knox

Release 0.1.14

Jun 10, 2021

Contents

Warning: This python project is not fully baked yet. When a full major version is bumped we will consider it functional. Until then enjoy reading the docs.

Note: **TL;DR;** A set of certificate management utilities using a default Vault backend.

CHAPTER 1

What is Knox v0.1.14

The name is derived from "Fort Knox" the safest place to store valuables in history. At least that is the myth. This tool or set of utilities is explicitly for managing TLS certificates including metadata about them and storing it in a backend.

Primary components used are Python, Hashicorp Vault, Let's Encrypt and certbot.

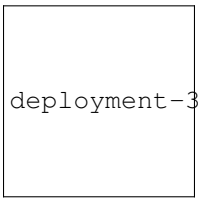
[Let's Encrypt](<https://letsencrypt.org>) is a certificate authority managed by the [Internet Security Research Group (ISRG)](<https://www.abetterinternet.org/about/>). It utilizes the [Automated Certificate Management Environment (ACME)](<https://github.com/ietf-wg-acme/acme/>) to automatically deploy free SSL certificates that are trusted by nearly all major browsers. [The certificate compatibility list can be found here](<https://letsencrypt.org/docs/certificate-compatibility/>). Lets Encrypt has revolutionized the distribution of certificates for public facing servers.

[Hashicorp Vault](<https://www.vaultproject.io/>) is a tool for storing secrets. It has a [PKI Secret Engine](<https://www.vaultproject.io/docs/secrets/pki/index.html>) backend which allows to use it as a certificate authority in an internal public key infrastructure deployment. Until now, Vault is best suited for issuing private certificates.

Let's Encrypt and Hashicorp Vault are complementary in certificate management.

CHAPTER 2

Dataflow Diagram



There may not necessarily be a container between Certbot or the Devops agent but the key is all access to manage the certs goes through a knox command. Once in place the cert can be accessed directly from Vault by deployment mechanisms with or without knox. Essentially its just a key value path to json. Knox just unifies how and what is stored and provides convenience methods for managing the certs.

CHAPTER 3

Installation

To get started:

```
pip install knox
```

You can also install the in-development version with:

```
pip install git+ssh://git@github.com/8x8cloud/knox.git@develop
```

Or run it as a container:

```
docker run 8x8cloud/knox
```

See [Dynaconf](https://dynaconf.readthedocs.io/) for how the configuration is read in. At its simplest just add environment variables into a *.env* file.

CHAPTER 4

Metadata

Knox will store the certificate body, in its entirety, along with metadata related to the details of the certificate. The data will be organized and retreived using a tree struction mimicking the DNS naming heirarchy.

Tree Structure:

```
certificates:
├── com
│       └── example
│           └── cloud
│               ├── acceptance
│               ├── production
│               └── staging
├── internal
│   └── example
└── net
    └── example
```

As a result the host name www.example.com storage path will be /com/example/www

Additional data will be stored with the body of the certificates. A jinja template will be provided. Although the cert body and cert data will have alternate RBAC rules for accessing. Below is a sample:

```
{
    "cert_info":
    {
        "subject": {
             "commonName": "www.example.com",
             "countryName": "US",
             "emailAddress": "cert@example.com",
             "localityName": "San Jose",
             "organizationName": "Example, Inc.",
             "organizationalUnitName": "Engineering",
             "stateOrProvinceName": "CA"
        },
        "issuer": {
```

(continues on next page)

(continued from previous page)

```
            "commonName": "www.example.com",
            "countryName": "US",
            "emailAddress": "cert@example.com",
            "localityName": "San Jose",
            "organizationName": "Example, Inc.",
            "organizationalUnitName": "Engineering",
            "stateOrProvinceName": "CA"
        },
        "key_details": {
            "fingerprint_sha256":
↪"f6874a226e4d2ea54eed11d8d71e27f5fbd965630aa84f71414209b0227c448c",
            "key": {
              "size": 4096,
              "type": "RSA"
            },
            "serial_number": "11672594923309745709",
            "version": "v1"
        },
        "validity": {
            "not_valid_after": "2021-05-17 18:49:00",
            "not_valid_before": "2020-05-17 18:49:00"
        }
    },

    "cert_body":
    {
        "private": "REDACTED",
        "chain": "REDACTED",
        "public": "REDACTED"
    }
}
```

CHAPTER 5

Documentation

https://knox.readthedocs.io/

CHAPTER 6

Development

This project was initialized using a very cool python project templating tool called [cookiecutter-pylibrary](https://github.com/ionelmc/cookiecutter-pylibrary) from [Ionel Cristian Mărieș](https://github.com/ionelmc). Definitely check it out to see all the tools available and good usage docs.

