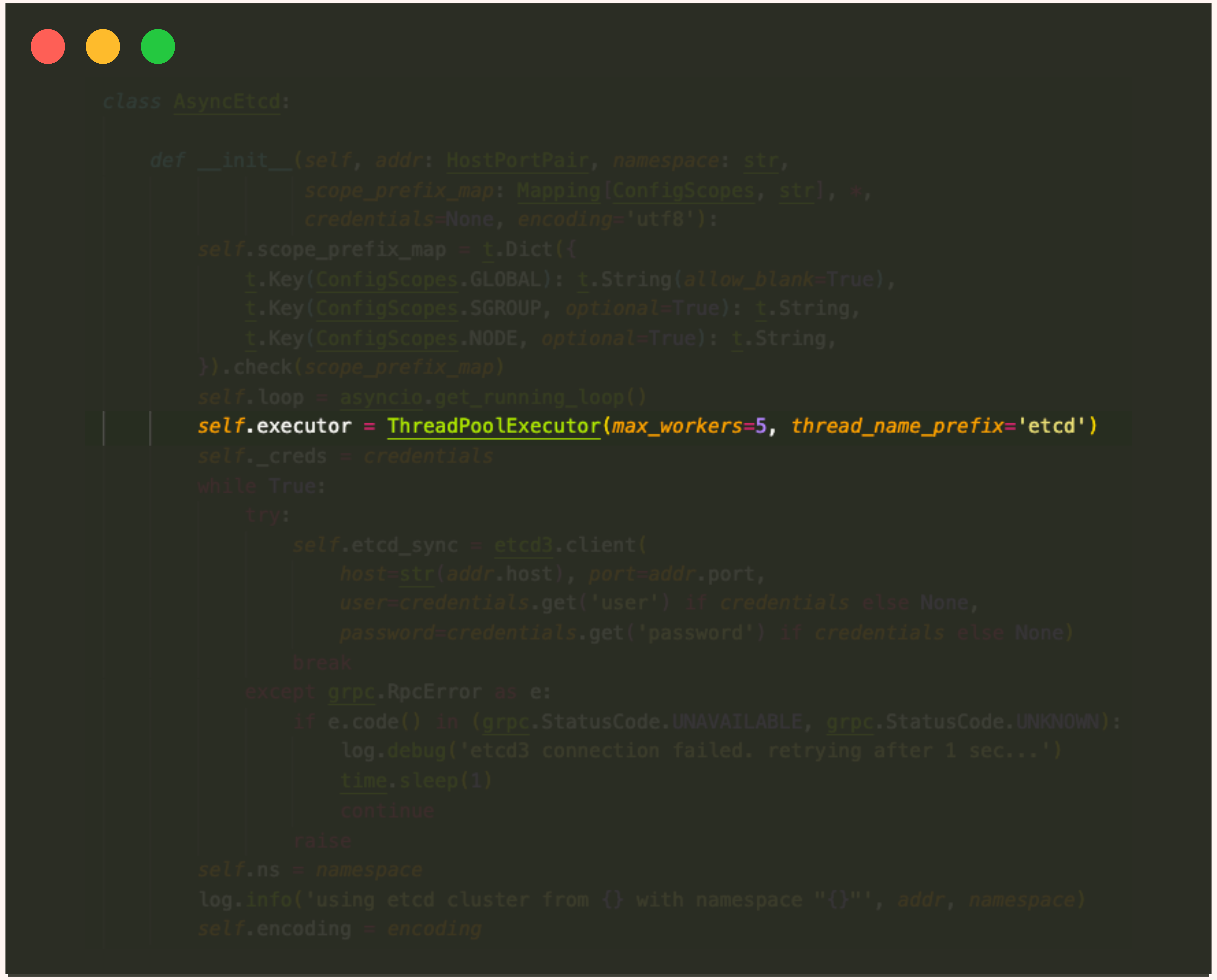
```python
class AsyncEtcd:

    def __init__(self, addr: HostPortPair, namespace: str,
                 scope_prefix_map: Mapping[ConfigScopes, str], *,
                 credentials=None, encoding='utf8'):
        self.scope_prefix_map = t.Dict({
            t.Key(ConfigScopes.GLOBAL): t.String(allow_blank=True),
            t.Key(ConfigScopes.SGROUP, optional=True): t.String,
            t.Key(ConfigScopes.NODE, optional=True): t.String,
        }).check(scope_prefix_map)
        self.loop = asyncio.get_running_loop()
        self.executor = ThreadPoolExecutor(max_workers=5, thread_name_prefix='etcd')
        self._creds = credentials
        while True:
            try:
                self.etcd_sync = etcd3.client(
                    host=str(addr.host), port=addr.port,
                    user=credentials.get('user') if credentials else None,
                    password=credentials.get('password') if credentials else None)
                break
            except grpc.RpcError as e:
                if e.code() in (grpc.StatusCode.UNAVAILABLE, grpc.StatusCode.UNKNOWN):
                    log.debug('etcd3 connection failed. retrying after 1 sec...')
                    time.sleep(1)
                    continue
                raise
        self.ns = namespace
        log.info('using etcd cluster from {} with namespace "{}"', addr, namespace)
        self.encoding = encoding
```

Initialize

/common/etcd.py

- AsyncEtcd class 의 생성자
  - executor

    : ThreadPoolExecutor 은 pool of threads를 사용하여 비동기적으로 호출을 시행하는 Executor의 하위 클래스

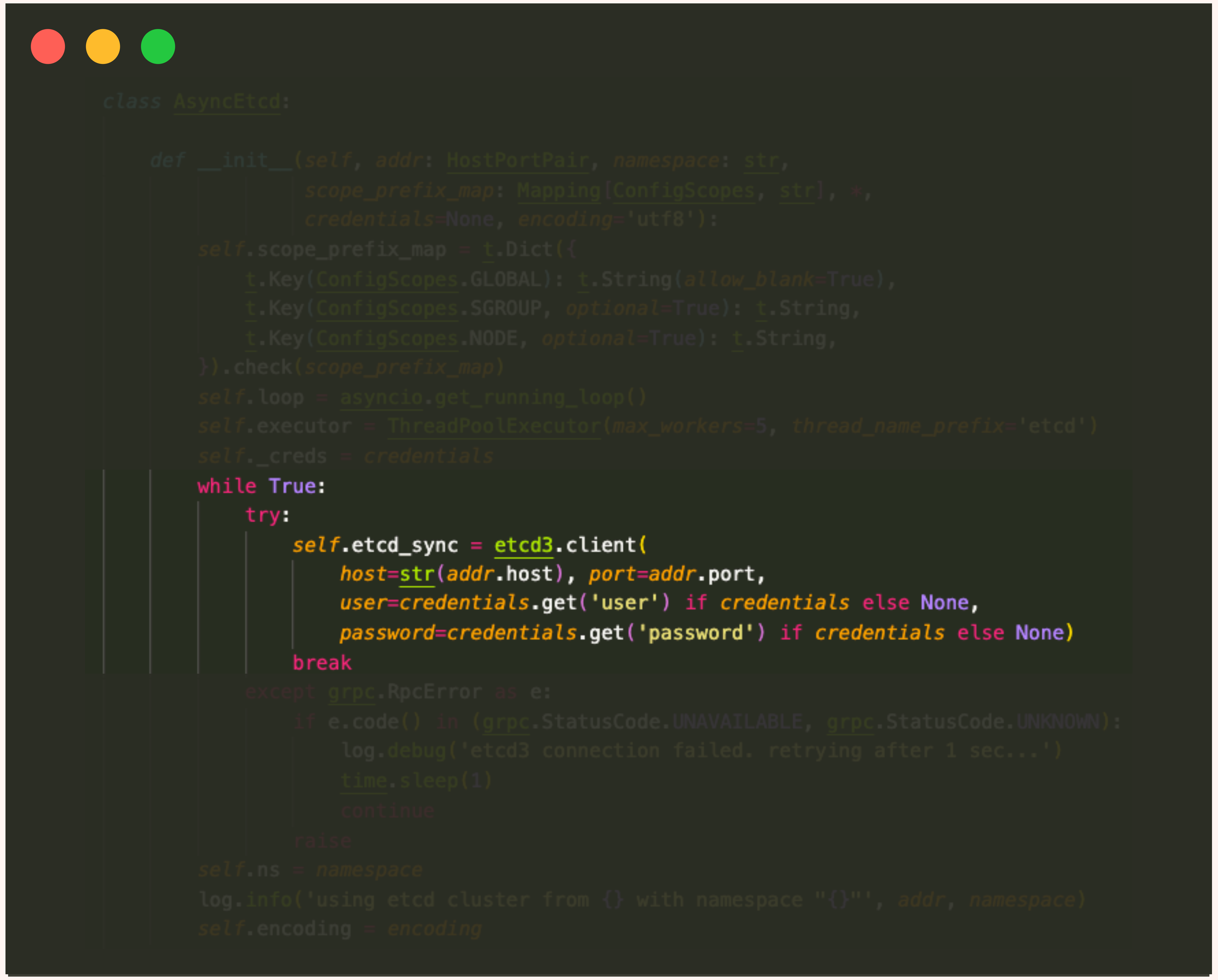


```python
class AsyncEtcd:

    def __init__(self, addr: HostPortPair, namespace: str,
                 scope_prefix_map: Mapping[ConfigScopes, str], *,
                 credentials=None, encoding='utf8'):
        self.scope_prefix_map = t.Dict({
            t.Key(ConfigScopes.GLOBAL): t.String(allow_blank=True),
            t.Key(ConfigScopes.SGROUP, optional=True): t.String,
            t.Key(ConfigScopes.NODE, optional=True): t.String,
        }).check(scope_prefix_map)
        self.loop = asyncio.get_running_loop()
        self.executor = ThreadPoolExecutor(max_workers=5, thread_name_prefix='etcd')
        self._creds = credentials
        while True:
            try:
                self.etcd_sync = etcd3.client(
                    host=str(addr.host), port=addr.port,
                    user=credentials.get('user') if credentials else None,
                    password=credentials.get('password') if credentials else None)
                break
            except grpc.RpcError as e:
                if e.code() in (grpc.StatusCode.UNAVAILABLE, grpc.StatusCode.UNKNOWN):
                    log.debug('etcd3 connection failed. retrying after 1 sec...')
                    time.sleep(1)
                    continue
                raise
        self.ns = namespace
        log.info('using etcd cluster from {} with namespace "{}"', addr, namespace)
        self.encoding = encoding
```

