Python 101

A Practical Introduction To Cryptography, Revisiting Docker, and Working with Databases in Python.

Contents

A practical introduction to Cryptography

Obfuscation - Making the actual data hard to understand. There is no secret involved. Eg. ROT13 (obfuscation used in **this** module of python3)

Hashing - Computing a shorter semi-unique key for larger chunks of data. Eg. SHA1, SHA256, MD5.

Encryption - Converting the data in a impossible to understand form without knowledge of the shared secret.

A practical introduction to Cryptography
(Encryption)

Encryption can be done using two ways-

1. Symmetric (Same keys used on both sides) - Fast but less secure.
2. Asymmetric (Different Keys used) - Slow but more secure.

In python we can perform encryption using external module **cryptography**.

https://cryptography.io/en/latest/

For encryption used keys are just a cryptographically secure random byte sequence. This can be generated using **secrets** module in python3.

see ex_crypto/ex_keygen.py

A practical introduction to Cryptography

(Encryption contd..)

Generally we do not use Asymmetric Key Cryptography directly to encrypt our data. Instead we Encrypt the Symmetric key with which we encrypt our data.

Encryption provides us Confidentiality only.

A practical introduction to Cryptography
(Hashing)

Hashing is done in python3 using **hashlib** module.

Hashing provides us **integrity** ie. The data is what we sent. There are several methods to hash data. But we use **SHA256** to hash sensitive data and **MD5** or **SHA1** to hash general data.

Sensitive data can be passwords. General Data can simply be our OS Images etc.

HMACs are Encrypted Hashes they are used to provide **confidentiality with integrity.**

A practical introduction to Cryptography
(Hashing contd..)

Mind that Hashing is not encryption.

Hashing has wholly different objective and Encryption has different.

Both are executed in completely different manner.

Hashing only tells if the data that we have received is what we were intended to receive.

Whereas, Encryption tries to hide the data from everyone else but the one for whom the data was intended to be received.

Docker - A primer

Docker is a containerization tool. It bundles all of the requirements of our projects irrespective of what the requirements are (ie. not only python specific, eg. MongoDB might be a dependency of our project) into a container which can run on any system.

Thus it makes dependency management in a project much easier.

Containers - A container is a lightweight standalone, stateless, executable package of a piece of software that includes everything needed to run it.

Stateless meaning that none of the data in a container persists. It wipes it all each time it restarts.

Docker - A primer contd..

Containers vs Virtual Machines - A VM runs on a system much like a container does. But the VM is not specific to a piece of software. Its literally a whole new system within a system. VMs are generally hard to setup and much much harder to manage upon that due to mismanaged Hypervisors the resource usage is also significantly higher than containers.

CaaS or Containers as a service thus became an alternative due to relatively harder management of VMs.

So instead of bothering about installing everything separately we need to just install one thing that is Docker. The rest of it will be managed by Docker.

Docker - A primer contd..

sudo apt install docker.io ← Install docker

As docker needs access to system resources in order to prepare containers it needs superuser priviledges. So we need to run each docker command as **sudo.** But an alternative approach is to add our user to the docker usergroup. Where docker usergroup has the required permissions for functioning docker.

sudo usermod -aG docker ${USER} ← Add user to docker

Further we will learn how to, **create, run, stop, attach and build** a container.

Docker - A primer contd..

There are a lot of containers people have built already and have stored on DockerHub much like people have created packages and stored in on PyPI.

We can use those containers on our system using **create.**

- **docker create -it ubuntu bash** ← it means create a ubuntu container and open an interface to it and run bash on the interface.
- **docker ps -a** ← see history of containers that have run or are running.

Let the id of created container be 12345678

- **docker start 12345678** ← run the container with id 12345678

Docker - A primer contd..

- **docker attach 12345678** ← attach our terminal's io to the io of container with id 12345678. Now using this terminal we can run any command inside the container.
- **docker rm 12345678** ← remove the container with id 12345678

Volumes in docker - As we know containers are stateless ie. they wipe data on each run. What if we needed some persisting data? Its quite natural that we might. Like what if the container is of a Database? We will need some persisting data in that case.

Docker - A primer contd..

We know that we can persist data on our base system by creating files and directories.

What if we could map a directory to a directory inside the container? This way all the content on our base system inside that directory could be accessed by container which would persist as the content actually resides on our base system instead of the containers. This is achieved using volumes.

- **docker create -it -v ~/some_directory_base:/some_directory ubuntu bash** ← create a container with ubuntu and start bash in it and map the some_directory_base at home to some_directory at root in container.

Docker - A primer contd..

Mapping ports - Every service runs on a port in our systems. Eg. 443 for https, 80 for http, 27017 for MongoDB, 6379 for Redis, 5432 for PostgreSQL, and 3306 for MySQL etc.

So if a service is running inside our container with which we want to communicate then we need to be able to map the ports of that container to the ports of our base system to be able to use those services on our base systems.

We can achieve this using this command -

- **docker create -it -p port_of_base:port_of_container ubuntu bash**

Docker - A primer contd..

Last thing left here is that if we want to just give our application to someone we can not. Because we will have to tell the person create a container do this and that. What if we could automate even that process ?

We use **Dockerfile** for this purpose. Dockerfile defines what steps to perform in order to build an image.

Docker - A primer contd..

