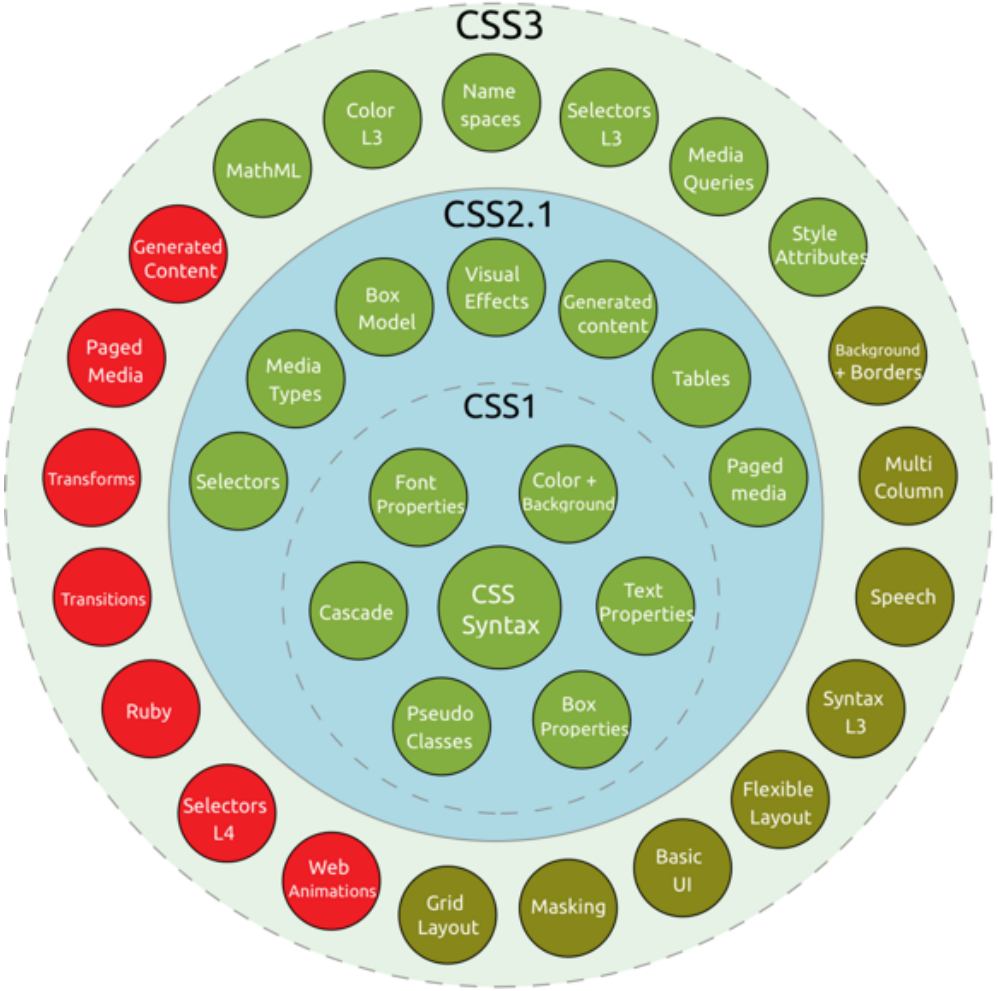
Last time 20190517

- Front-end tools and skills, CSS framework and time/date server/client management web service, including bootstrap, different CSS frameworks, time-date management, flask-bootstrap, and flask-moment
- Other skill-sets to extend JavaScript, Node JS, React/Redux, Vue, Angular JS…
- A simple web service with improved front-end