Initialize

/common/etcd.py

- AsyncEtcd 의 생성자
  - 특정 옵션을 가진 etcd3 client 생성



```python
class AsyncEtcd:

    def __init__(self, addr: HostPortPair, namespace: str,
                 scope_prefix_map: Mapping[ConfigScopes, str], *,
                 credentials=None, encoding='utf8'):
        self.scope_prefix_map = t.Dict({
            t.Key(ConfigScopes.GLOBAL): t.String(allow_blank=True),
            t.Key(ConfigScopes.SGROUP, optional=True): t.String,
            t.Key(ConfigScopes.NODE, optional=True): t.String,
        }).check(scope_prefix_map)
        self.loop = asyncio.get_running_loop()
        self.executor = ThreadPoolExecutor(max_workers=5, thread_name_prefix='etcd')
        self._creds = credentials
        while True:
            try:
                self.etcd_sync = etcd3.client(
                    host=str(addr.host), port=addr.port,
                    user=credentials.get('user') if credentials else None,
                    password=credentials.get('password') if credentials else None)
                break
            except grpc.RpcError as e:
                if e.code() in (grpc.StatusCode.UNAVAILABLE, grpc.StatusCode.UNKNOWN):
                    log.debug('etcd3 connection failed. retrying after 1 sec...')
                    time.sleep(1)
                    continue
                raise
        self.ns = namespace
        log.info('using etcd cluster from {} with namespace "{}"', addr, namespace)
        self.encoding = encoding
```

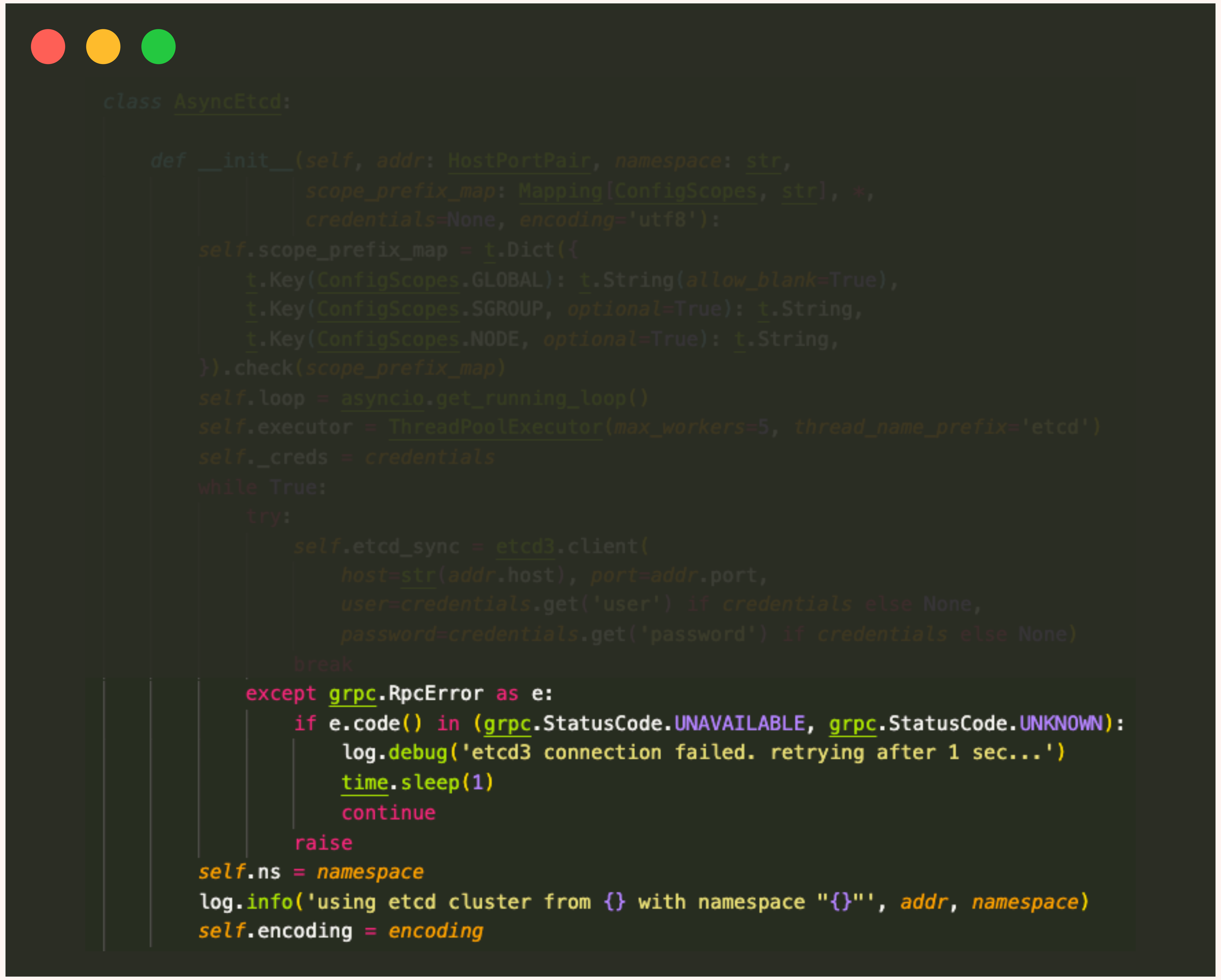
Initialize

/common/etcd.py

- AsyncEtcd class 의 생성자
  - RpcError 의 발생은 etcd3 연결이 실패함을 의미

(*RpcError: non-OK-status RPC 종료를 알려주는gRPC 라이브러리에 의해 발생)

  - 에러 발생 시 1초 뒤 재연결 시도

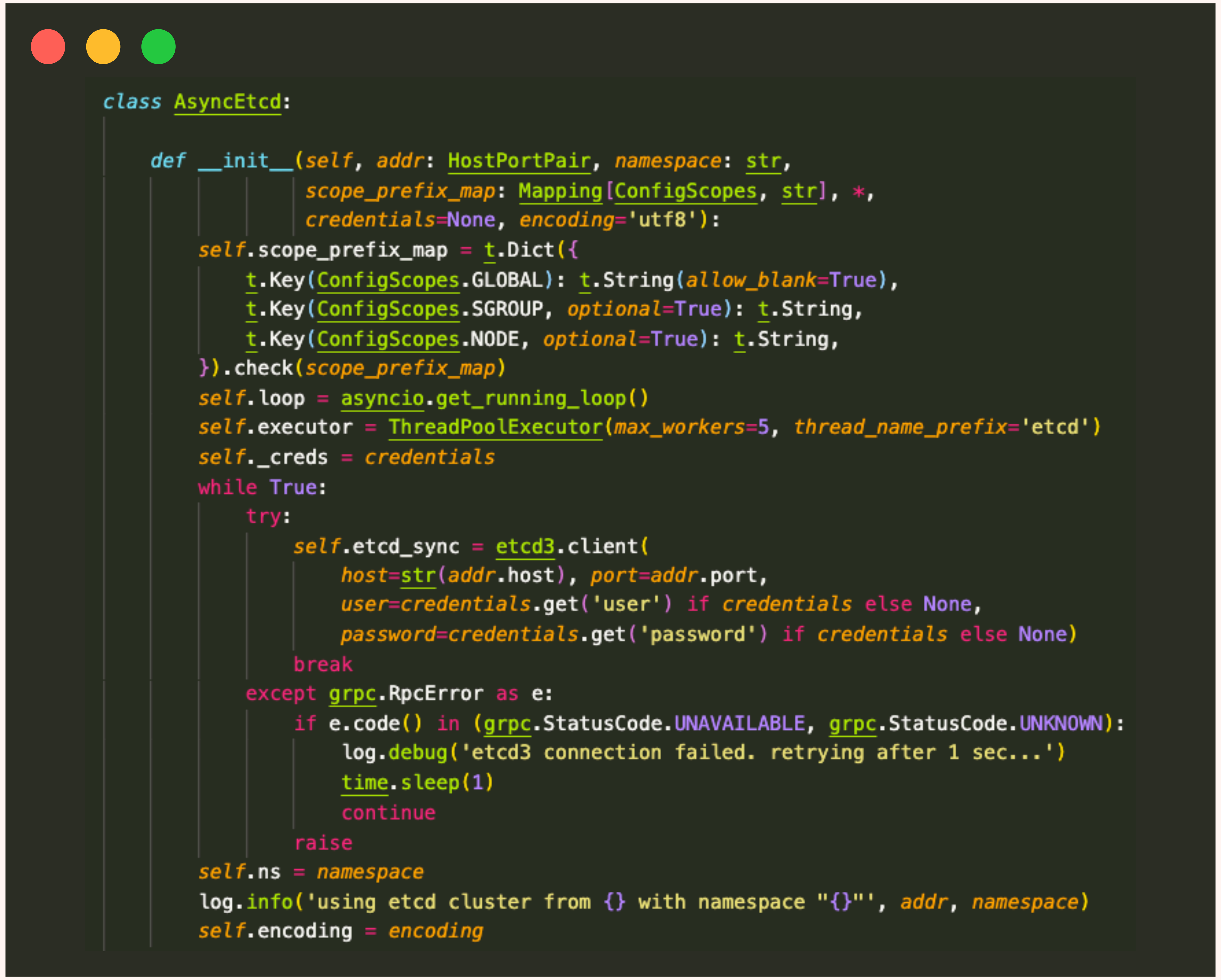


```
class AsyncEtcd:

    def __init__(self, addr: HostPortPair, namespace: str,
                 scope_prefix_map: Mapping[ConfigScopes, str], *,
                 credentials=None, encoding='utf8'):
        self.scope_prefix_map = t.Dict({
            t.Key(ConfigScopes.GLOBAL): t.String(allow_blank=True),
            t.Key(ConfigScopes.SGROUP, optional=True): t.String,
            t.Key(ConfigScopes.NODE, optional=True): t.String,
        }).check(scope_prefix_map)
        self.loop = asyncio.get_running_loop()
        self.executor = ThreadPoolExecutor(max_workers=5, thread_name_prefix='etcd')
        self._creds = credentials
        while True:
            try:
                self.etcd_sync = etcd3.client(
                    host=str(addr.host), port=addr.port,
                    user=credentials.get('user') if credentials else None,
                    password=credentials.get('password') if credentials else None)
                break
            except grpc.RpcError as e:
                if e.code() in (grpc.StatusCode.UNAVAILABLE, grpc.StatusCode.UNKNOWN):
                    log.debug('etcd3 connection failed. retrying after 1 sec...')
                    time.sleep(1)
                    continue
                raise
        self.ns = namespace
        log.info('using etcd cluster from {} with namespace "{}"', addr, namespace)
        self.encoding = encoding
```