Python setup on a Mac using [pyenv](https://github.com/pyenv/pyenv#readme):

```
# Pyenv for managing multiple versions of python
brew install pyenv

# Install all versions you want to test locally
pyenv install <version list>

# Enable the versions
pyenv local <version list>
```

To execute everything run:

```
tox
```

To see all the tox environments:

```
tox -l
```

To only build the docs:

```
tox -e docs
```

To build and verify that the built package is proper and other code QA checks:

```
tox -e check
```

To update [Travis CI](https://travis-ci.org) configuration:

```
tox -e bootstrap
```

You will need a [Vault](https://hub.docker.com/_/vault) server running locally:

```
>docker run \
--cap-add=IPC_LOCK \
-p 8201:8201 \
-p 8200:8200 \
-e 'VAULT_DEV_ROOT_TOKEN_ID=knox' \
-d --name=dev-vault \
vault

>docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
↪STATUS              PORTS                                NAMES
d89fbfd340c3        vault               "docker-entrypoint.s..."   5 hours ago
↪ Up 5 hours          0.0.0.0:8200-8201->8200-8201/tcp   dev-vault
```

Set the token ID and container name to your preferences. Verify you can talk to vault using the vault cli:

```
>export VAULT_ADDR=http://0.0.0.0:8200
>export VAULT_TOKEN=knox

>vault status

Key             Value
---             -----
Seal Type       shamir
Initialized     true
Sealed          false
Total Shares    1
Threshold       1
Version         1.4.1
Cluster Name    vault-cluster-31da8ea9
Cluster ID      043bfc14-09b1-6033-1c3b-8aeace3adc60
HA Enabled      false
```

Setup your local app role:

```
# Add the cert admin policy
>vault policy write cert_admin config/cert_admin-policy.hcl
Success! Uploaded policy: cert_admin

# Enable approle auth
>vault auth enable approle
Success! Enabled approle auth method at: approle/

# Create an app role
>vault write auth/approle/role/knox-admin \
  bind_secret_id=true \
  period=0 \
  policies="cert_admin" \
  token_num_uses=1 \
  token_ttl=5m \
  token_max_tll=30m \
  secret_id_num_uses=0 \
  secret_id_ttl=0 \
  token_no_default_policy=true
Success! Data written to: auth/approle/role/knox-admin
```

(continues on next page)

(continued from previous page)

```
# Read role-id
vault read auth/approle/role/knox-admin/role-id
export KNOX_VAULT_APPROLE=$(vault read -format=json auth/approle/role/knox-admin/role-
↪id | jq -r '.data.role_id')

# generate secret-id
vault write -f auth/approle/role/knox-admin/secret-id
export KNOX_VAULT_SECRET_ID=$(vault write -f -format=json auth/approle/role/knox-
↪admin/secret-id | jq -r '.data.secret_id')
```

Update your knox configuration using *.env* or direct environment variables:

```
ENVVAR_PREFIX_FOR_DYNACONF=KNOX
INCLUDES_FOR_DYNACONF='./config/*'

KNOX_TEMP=/tmp
KNOX_LOG_LEVEL=DEBUG
KNOX_STORE_ENGINE=vault
KNOX_VAULT_URL=http://127.0.0.1:8200
KNOX_VAULT_TOKEN="knox"
KNOX_VAULT_MOUNT="certificates"
KNOX_VAULT_CLIENT_MAX_VERSIONS=10
KNOX_VAULT_CLIENT_CAS=False
KNOX_FILE_HOME=./test
```

And Or use a settings file:

```
{
  "default": {
    "ENVVAR_PREFIX_FOR_DYNACONF": "KNOX",
    "INCLUDES_FOR_DYNACONF": "./config/*",
    "KNOX_TEMP": "./tmp",
    "KNOX_LOG_LEVEL": "DEBUG",
    "KNOX_STORE_ENGINE": "vault",
    "KNOX_VAULT_URL": "http://127.0.0.1:8200",
    "KNOX_VAULT_TOKEN": "knox",
    "KNOX_VAULT_MOUNT": "certificates",
    "KNOX_VAULT_CLIENT_MAX_VERSIONS": "10",
    "KNOX_VAULT_CLIENT_CAS": "True",
    "KNOX_FILE_HOME": "./test"
  },
  "development": {
    "ENVVAR_PREFIX_FOR_DYNACONF": "KNOX",
    "INCLUDES_FOR_DYNACONF": "./config/*"
  },
  "production": {
    "ENVVAR_PREFIX_FOR_DYNACONF": "KNOX",
    "INCLUDES_FOR_DYNACONF": "./config/*"
  }
}
```