Modern look and more – From Backend to Frontend

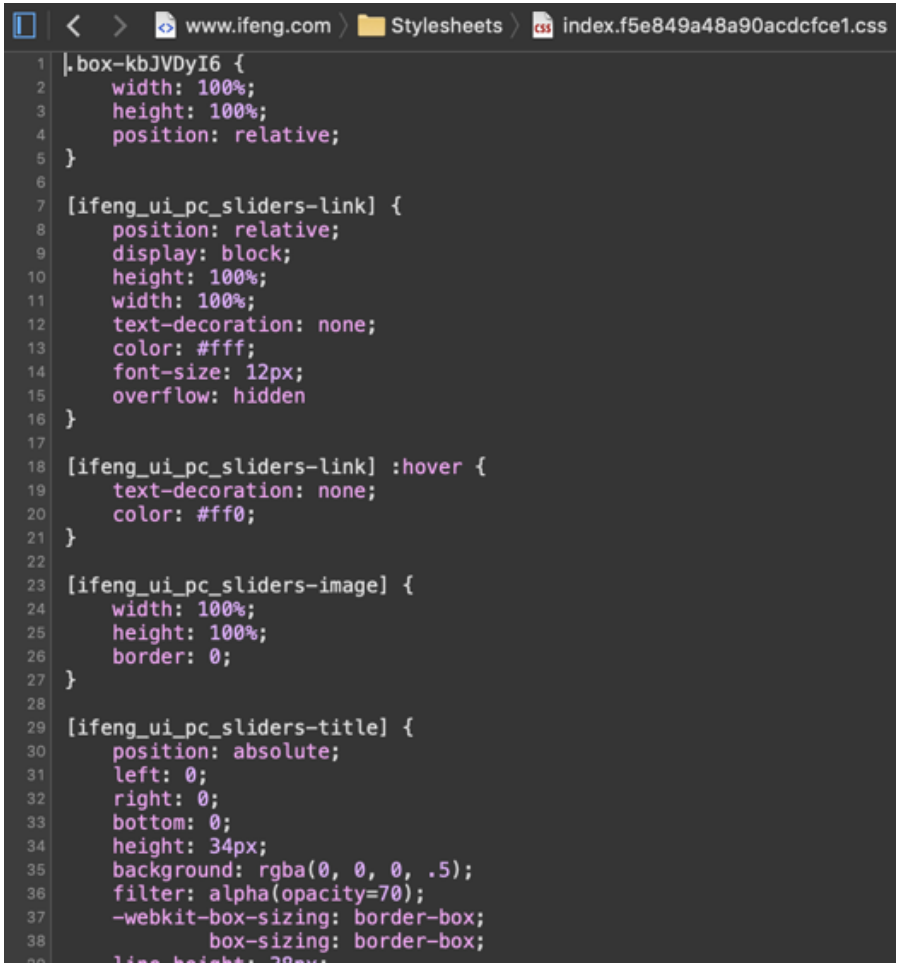
- While we can argue that coding is hard, our pains are nothing compared to those of web designers, who have to write templates that have a nice and consistent look on a list of web browsers.
- It has gotten better in recent years, but there are still obscure bugs or quirks in some browsers that make the task of designing web pages that look nice everywhere very hard.
- This is even harder if you also need to target resource and screen limited browsers of tablets and smartphones.
- HTML5 and CSS (Cascading Style Sheets) are the answer

Modern look and more



Modern look and more

- It’s easy to find thousands of css elements, hundreds kilobytes, or even megabytes of css file defining what the webpage should look like, e.g. in different browsers



Modern look and more

- Many pros:
  - Similar look in all major web browsers
  - Handling of desktop, tablet and phone screen sizes
  - Customizable layouts
  - Nicely styled navigation bars, forms, buttons, alerts, popups, etc.
- The most direct way to use Bootstrap is to simply import the *bootstrap.min.css* file in your base template.
- You can either download a copy of this file and add it to your project, or import it directly from a CDN.

Modern look and more

- You may also want to import the *bootstrap.min.js* file containing the framework's JavaScript code, so that you can also use the most advanced features.
- Flask extension called Flask-Bootstrap that provides a ready to use base template that has the Bootstrap framework installed.
  - $ flask/bin/pip install flask_bootstrap

Mc

```
{% extends 'bootstrap/base.html' %}

{% block title %}
    {% if title %}{{ title }} - Microblog{% else %}Welcome to Microblog{% endif %}
{% endblock %}

{% block navbar %}
    <nav class="navbar navbar-default">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#
bs-example-navbar-collapse-1" aria-expanded="false">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="{{ url_for('index') }}">Microblog</a>
            </div>
            <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
                <ul class="nav navbar-nav">
                    <li><a href="{{ url_for('index') }}">Home</a></li>
                    <li><a href="{{ url_for('explore') }}">Explore</a></li>
                </ul>
                <ul class="nav navbar-nav navbar-right">
                    {% if current_user.is_anonymous %}
                    <li><a href="{{ url_for('login') }}">Login</a></li>
                    {% else %}
                    <li><a href="{{ url_for('user', username=current_user.username) }}">Profile</a></li>
                    <li><a href="{{ url_for('logout') }}">Logout</a></li>
                    {% endif %}
                </ul>
            </div>
```