Initialize

/common/etcd.py

AsyncEtcd class 의 생성자



```python
class AsyncEtcd:

    def __init__(self, addr: HostPortPair, namespace: str,
                 scope_prefix_map: Mapping[ConfigScopes, str], *,
                 credentials=None, encoding='utf8'):
        self.scope_prefix_map = t.Dict({
            t.Key(ConfigScopes.GLOBAL): t.String(allow_blank=True),
            t.Key(ConfigScopes.SGROUP, optional=True): t.String,
            t.Key(ConfigScopes.NODE, optional=True): t.String,
        }).check(scope_prefix_map)
        self.loop = asyncio.get_running_loop()
        self.executor = ThreadPoolExecutor(max_workers=5, thread_name_prefix='etcd')
        self._creds = credentials
        while True:
            try:
                self.etcd_sync = etcd3.client(
                    host=str(addr.host), port=addr.port,
                    user=credentials.get('user') if credentials else None,
                    password=credentials.get('password') if credentials else None)
                break
            except grpc.RpcError as e:
                if e.code() in (grpc.StatusCode.UNAVAILABLE, grpc.StatusCode.UNKNOWN):
                    log.debug('etcd3 connection failed. retrying after 1 sec...')
                    time.sleep(1)
                    continue
                raise
        self.ns = namespace
        log.info('using etcd cluster from {} with namespace "{}"', addr, namespace)
        self.encoding = encoding
```

Initialize

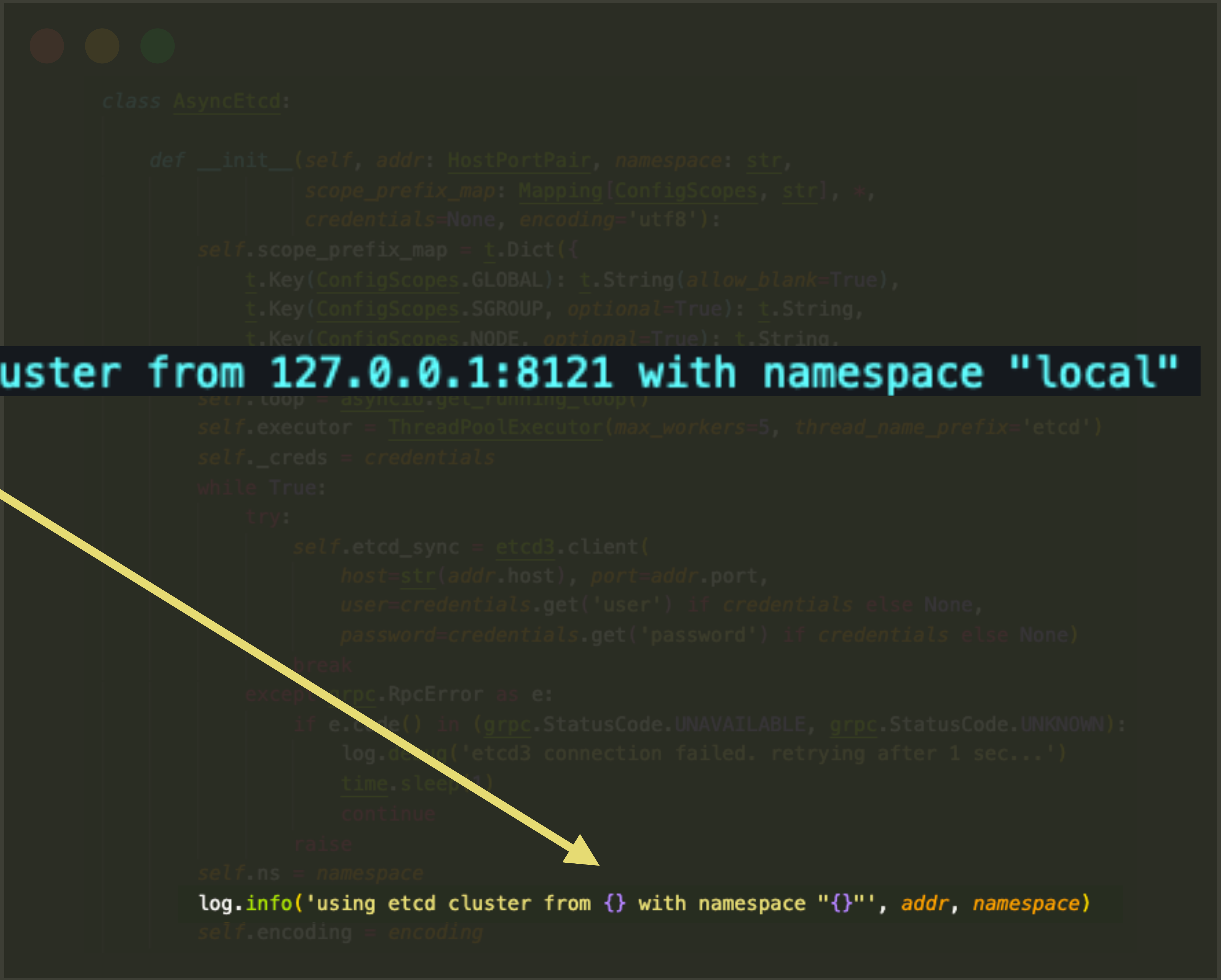
/common/etcd.py

- Constructor of AsyncEtcd class
  - The occurrence of RpcError [illegible] failed.

(*RpcError: Raised by the gRPC library to indicate non-OK-status RPC termination)

- So retrying to connect after 1 second.

```
INFO ai.backend.common.etcd [51062] using etcd cluster from 127.0.0.1:8121 with namespace "local"
```

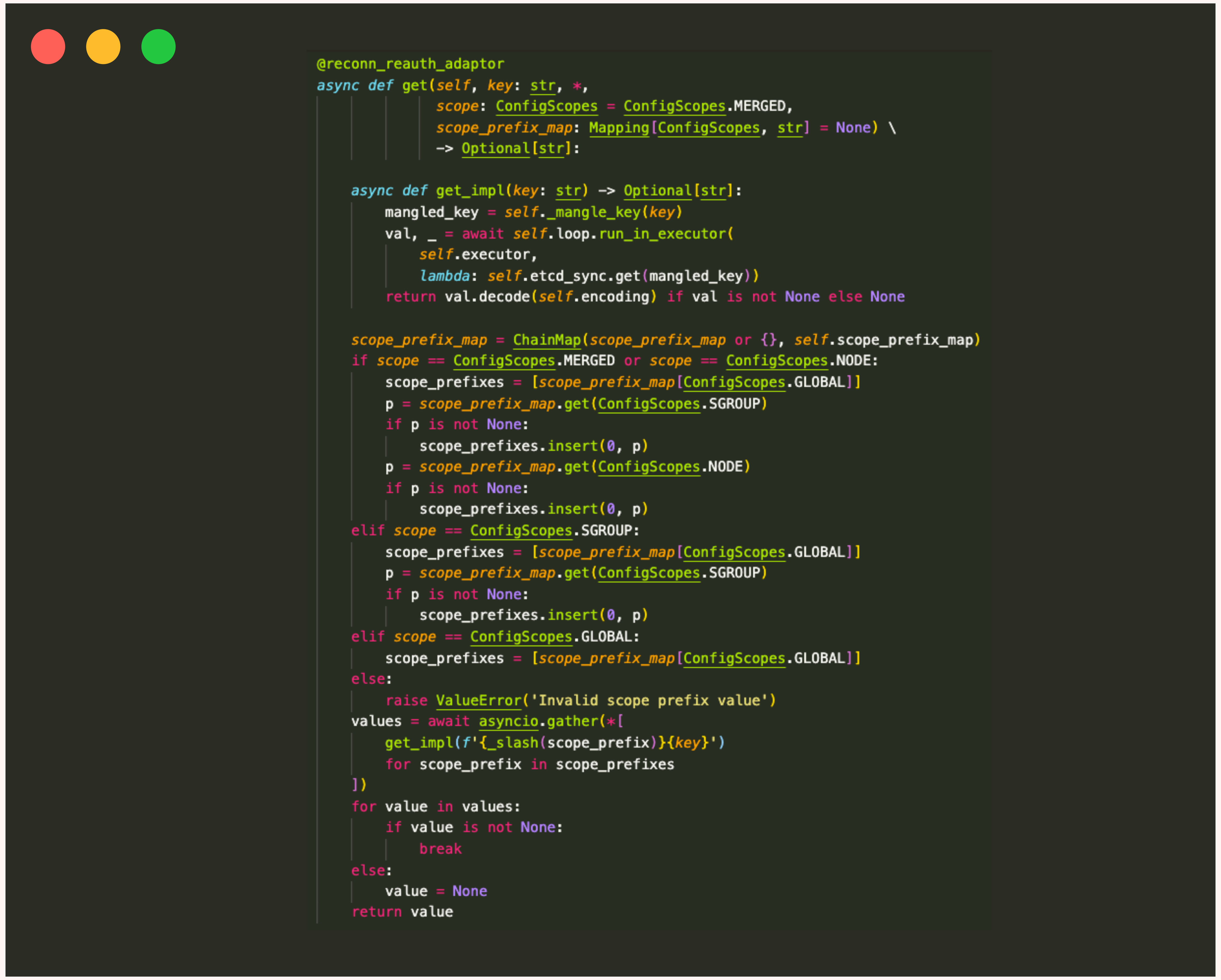


```python
class AsyncEtcd:

    def __init__(self, addr: HostPortPair, namespace: str,
                 scope_prefix_map: Mapping[ConfigScopes, str], *,
                 credentials=None, encoding='utf8'):
        self.scope_prefix_map = t.Dict({
            t.Key(ConfigScopes.GLOBAL): t.String(allow_blank=True),
            t.Key(ConfigScopes.SGROUP, optional=True): t.String,
            t.Key(ConfigScopes.NODE, optional=True): t.String,
        self.loop = asyncio.get_running_loop()
        self.executor = ThreadPoolExecutor(max_workers=5, thread_name_prefix='etcd')
        self._creds = credentials
        while True:
            try:
                self.etcd_sync = etcd3.client(
                    host=str(addr.host), port=addr.port,
                    user=credentials.get('user') if credentials else None,
                    password=credentials.get('password') if credentials else None)
                break
            except grpc.RpcError as e:
                if e.code() in (grpc.StatusCode.UNAVAILABLE, grpc.StatusCode.UNKNOWN):
                    log.debug('etcd3 connection failed. retrying after 1 sec...')
                    time.sleep(1)
                    continue
                raise
        self.ns = namespace
        log.info('using etcd cluster from {} with namespace "{}"', addr, namespace)
        self.encoding = encoding
```

get()