Generate some test self signed certificates:

```
# create a config file for openssl
[req]
distinguished_name = req_distinguished_name
x509_extensions = v3_req
```

(continues on next page)

(continued from previous page)

```
prompt = no
[req_distinguished_name]
C = US
ST = VA
L = SomeCity
O = MyCompany
OU = MyDivision
CN = www.company.com
[v3_req]
keyUsage = keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
[alt_names]
DNS.1 = www.company.net
DNS.2 = company.com
DNS.3 = company.net

openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 \
 -keyout cert-key.pem \
 -out cert-pub.pem \
 -config san.cnf -extensions 'v3_req'

xtensions 'v3_req'
Generating a 2048 bit RSA private key
.................................+++
...........+++
writing new private key to 'cert-key.pem'
-----
```

Save a certificate to vault:

```
export VAULT_ADDR=http://localhost:8200
export KNOX_VAULT_URL=http://localhost:8200
export KNOX_VAULT_TOKEN=knox
export KNOX_VAULT_APPROLE=$(vault read -format=json auth/approle/role/knox-admin/role-
↪id | jq -r '.data.role_id')
export KNOX_VAULT_SECRET_ID=$(vault write -f -format=json auth/approle/role/knox-
↪admin/secret-id | jq -r '.data.secret_id')

knox cert --pub cert-pub.pem --key cert-key.pem save www.company.com
```

Search for stored certificates:

```
knox store find \*              # list all the certificates info
knox store find www.company.com
knox store find *.example.com   # list all the *.example.com certificates
knox store find com/example/www # list about www.example.com
```

Don't want to install python, I got you:

```
docker run --net=host 8x8cloud/knox --help
Usage: knox [OPTIONS] COMMAND [ARGS]...

  Utilities for managing and storing TLS certificates using backing store
  (Vault).

Options:
```

(continues on next page)

(continued from previous page)

```
  -l, --log [TRACE|DEBUG|INFO|SUCCESS|WARNING|ERROR|CRITICAL]
                                  Sets the level of logging displayed
                                  [default: INFO]

  -v, --verbose                   Display log output to console
  --version                       Show the version and exit.
  --help                          Show this message and exit.

Commands:
  cert   Certificate utilities.
  store  Store commands.
```

If using docker mount a volume to get to your certs:

```
docker run --net=host \
-v ~/dev/knox/examples/:/examples \
8x8cloud/knox cert \
--pub /examples/sample_cert1.pem \
--key /examples/sample_key1.pem \
save www.example.com
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER 7

Installation

At the command line:

```
pip install knox
```

CHAPTER 8

Usage

To use knox in a project:

```
import knox
```

CHAPTER 9

Reference

9.1 knox

class knox.**Knox**(*ctx: dict*)

Bases: object

Composite class for Knox package

attach(*engine_name: str*) → bool
Instantiate an additional store

conf
Access to the knox Conf object

settings
Access to the Dynaconf settings object

store
Access to the default store engine

stores(*engine_name: str = None*) → knox.backend.store.Store
Retrieve a named store, otherwise the default store

9.2 knox.backend

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class knox.backend.**Store**(*settings*, *engine_name: str = 'vault'*)
Bases: object

Abstract class to generalize access to the different stores

_engine_map = {'aws': <class 'knox.backend.store_acm.ACMStoreEngine'>, 'file': <clas

delete(*path: str*, *name: str*) → bool
Remove the object from the store

find(*pattern: str*) → list
Given a pattern, return collection of all objects Search patterns : abc.8x8.com, abc.8x8.com/*, 8x8.com/*

get(*path: str*, *name: str*, *type=None*) → knox.backend.store_object.StoreObject
Given path read object

save(*obj: knox.backend.store_object.StoreObject*) → bool
Save the given object to persistence

subjectaltfind(*pattern: str*) → list
Fetch the certificate information based on subject alternative name

class knox.backend.**StoreObject**(*name: str*, *path: str*, *body: str*, *info: str*, *type=None*)
Bases: object

Metadata interface for objects being persisted in a backend

_body = None
Content that will be persisted

_data = None
Complete map of object

_info = None
Metadata about the object being stored

_mount = None
Mount point

_name = None
Name of the objects store key

_path = None
Path from store mount point to find store key

_type = None
A way to classify StoreObjects

_version = None
Store revision

body
Content to persist, typically JSON

data

info
Object metadata

static md5(*obj: {}*) → str

name
Object name

path
Path attribute

path_name
Convenience method to generate path/name for store

type

version
Object version

class knox.backend.**StoreEngine**
Bases: object

The abstract persistence strategy for storing the certificates

close() → bool
Close access to the persistence

delete(*path: str*, *name: str*) → bool
Delete from the store

initialize() → bool
Ensure the store is configured properly

open() → bool
Initialize access to the persistence

read(*path: str*, *name: str*, *type=None*) → knox.backend.store_object.StoreObject
Read from the store

write(*obj: knox.backend.store_object.StoreObject*) → bool
Write to the store

class knox.backend.**VaultStoreEngine**(*settings*)
Bases: knox.backend.store_engine.StoreEngine

Vault implementation of the StoreEngine interface

close() → bool
Ensure we close the vault connection

delete(*path: str*, *name: str*) → bool
Delete from the store

find(*pattern*) → list
Search certificate info for a given search pattern

Parameters **pattern** (*str*) – Search glob pattern ex: *, abc.8x8.com, abc.8x8.com/*, 8x8.com/*

Returns list

initialize() → bool
Ensure the Vault client is initialized

open() → bool
Ensure the Vault client is connected

read(*path: str*, *name: str*, *type=None*) → knox.backend.store_object.StoreObject
Using the provided path and name retrieve the data from the store and create a new StoreObject

Parameters

- **path** (*str*) – Store path to the object
- **name** (*str*) – Name of the object to retrieve
- **type** (*str*) – StoreObject type, if known

Returns StoreObject

write(*obj: knox.backend.store_object.StoreObject*) → bool
Given a StoreObject, store it into vault using mount/path/name == body,info

Parameters **obj** (StoreObject) – The StoreObject to persist

Returns bool

class knox.backend.**VaultClient**(*settings: dynaconf.base.LazySettings*)
Bases: object

Client commands not available via hvac

_VaultClient__headers = {'Content-Type': 'application/json', 'X-Vault-Token': ''}

__approle = None
Application Role ID

__mount = None
Engine mount path

__mounts = None
Map of Vault mounts

__secretid = None
Application Role Secret ID

__token = None
Auth token

__url = None
Vault server URL

_get(*path: str*) → <module 'json' from '/home/docs/.pyenv/versions/3.7.9/lib/python3.7/json/__init__.py'>
GET REST API wrapper method

Parameters **path** (*String*) – Vault API to query

Returns JSON paylod

_post(*path: str*, *data: <module 'json' from '/home/docs/.pyenv/versions/3.7.9/lib/python3.7/json/__init__.py'>*)
→ <module 'json' from '/home/docs/.pyenv/versions/3.7.9/lib/python3.7/json/__init__.py'>
POST REST API wrapper method

Parameters

- **path** (*String*) – Vault API to change or create
- **data** (*JSON*) – Required request body

Returns requests.Response object

_put(*path: str*, *data: <module 'json' from '/home/docs/.pyenv/versions/3.7.9/lib/python3.7/json/__init__.py'>*)
→ requests.models.Response
PUT REST API wrapper method

Parameters

- **path** (*String*) – Vault API to change or create
- **data** (*JSON*) – Required request body

Returns requests.Response object

connect() → bool
Knox uses an approle scheme to authenticate with Vault. This requires fetching a fresh, short lived, API token for every call to the API.

get_mounts() → <module 'json' from '/home/docs/.pyenv/versions/3.7.9/lib/python3.7/json/__init__.py'>
Refresh set of mounts from Vault

Returns JSON

initialize() → bool
During initialization, if in admin mode, ensure the kv mount point has been registered with Vault. To enable admin mode use the hidden param –admin with any command.

knox –admin store find

logout() → bool

match = 'False'

mount

new_mount(*mount: str*) → bool
Will create a vault mount of type k/v V2 if it doesn't exist

Parameters **mount** (*String*) – Name of the Vault K/V Secret Engine

Returns Boolean

read(*path: str*, *name: str*, *type: str = None*) → tuple
Given a path and name retrieve a tuple of dictionaries to create a StoreObject cert_body cert_info

Parameters

- **path** (*str*) – The path where the StoreObjects data is stored
- **name** (*str*) – Name of the StoreObject to retrieve