Modern look and more

- Rendering bootstrap forms is easy.
- Instead of having to style the form fields one by one, Flask-Bootstrap comes with a macro that accepts a Flask-WTF form object as an argument and renders the complete form using Bootstrap styles.
- register.html template now:

```
{% extends "base.html" %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block app_content %}
    <h1>Register</h1>
    <div class="row">
        <div class="col-md-4">
            {{ wtf.quick_form(form) }}
        </div>
    </div>
{% endblock %}
```

Modern look and more

- Other templates have small modifications accordingly, check and compare by yourself.
- Another problem for running Python on the server to render dates and times that are presented to users on their web browsers is that:
  - Say now we are 09:30 on May 17th, 2019.
  - Our timezone is Beijing time UTC+8.
  - Running in a Python interpreter we get:

Modern look and more

- It is pretty clear that the server must manage times that are consistent and independent of location.
- If this application grows to the point of needing several production servers in different regions around the world, we don’t want each server to write timestamps to the database in different timezones, because that would make working with these times impossible.
- Since UTC is the most used uniform timezone and is supported in the datetime class, we should use it.
- The users can then decide their preferred own timezone.

Modern look and more

- The JS web service to use is Moment.js:
  - http://momentjs.com
- Same time we can install the module:
  - $ flask/bin/pip install flask_moment
- Then add the module to __init__.py
- Then add to the base.html template:
  - {% block scripts %}
  - {{ super() }}
  - {{ moment.include_moment() }}
  - {% endblock %}

Modern look and more

- Then you can define your timestamp in ISO 8601 format, by default it's:
  - {{ year }}-{{ month }}-{{ day }}T{{ hour }}:{{ minute }}:{{ second }}{{ timezone }}
  - 2019-05-17T09:35:23Z
- You can use different time/date format:
  - moment('XXX').format('L') Long
  - moment('XXX').format('LL') LongX2
  - moment('XXX').format('LLL') LongX3
  - moment('XXX').format('LLLL') LongX4

Modern look and more

- Modify the templates, e.g. user.html:

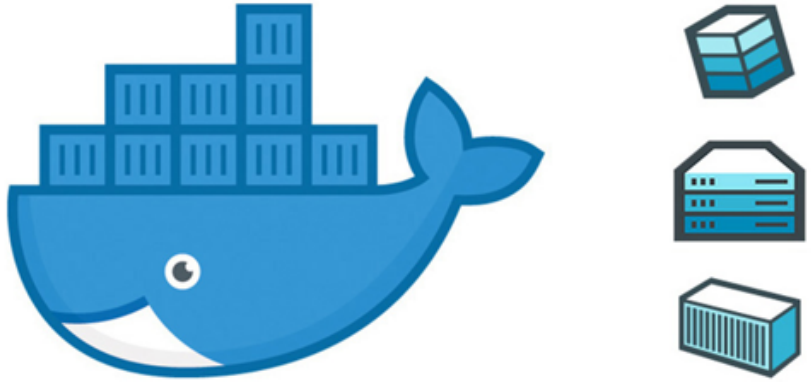
```
{% extends "base.html" %}

{% block app_content %}
    <table class="table table-hover">
        <tr>
            <td width="256px"><img src="{{ user.avatar(256) }}"></td>
            <td>
                <h1>User: {{ user.username }}</h1>
                {% if user.about_me %}<p>{{ user.about_me }}</p>{% endif %}
                {% if user.last_seen %}
                <p>Last seen on: {{ moment(user.last_seen).format('LLL') }}</p>
                {% endif %}
                <p>{{ user.followers.count() }} followers, {{ user.followed.count() }} following.</p>
                {% if user == current_user %}
                <p><a href="{{ url_for('edit_profile') }}">Edit your profile</a></p>
                {% elif not current_user.is_following(user) %}
                <p><a href="{{ url_for('follow', username=user.username) }}">Follow</a></p>
                {% else %}
                <p><a href="{{ url_for('unfollow', username=user.username) }}">Unfollow</a></p>
                {% endif %}
            </td>
        </tr>
    </table>
```