/common/etcd.py

- etcd 키로 값 불러오기



```python
@reconn_reauth_adaptor
async def get(self, key: str, *,
              scope: ConfigScopes = ConfigScopes.MERGED,
              scope_prefix_map: Mapping[ConfigScopes, str] = None) \
              -> Optional[str]:

    async def get_impl(key: str) -> Optional[str]:
        mangled_key = self._mangle_key(key)
        val, _ = await self.loop.run_in_executor(
            self.executor,
            lambda: self.etcd_sync.get(mangled_key))
        return val.decode(self.encoding) if val is not None else None

    scope_prefix_map = ChainMap(scope_prefix_map or {}, self.scope_prefix_map)
    if scope == ConfigScopes.MERGED or scope == ConfigScopes.NODE:
        scope_prefixes = [scope_prefix_map[ConfigScopes.GLOBAL]]
        p = scope_prefix_map.get(ConfigScopes.SGROUP)
        if p is not None:
            scope_prefixes.insert(0, p)
        p = scope_prefix_map.get(ConfigScopes.NODE)
        if p is not None:
            scope_prefixes.insert(0, p)
    elif scope == ConfigScopes.SGROUP:
        scope_prefixes = [scope_prefix_map[ConfigScopes.GLOBAL]]
        p = scope_prefix_map.get(ConfigScopes.SGROUP)
        if p is not None:
            scope_prefixes.insert(0, p)
    elif scope == ConfigScopes.GLOBAL:
        scope_prefixes = [scope_prefix_map[ConfigScopes.GLOBAL]]
    else:
        raise ValueError('Invalid scope prefix value')
    values = await asyncio.gather(*[
        get_impl(f'{_slash(scope_prefix)}{key}')
        for scope_prefix in scope_prefixes
    ])
    for value in values:
        if value is not None:
            break
    else:
        value = None
    return value
```

get()

/common/etcd.py

- etcd 키로 값 불러오기

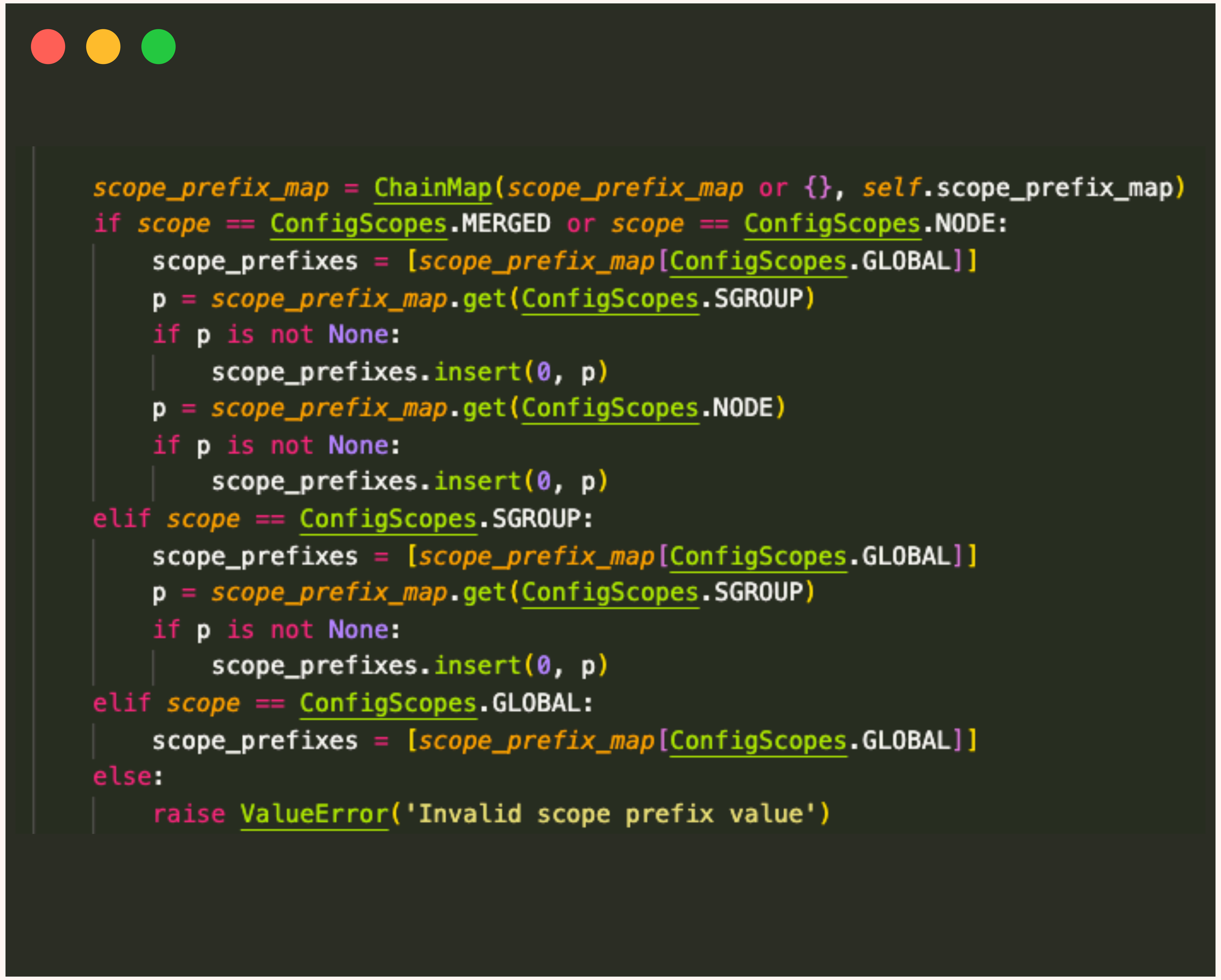
Global

↑

Scaling group

↑

Node

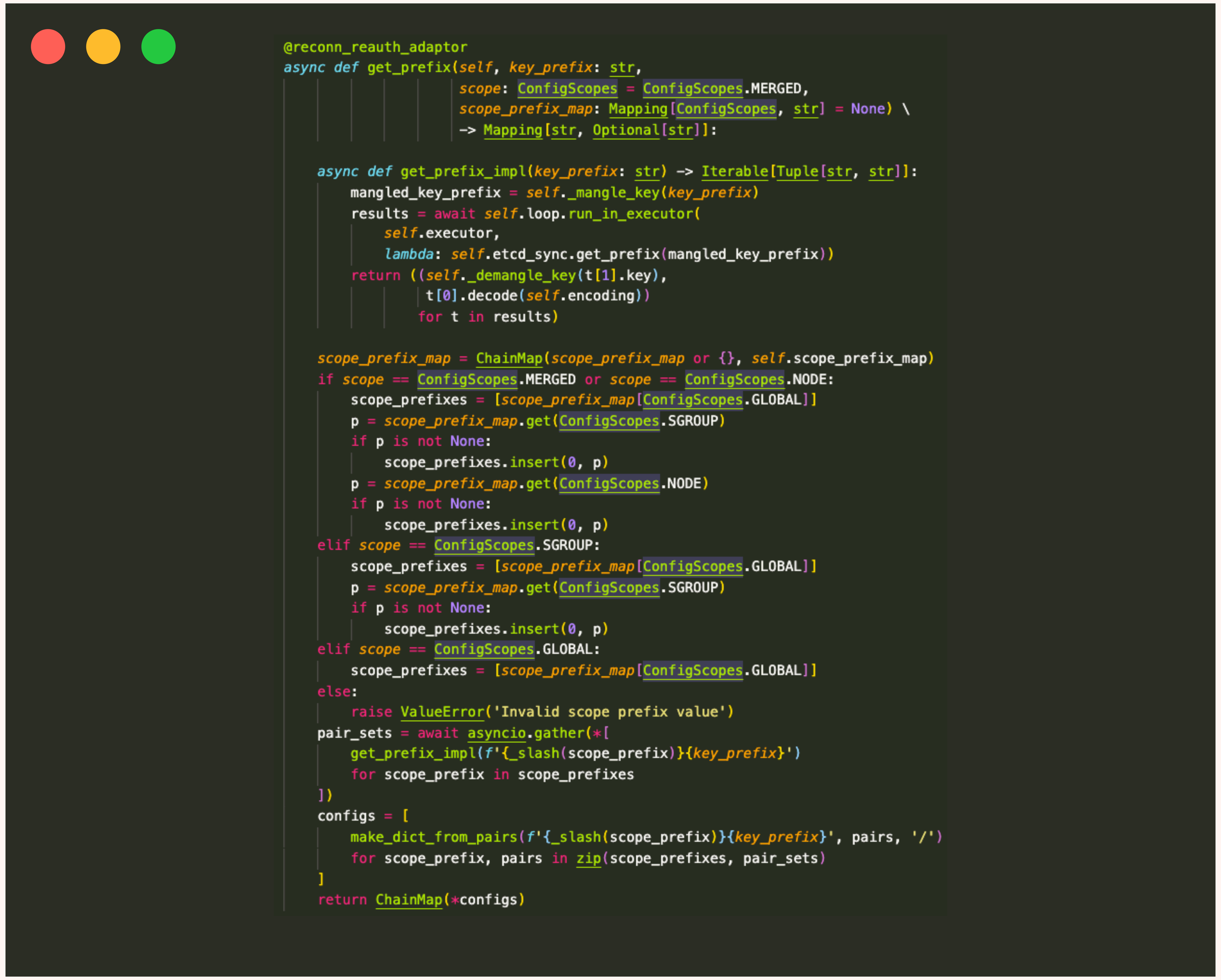


```
scope_prefix_map = ChainMap(scope_prefix_map or {}, self.scope_prefix_map)
if scope == ConfigScopes.MERGED or scope == ConfigScopes.NODE:
    scope_prefixes = [scope_prefix_map[ConfigScopes.GLOBAL]]
    p = scope_prefix_map.get(ConfigScopes.SGROUP)
    if p is not None:
        scope_prefixes.insert(0, p)
    p = scope_prefix_map.get(ConfigScopes.NODE)
    if p is not None:
        scope_prefixes.insert(0, p)
elif scope == ConfigScopes.SGROUP:
    scope_prefixes = [scope_prefix_map[ConfigScopes.GLOBAL]]
    p = scope_prefix_map.get(ConfigScopes.SGROUP)
    if p is not None:
        scope_prefixes.insert(0, p)
elif scope == ConfigScopes.GLOBAL:
    scope_prefixes = [scope_prefix_map[ConfigScopes.GLOBAL]]
else:
    raise ValueError('Invalid scope prefix value')
```

