Authentication

Authentication Overview

- Registration
  - User sends username and password
  - Validate password strength
  - Store salted hash of the password
- Authentication
  - User sends username/password
  - Retrieve the stored salted hash
  - Salt and hash the provided password
  - If both salted hashes are identical, the user is authenticated

Authentication Tokens

- Need to avoid asking for username/password on every request
- When a user is authenticated, generate a random token
  - The token should have enough entropy that is cannot be guessed
  - Generally, there should be at least 2^80 unique tokens that could be generated
- Associate this token with the user

Authentication Tokens

- Need to avoid asking for username/password on every request
- When a user is authenticated, generate a random token
  - The token should have enough entropy that is cannot be guessed
  - Generally, there should be at least 2^80 unique tokens that could be generated
- Associate this token with the user

Authentication

- Once a token is generated, set it as a cookie
  - Can sign the cookie first for more security
- Now the token will be sent with all subsequent requests
- Use the token to lookup the user
- The possession of the token verifies that this user did authenticate in the past

Cookies

- **Caution**: These tokens need to be stored on the server
- These tokens are as sensitive as passwords!
    - Stealing a token and setting a cookie with that value grants access to an account without even needing a password
- **Solution**: Only store hashes of the tokens
    - Can salt for extra security (Not necessary since the entropy is so high)

Authentication w/ Tokens

- Check each request for a cookie with a token
  - Lookup the hash of the token in the database
  - If the token is found, read the associated username
  - Proceed as though this request was made by that user
- If the token is invalid or no cookie is set
  - Redirect to the login page
- Ensure all sensitive pages/features are secured this way!
  - Remember, the front end cannot be trusted
  - A user can manually make any request

Docker Compose

docker-compose.yml

```yaml
version: '3.3'
services:
  mongo:
    image: mongo:4.2.5
  app:
    build: .
    environment:
      WAIT_HOSTS: mongo:27017
    ports:
      - '8080:8000'
```

- We need the application and database both running
- We did this with docker-compose
- Let's walk through a docker-compose.yml file

Docker Compose

docker-compose.yml

```yaml
version: '3.3'
services:
  mongo:
    image: mongo:4.2.5
  app:
    build: .
    environment:
      WAIT_HOSTS: mongo:27017
    ports:
      - '8080:8000'
```

- Specify the docker compose file format version

Docker Compose

docker-compose.yml

```
version: '3.3'
services:
  mongo:
    image: mongo:4.2.5
  app:
    build: .
    environment:
      WAIT_HOSTS: mongo:27017
    ports:
      - '8080:8000'
```

- List all of the services for docker compose to run
- A docker container is created for each service

Docker Compose

docker-compose.yml

```
version: '3.3'
services:
  mongo:
    image: mongo:4.2.5
  app:
    build: .
    environment:
      WAIT_HOSTS: mongo:27017
    ports:
      - '8080:8000'
```

- Name each service
- These names are used as the hostnames for each container
  - Used to communicate between containers

Docker Compose

docker-compose.yml

```
version: '3.3'
services:
  mongo:
    image: mongo:4.2.5
  app:
    build: .
    environment:
      WAIT_HOSTS: mongo:27017
    ports:
      - '8080:8000'
```

- This service named 'mongo' uses a pre-build image
  - Same as having a 1-line Dockerfile:
    - "FROM mongo:4.2.5"
- No Dockerfile is needed

Docker Compose

docker-compose.yml

```
version: '3.3'
services:
  mongo:
    image: mongo:4.2.5
  app:
    build: .
    environment:
      WAIT_HOSTS: mongo:27017
    ports:
      - '8080:8000'
```

- This service named 'app' uses a Dockerfile
- Use 'build' to specify the path to build from
- Same as the trailing '.' when building an image

Docker Compose

docker-compose.yml

```yaml
version: '3.3'
services:
  mongo:
    image: mongo:4.2.5
  app:
    build: .
    environment:
      WAIT_HOSTS: mongo:27017
    ports:
      - '8080:8000'
```

- Use 'environment' to set any needed environment variables
- If using MySQL, set variables for your username/password

Docker Compose

docker-compose.yml

```
version: '3.3'
services:
  mongo:
    image: mongo:4.2.5
  app:
    build: .
    environment:
      WAIT_HOSTS: mongo:27017
    ports:
      - '8080:8000'
```

- We use an environment variable to tell our app to wait until the database is running before connecting to it

```
FROM python:3.8.2

ENV HOME /root
WORKDIR /root

COPY . .
RUN pip install -r requirements.txt

EXPOSE 8000

ADD https://github.com/ufoscout/docker-compose-wait/releases/download/2.2.1/wait /wait
RUN chmod +x /wait

CMD /wait && python app.py
```

Docker Compose

docker-compose.yml

```
version: '3.3'
services:
  mongo:
    image: mongo:4.2.5
  app:
    build: .
    environment:
      WAIT_HOSTS: mongo:27017
    ports:
      - '8080:8000'
```

- Map a local port to a container port
- Same as using "--publish 8080:8000" when running a single container

Docker Compose

docker-compose.yml

```yaml
version: '3.3'
services:
  mongo:
    image: mongo:4.2.5
  app:
    build: .
    environment:
      WAIT_HOSTS: mongo:27017
    ports:
      - '8080:8000'
```

- This file is used to build both images and run both containers using docker-compose

Docker Compose

docker-compose.yml

```
version: '3.3'
services:
  mongo:
    image: mongo:4.2.5
  app:
    build: .
    environment:
      WAIT_HOSTS: mongo:27017
    ports:
      - '8080:8000'
```

~~`mongo_client = MongoClient('localhost')`~~

```
mongo_client = MongoClient('mongo')
```

- Recall that we chose names for each service
- When connecting to the database in your app
    - The service name is the hostname for the container

Docker Compose

docker-compose.yml

```yaml
version: '3.3'
services:
  mongo:
    image: mongo:4.2.5
  app:
    build: .
    environment:
      WAIT_HOSTS: mongo:27017
    ports:
      - '8080:8000'
```

~~`mongo_client = MongoClient('localhost')`~~

```python
mongo_client = MongoClient('mongo')
```

- Instead of using "localhost"/"127.0.0.1"/"0.0.0.0"
- Use the name of the service
- docker-compose will resolve this hostname to the appropriate container

Example App