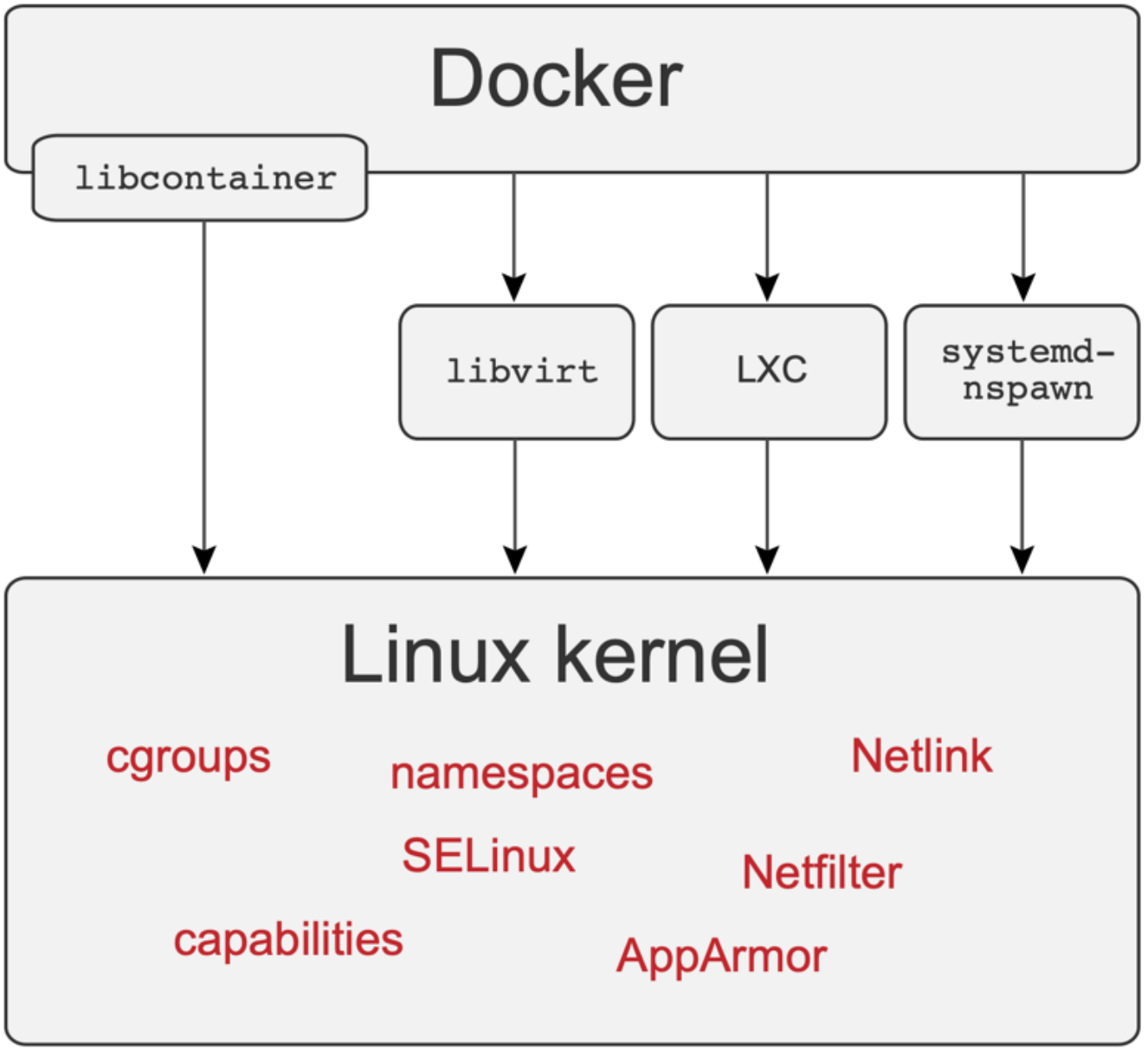
Exercise

- Run & Understand the CSS framework and time/date server/client management web service, including bootstrap, different CSS frameworks, time-date management, flask-bootstrap, and flask-moment
- Explore front-end skills, e.g. HTML5, CSS3, JavaScript, HTTP, Node JS, React/Redux, Vue, Angular JS…
- Explore different CSS frameworks, and different front-end frameworks
- Refactor the web service app with different CSS and front-end frameworks.

Deployment

- The web service must be deployed on some real environment, the best practice in industry is based on *containers*, more particularly on the Docker container platform.




Docker
libcontainer
libvirt
LXC
systemd-
nspawn
Linux kernel
cgroups
namespaces
Netlink
SELinux
Netfilter
capabilities
AppArmor

Deployment

- As a OS-level virtualization technology, Docker is most popular
- Containers are built on a lightweight virtualization technology (similar to the “virtual environment”) that allows an application, along with its dependencies and configuration to run in complete isolation.
- Containers don’t need to use a full blown virtualization solution such as virtual machines, which need a lot more resources and can sometimes have a significant performance degradation in comparison to the host.