get_prefix()

/common/etcd.py

- 특정 prefix를 가진 키 값들 불러오기
  - `collections.ChainMap` instance 반환 (Global ← Scaling group ← Node)

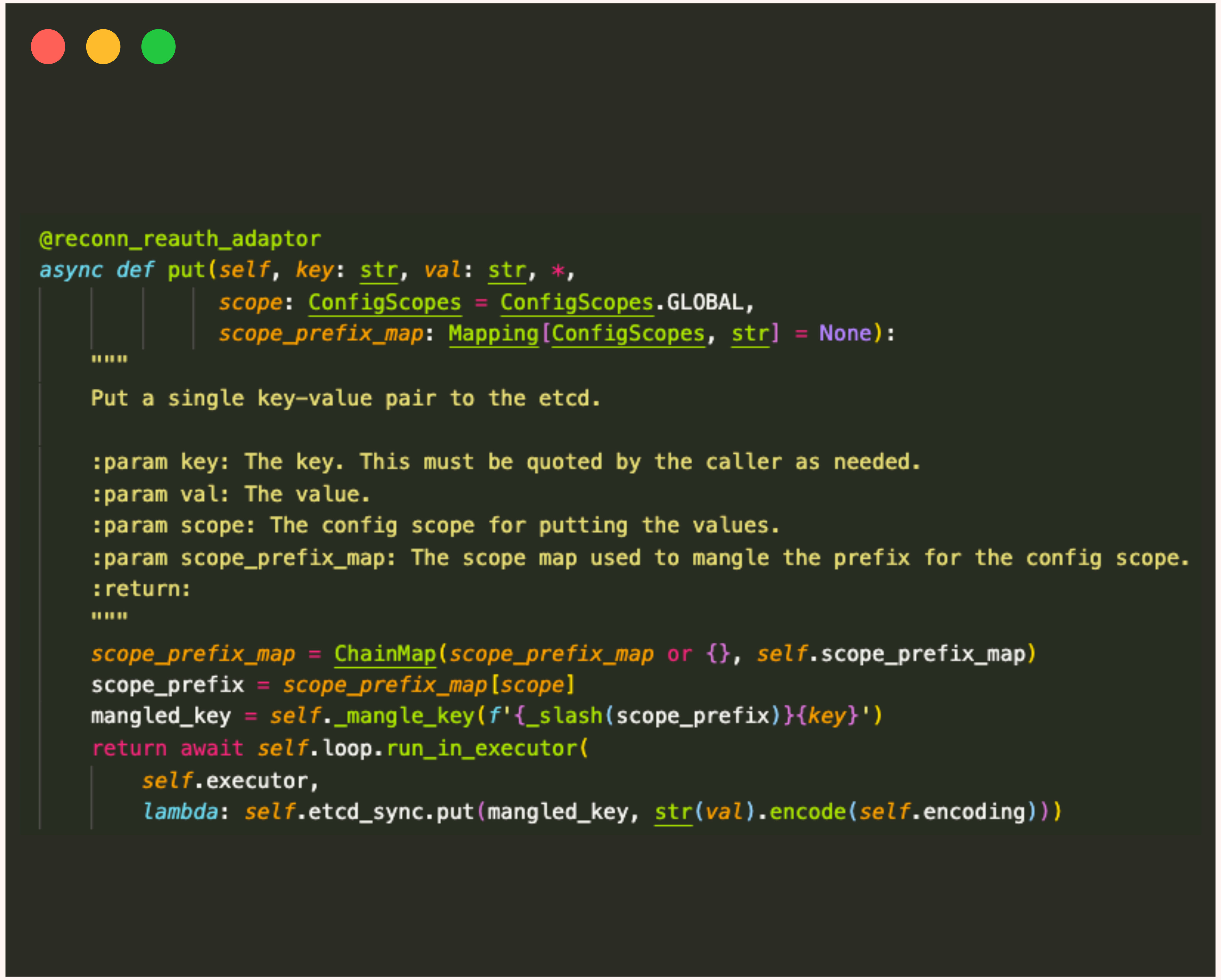


```python
@reconn_reauth_adaptor
async def get_prefix(self, key_prefix: str,
                     scope: ConfigScopes = ConfigScopes.MERGED,
                     scope_prefix_map: Mapping[ConfigScopes, str] = None) \
                     -> Mapping[str, Optional[str]]:

    async def get_prefix_impl(key_prefix: str) -> Iterable[Tuple[str, str]]:
        mangled_key_prefix = self._mangle_key(key_prefix)
        results = await self.loop.run_in_executor(
            self.executor,
            lambda: self.etcd_sync.get_prefix(mangled_key_prefix))
        return ((self._demangle_key(t[1].key),
                 t[0].decode(self.encoding))
                for t in results)

    scope_prefix_map = ChainMap(scope_prefix_map or {}, self.scope_prefix_map)
    if scope == ConfigScopes.MERGED or scope == ConfigScopes.NODE:
        scope_prefixes = [scope_prefix_map[ConfigScopes.GLOBAL]]
        p = scope_prefix_map.get(ConfigScopes.SGROUP)
        if p is not None:
            scope_prefixes.insert(0, p)
        p = scope_prefix_map.get(ConfigScopes.NODE)
        if p is not None:
            scope_prefixes.insert(0, p)
    elif scope == ConfigScopes.SGROUP:
        scope_prefixes = [scope_prefix_map[ConfigScopes.GLOBAL]]
        p = scope_prefix_map.get(ConfigScopes.SGROUP)
        if p is not None:
            scope_prefixes.insert(0, p)
    elif scope == ConfigScopes.GLOBAL:
        scope_prefixes = [scope_prefix_map[ConfigScopes.GLOBAL]]
    else:
        raise ValueError('Invalid scope prefix value')
    pair_sets = await asyncio.gather(*[
        get_prefix_impl(f'{_slash(scope_prefix)}{key_prefix}')
        for scope_prefix in scope_prefixes
    ])
    configs = [
        make_dict_from_pairs(f'{_slash(scope_prefix)}{key_prefix}', pairs, '/')
        for scope_prefix, pairs in zip(scope_prefixes, pair_sets)
    ]
    return ChainMap(*configs)
```

put()