- **type** (*str*) – The type of StoreObject i.e. PEM

search(*rootpath: str*, *rootkey: str*, *searchresults: list*, *pattern: str = None*) → list
Search for 'cert_info' for a given vault path

Parameters

- **rootpath** (*str*) – Beginning search path
- **rootkey** (*str*) – Used to get commonname from search path
- **searchresults** (*list*) – Stores the search results..default is empty
- **pattern** (*str*) – Unaltered search pattern

Returns list

token

upsert(*obj: knox.backend.store_object.StoreObject*) → bool
Given a StoreObject create or update it into Vault. Metadata and content are stored separately to allow querying of non sensitive details.

param obj The object to store

type obj StoreObject

return Boolean

url

class knox.backend.**FileStoreEngine**(*settings*)
Bases: knox.backend.store_engine.StoreEngine

close() → bool
Close access to the persistence

delete(*path: str*, *name: str*) → bool
Delete from the store

initialize() → bool
Ensure the store is configured properly

open() → bool
Initialize access to the persistence

read(*path: str*, *name: str*, *type=None*) → knox.backend.store_object.StoreObject
Read from the store

write(*obj: knox.backend.store_object.StoreObject*) → bool
Write to the store

9.3 knox.certificate

Apache Software License 2.0

Copyright (c) 2020, 8x8, Inc.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

https://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class knox.certificate.**Cert**(*settings: dynaconf.base.LazySettings*, *common_name=None*)
Bases: knox.backend.store_object.StoreObject

Object representation of a TLS certificate

class CertTypes
Bases: enum.Enum

An enumeration.

DER = 2

PEM = 1

PFX = 3

valid = <bound method Cert.CertTypes.valid of <enum 'CertTypes'>>

DER = 2

PEM = 1

PFX = 3

_body = None
String representation of private, chain and public portions of certificate as a map/json

_common_name = None
Defaults to value from certificate

_data = None
Combined body and info map

_file = None
Raw file contents of certificate

_info = None
Certificate details

_jinja = None
Template engine

_mount = None
Based on certificate its mount is either KNOX_VAULT_MOUNT or KNOX_VAULT_MOUNT/client

_path = None
Objects stored using <mount><path><name><type>

_policy = None
Vault access policy, gen from jinja template, explicit to instance of cert

_type = None
Certificate type identifier

_x509 = None
Parsed data object from raw file

body() → str
Content to persist, typically JSON

chain
Unless its a dict, its not loaded yet

data
Content to persist, typically JSON

generate() → None
Generate certificate for a given common name

info() → str
Object metadata

isValid() → bool
Check certificate validity period

issuer() → str
Return the certificate issuer details

key_details() → str
Return characteristics of key used to generate the certificate

load(*pub: str*, *key: str*, *certtype: enum.Enum = <CertTypes.PEM: 1>*, *chain: str = None*) → None
Read in components of a certificate, given filename paths for each

Parameters

- **pub** (*str*) – File name of public portion of key
- **key** (*str*) – File name of private portion of key
- **chain** (*str*) – File name of intermediate certificates. Optional as they could be in pub

- **certtype** (*Enum*) – Enum of certificate types [PEM=1, DER=2]

load_x509(*path: str*) → None
Given path to PEM x509 read in certificate

Parameters **path** (*str*) – File path to x509 PEM file

static md5(*obj: {}*) → str

mount

name
Object name

path
Path attribute

path_name
Convenience method to generate path/name for store

policy() → str

policy_mount

private
Unless its a dict, its not loaded yet

public
Convenience method for Jinja2 templates. Jinja2 does not process the string if it has carriage returns.

subject() → str
Return the certificate subject details

subjectaltnames() → str
Return Subject alternate names

static to_store_path(*common_name: str*) → str
Generate a backend store path based on the certificates common name www.example.com becomes /com/example/www

return str

type

classmethod valid_name(*value: str*) → str
Some engines might have problems with astrix, as they are used for glob searching and or RBAC. Replace it with the key word 'wildcard'. This does not affect the actual certificate.

validity() → str
Return the certificates dates of validity

version
Object version

9.4 knox.config

Apache Software License 2.0

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

https://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class knox.config.**Conf**(*loglevel=None*)

Bases: object

Manage application settings

_banner = '\n_____ __\n___ //_/_____ __\n__ ,< __ __ \\ __ _ |/_/\n_ /|

classmethod log_filter(*record*) → bool

log_level = 'INFO'

classmethod set_loglevel(*level: str*) → None

settings

version

CHAPTER 10

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

10.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

10.2 Documentation improvements

knox could always use more documentation, whether as part of the official knox docs, in docstrings, or even on the web in blog posts, articles, and such.

10.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/8x8cloud/knox/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

10.4 Development

To set up *knox* for local development:

1. Fork knox (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com/8x8cloud/knox.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes run all the checks and docs builder with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

10.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

10.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 11

Authors

- Lance Johnson - https://github.com/rljohnsn
- Shankar Balajagadeesan - https://www.8x8.com
- Kadiri Purushotham Reddy - https://www.8x8.com

CHAPTER 12

Changelog

12.1 0.0.0 (2020-05-08)

- First release on PyPI.

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index

Symbols

A

B

C

D

F

G

I