FROM base_image_to_use

MAINTAINER “Name of the creator”

RUN commands_to_run_on_creation

VOLUME volume_to_mount_from_base

EXPOSE port_to_expose

WORKDIR directory_path_in_which_to_work_on_startup_of_container

CMD command_to_run_on_container_startup

Docker - A primer contd..

```
FROM ubuntu

VOLUME /content

RUN apt update

RUN apt install -y python3

EXPOSE 8000

WORKDIR /content

CMD python3 -m http.server
```

Docker - A primer contd..

Building the container image using the dockerfile-

- **docker build . -t demo_image:version1** ← Build a docker image using the Dockerfile available in current directory with the name as demo_image and tag as version1.
- **docker create -it -v ~/content_to_host:/content -p 8000:8000 demo_image:version1** ← Create a container based on image docker_image:version1 and and map content_to_host folder on base to it's content folder and map the port 8000 of base to its port 8000

Docker - A primer contd..

Building the container image using the dockerfile-

- **docker build . -t demo_image:version1** ← Build a docker image using the Dockerfile available in current directory with the name as demo_image and tag as version1.
- **docker images** ← list all of the locally available images
- **docker rmi 12345678** ← remove the image with image id as **12345678**
- **docker create -it -v ~/content_to_host:/content -p 8000:8000 demo_image:version1** ← Create a container based on image docker_image:version1 and and map content_to_host folder on base to it's content folder and map the port 8000 of base to its port 8000

Docker - A primer contd..

Now we have the capability of creating an image ourselves but we still need to tell others what port to be mapped and what directory to be mapped etc. It would be great if we could automate even this.

Docker-compose is the tool that is aimed at achieving this.

Install it as-

```
sudo apt install docker-compose
```

Docker - A primer contd..

See the file alongside Dockerfile named **docker-compose.yml**

Run command **docker-compose build** to build the image corresponding to the docker-compose.yml's build key.

Run command **docker-compose up** to start the docker image.

This is how we can ship our project's environment across platforms.

Databases

Data - Base = A place where you can store data and retrieve it later from.

Now when we talk about such work related to data read and write it wont be efficient if we just did it on a regular file as the overhead of reading and writing would be too much.

So instead of simple files we use specially formatted files to store data in. These files store data in form of mathematical models that we call as data structures. Commonly used data structures are tries, BSTs, B Trees, Linked Lists, HashTables etc.

Databases

There is a different **set of rules** for accessing each database. So we can not just read the database files as it is. Also we need to be able to access databases remotely as its not necessary that our database and our server must be on the same device.

So for externally accessing the database we expose a server socket that reads that database. And for defining the set of rules based on which the database must be accessed we define a protocol as well.

eg. MongoDB's Database Server runs on IP:**27017** by default and the protocol is **mongodb.** This is how the Database servers work.

Databases

We have defined how the server part of database works.

In order to access it we need some sort of client. The client must obviously be a client socket which can send requests to our database's server socket based on the database's protocol. This client socket taht can communicate based on database's protocol is called database driver.

As every database has different protocol. So every database has different database driver.

Relational Database - PostgreSQL

A Relational Database is the one which has data stored in form of Relations.

These relations are also called Tables.

Each Table contains columns that are called attributes.

There are several rows indicating one entry in the table based on the attributes.

Each table has a unique attribute for each row that is called primary key.

And the tables can be related with each other with the help of an attribute known as foreign key.

Relational Database - PostgreSQL

In order to better understand it, We should work with a relational database, we will be using PostgreSQL for this purpose.

PostgreSQL is completely opensource under MIT license, It supports ACID (Atomicity, Consistency, Isolation and Durability) properties of transactions completely, its easier to clusterize, more secure and even provides some NoSQL features.

See link below for thorough comparision of it with prominent SQL db ie. MySQL

https://www.2ndquadrant.com/en/postgresql/postgresql-vs-mysql/

Relational Database - PostgreSQL - (installation & usage)

We can simply install it using docker using the docker-compose file.

`See ex_databasing/ex_postgre`

The driver in python for PostgreSQL is **psycopg2**

```
pip3 install psycopg2
```

See CRUD (Create, Read, Update, Delete) in ex_databasing/ex_postgre

NoSQL Database - MongoDB - (installation & usage)

NoSQL databases are non relational ie. There are no tables.

Instead of tables we use **collections** in NoSQL databases.

Each collection has data records saved in it in form of **documents.**

In MongoDB documents are saved in BSON format which is exactly same as of JSON.

NoSQL databses are ideal to dump data in which has no schema. Ie. No Fixed format. Like data emitted by sensors or data for analytics. These are extremely scalable and can store large chunks of data.

NoSQL Database - MongoDB - (installation & usage)

We can simply install it using docker using the docker-compose file.

`See ex_databasing/ex_mongo`

The driver in python for MongoDB is **pymongo**

```
pip3 install pymongo
```

See CRUD (Create, Read, Update, Delete) in ex_databasing/ex_mongo

NoSQL Key-Value Database - Redis - (installation & usage)

Redis is an in-memory key value NoSQL database.

Ie. instead of saving a document like mongodb or tables like postgreSQL it stores data in a mapping.

It is very fast being in-memory, Open Source under BSD license and it's really good for frequently updated real-time data, such as session store, state database, statistics, caching.

Key-Value Database - Redis - (installation & usage)

We can simply install it using docker using the docker-compose file.

`See ex_databasing/ex_redis`

The driver in python for MongoDB is **redis-py.**

```
pip3 install redis
```

See get, set in ex_databasing/ex_redis