/common/etcd.py

- etcd 에 값 저장하기

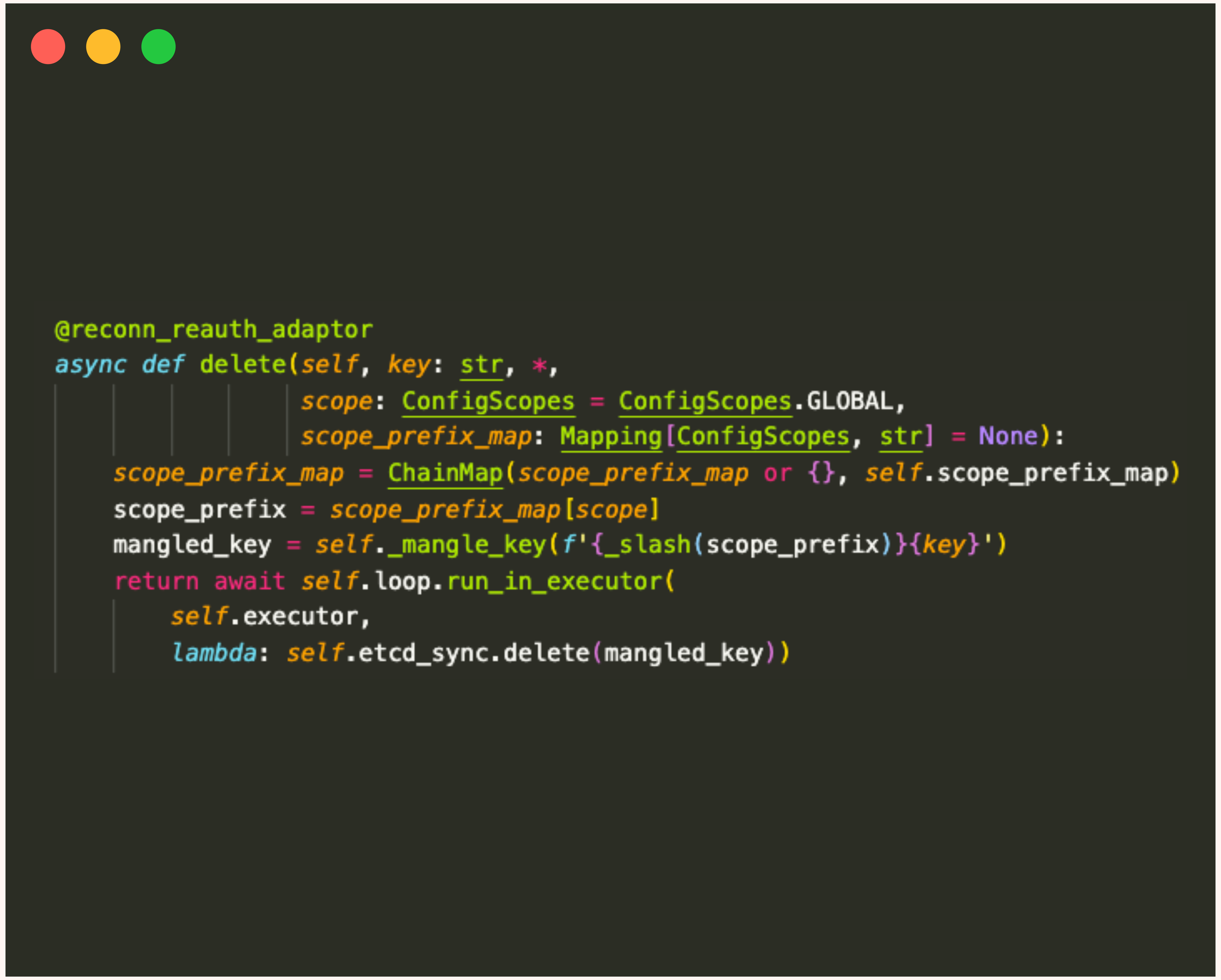


```python
@reconn_reauth_adaptor
async def put(self, key: str, val: str, *,
              scope: ConfigScopes = ConfigScopes.GLOBAL,
              scope_prefix_map: Mapping[ConfigScopes, str] = None):
    """
    Put a single key-value pair to the etcd.

    :param key: The key. This must be quoted by the caller as needed.
    :param val: The value.
    :param scope: The config scope for putting the values.
    :param scope_prefix_map: The scope map used to mangle the prefix for the config scope.
    :return:
    """
    scope_prefix_map = ChainMap(scope_prefix_map or {}, self.scope_prefix_map)
    scope_prefix = scope_prefix_map[scope]
    mangled_key = self._mangle_key(f'{_slash(scope_prefix)}{key}')
    return await self.loop.run_in_executor(
        self.executor,
        lambda: self.etcd_sync.put(mangled_key, str(val).encode(self.encoding)))
```

delete()

/common/etcd.py

- 키 삭제

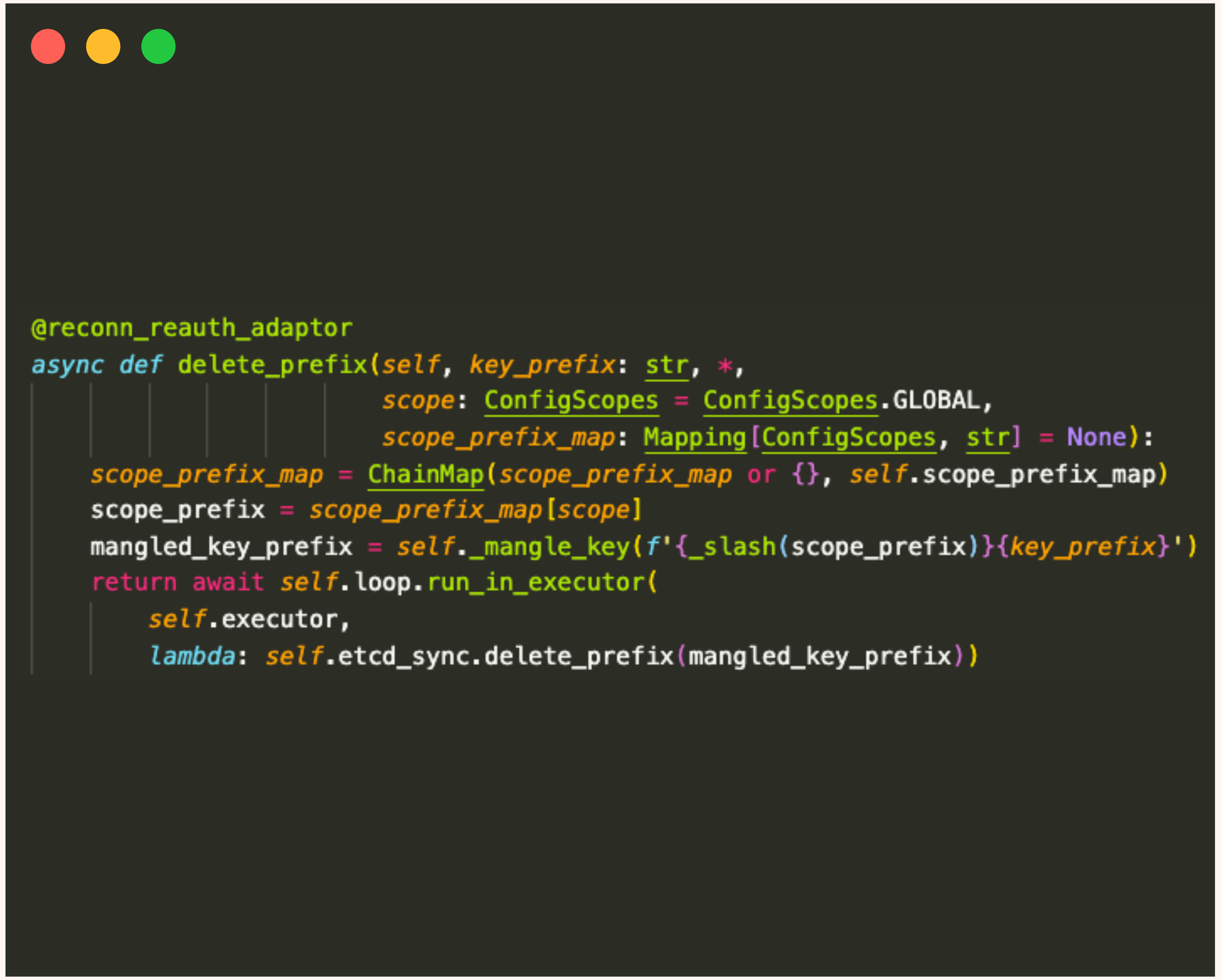


```python
@reconn_reauth_adaptor
async def delete(self, key: str, *,
                 scope: ConfigScopes = ConfigScopes.GLOBAL,
                 scope_prefix_map: Mapping[ConfigScopes, str] = None):
    scope_prefix_map = ChainMap(scope_prefix_map or {}, self.scope_prefix_map)
    scope_prefix = scope_prefix_map[scope]
    mangled_key = self._mangle_key(f'{_slash(scope_prefix)}{key}')
    return await self.loop.run_in_executor(
        self.executor,
        lambda: self.etcd_sync.delete(mangled_key))
```

delete_prefix()

/common/etcd.py

- 특정 prefix를 가진 키들 삭제

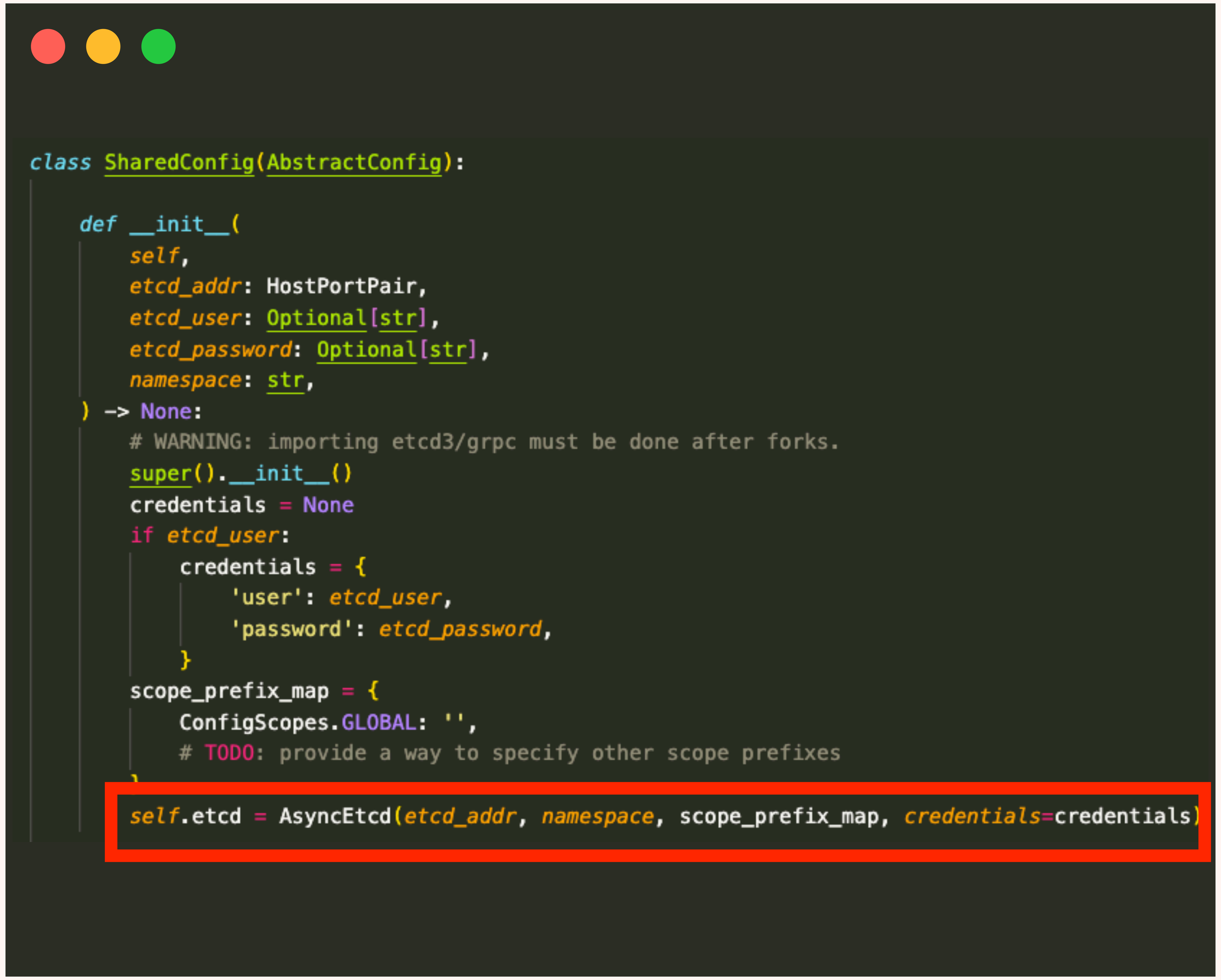


```python
@reconn_reauth_adaptor
async def delete_prefix(self, key_prefix: str, *,
                        scope: ConfigScopes = ConfigScopes.GLOBAL,
                        scope_prefix_map: Mapping[ConfigScopes, str] = None):
    scope_prefix_map = ChainMap(scope_prefix_map or {}, self.scope_prefix_map)
    scope_prefix = scope_prefix_map[scope]
    mangled_key_prefix = self._mangle_key(f'{_slash(scope_prefix)}{key_prefix}')
    return await self.loop.run_in_executor(
        self.executor,
        lambda: self.etcd_sync.delete_prefix(mangled_key_prefix))
```

SharedConfig

/manager/config.py

- SharedConfig 에 member 변수로 초기화

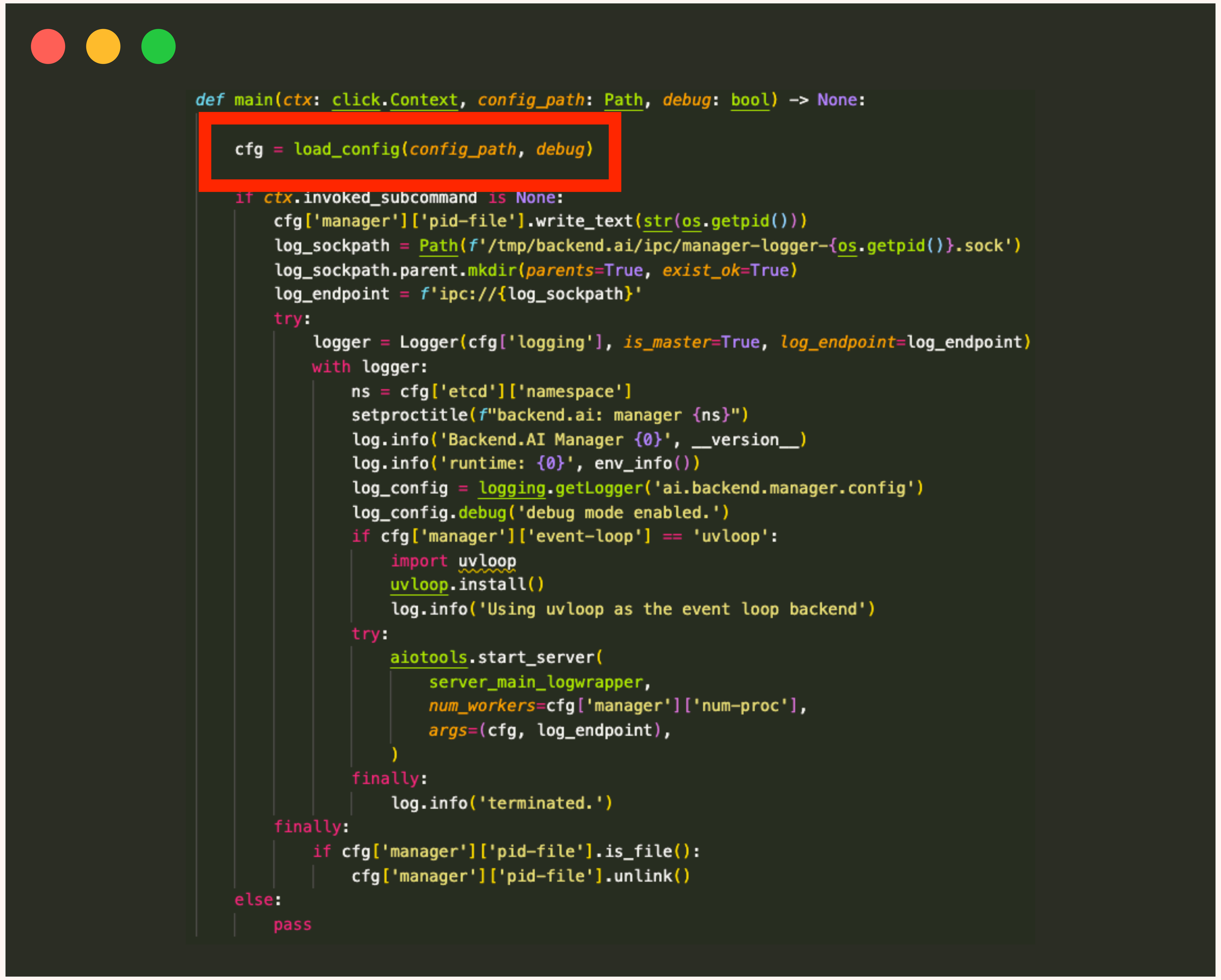


```python
class SharedConfig(AbstractConfig):

    def __init__(
        self,
        etcd_addr: HostPortPair,
        etcd_user: Optional[str],
        etcd_password: Optional[str],
        namespace: str,
    ) -> None:
        # WARNING: importing etcd3/grpc must be done after forks.
        super().__init__()
        credentials = None
        if etcd_user:
            credentials = {
                'user': etcd_user,
                'password': etcd_password,
            }
        scope_prefix_map = {
            ConfigScopes.GLOBAL: '',
            # TODO: provide a way to specify other scope prefixes
        }
        self.etcd = AsyncEtcd(etcd_addr, namespace, scope_prefix_map, credentials=credentials)
```

Start the manager service

/manager/server.py

- 매니저 서비스 시작
- etcd, redis, vfolder 등을 포함한 환경설정 로드



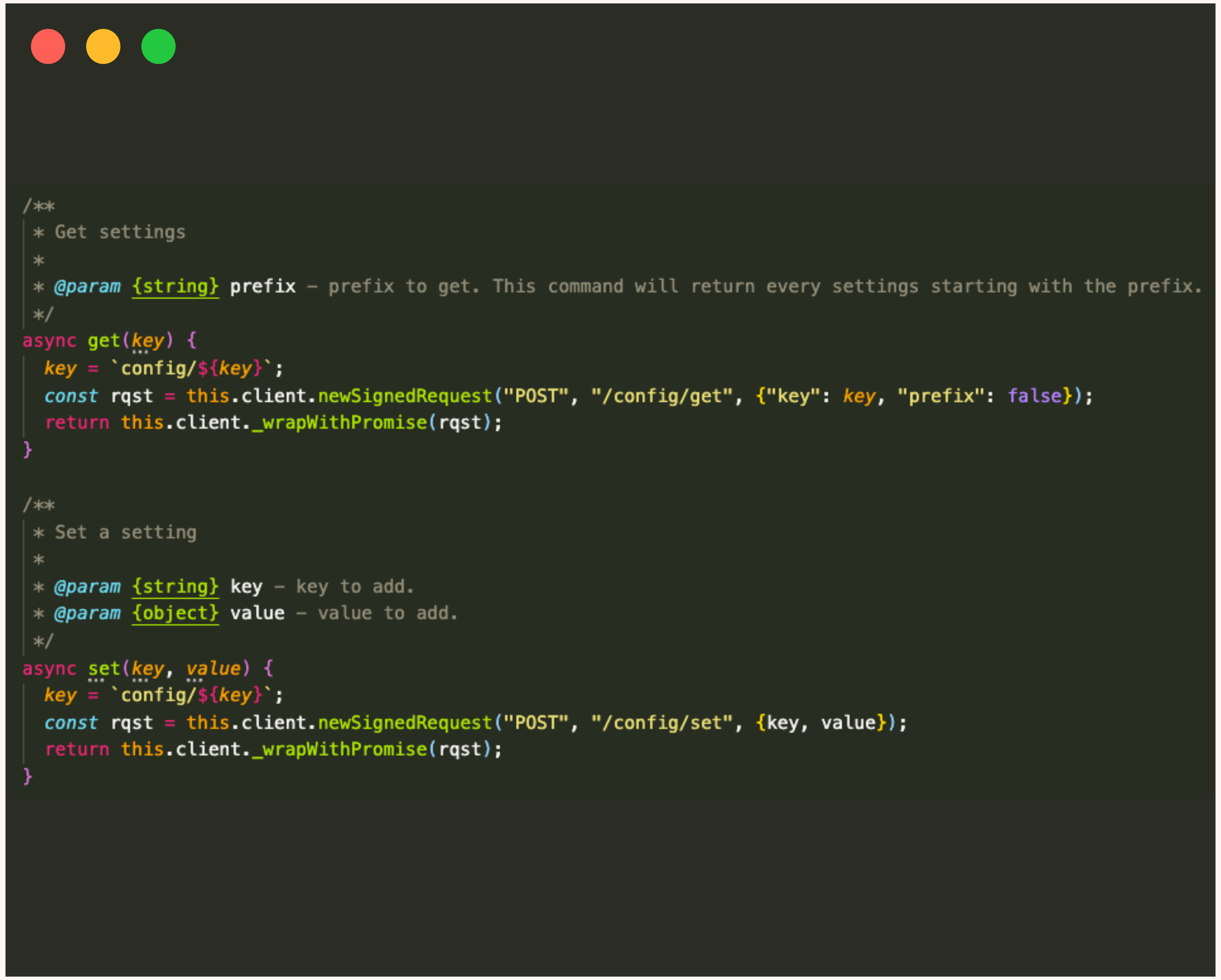
```python
def main(ctx: click.Context, config_path: Path, debug: bool) -> None:

    cfg = load_config(config_path, debug)

    if ctx.invoked_subcommand is None:
        cfg['manager']['pid-file'].write_text(str(os.getpid()))
        log_sockpath = Path(f'/tmp/backend.ai/ipc/manager-logger-{os.getpid()}.sock')
        log_sockpath.parent.mkdir(parents=True, exist_ok=True)
        log_endpoint = f'ipc://{log_sockpath}'
        try:
            logger = Logger(cfg['logging'], is_master=True, log_endpoint=log_endpoint)
            with logger:
                ns = cfg['etcd']['namespace']
                setproctitle(f"backend.ai: manager {ns}")
                log.info('Backend.AI Manager {0}', __version__)
                log.info('runtime: {0}', env_info())
                log_config = logging.getLogger('ai.backend.manager.config')
                log_config.debug('debug mode enabled.')
                if cfg['manager']['event-loop'] == 'uvloop':
                    import uvloop
                    uvloop.install()
                    log.info('Using uvloop as the event loop backend')
                try:
                    aiotools.start_server(
                        server_main_logwrapper,
                        num_workers=cfg['manager']['num-proc'],
                        args=(cfg, log_endpoint),
                    )
                finally:
                    log.info('terminated.')
        finally:
            if cfg['manager']['pid-file'].is_file():
                cfg['manager']['pid-file'].unlink()
    else:
        pass
```

Example of request to manager in webui

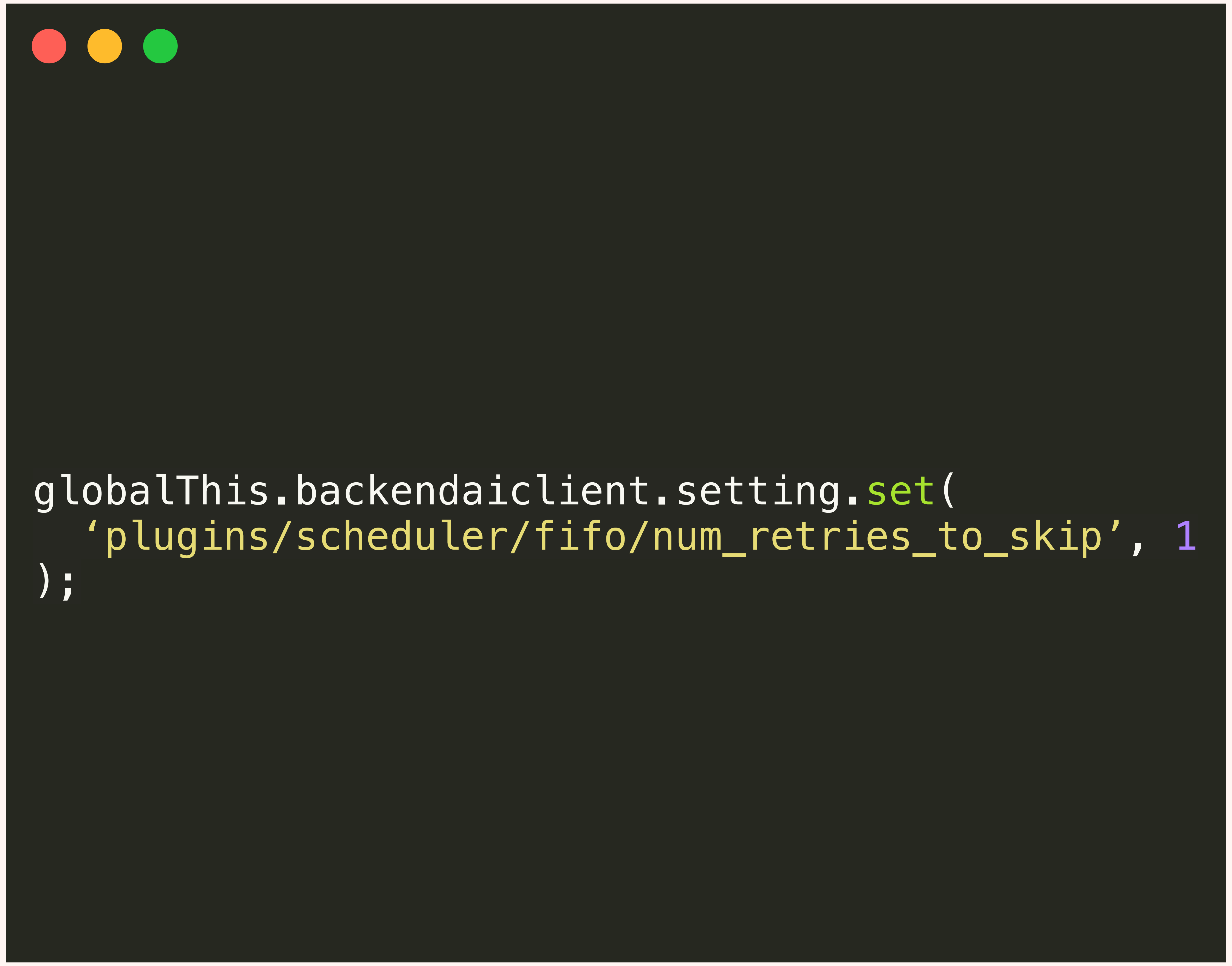
/backend.ai-webui/src/lib/backend.ai-client-node.ts

Backend.AI manager 에 요청

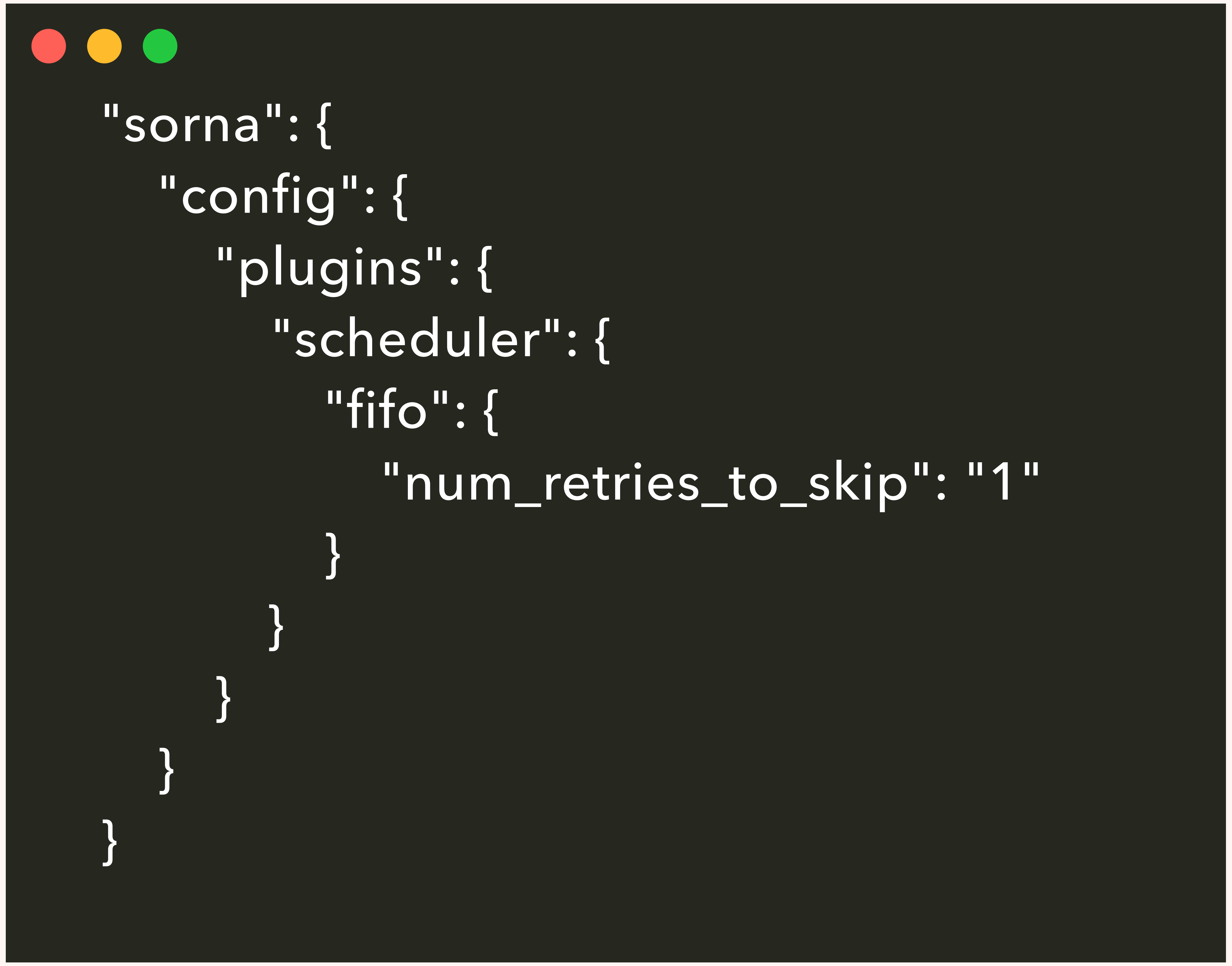
(_wrapWithPromise() : Backend.AI 매니저의 비동기 요청을 위한 promise wrapper)



```
/**
 * Get settings
 *
 * @param {string} prefix - prefix to get. This command will return every settings starting with the prefix.
 */
async get(key) {
  key = `config/${key}`;
  const rqst = this.client.newSignedRequest("POST", "/config/get", {"key": key, "prefix": false});
  return this.client._wrapWithPromise(rqst);
}

/**
 * Set a setting
 *
 * @param {string} key - key to add.
 * @param {object} value - value to add.
 */
async set(key, value) {
  key = `config/${key}`;
  const rqst = this.client.newSignedRequest("POST", "/config/set", {key, value});
  return this.client._wrapWithPromise(rqst);
}
```



```
globalThis.backendaiclient.setting.set(
  ‘plugins/scheduler/fifo/num_retries_to_skip’, 1
);
```



```
"sorna": {
    "config": {
        "plugins": {
            "scheduler": {
                "fifo": {
                    "num_retries_to_skip": "1"
                }
            }
        }
    }
}
```

THANK YOU '◡'✿