Deployment

- A system configured as a container host can execute many containers, all of them sharing the host's kernel and direct access to the host's hardware.
- This is in contrast to virtual machines, e.g. VirtualBox, VMWare, which have to emulate a complete system, including CPU, disk, other hardware, kernel, etc.
- In spite of having to share the kernel, the level of isolation in a container is pretty high.
- A container has its own file system, and can be based on an operating system that is different than the one used by the container host.

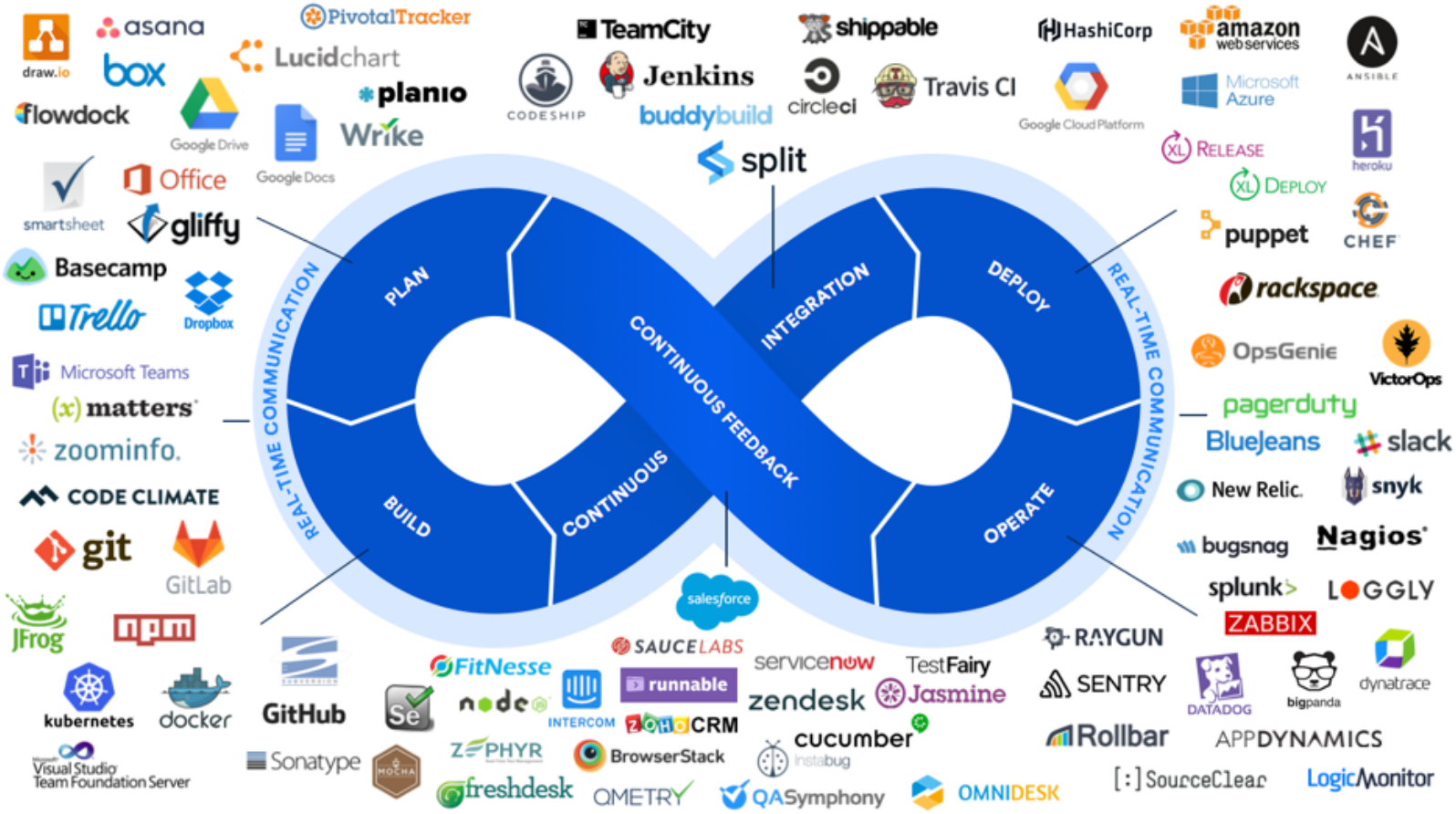
Deployment

- For example, you can run containers based on Ubuntu Linux on a Fedora host, or vice versa.
- While containers are a technology that is native to the Linux operating system, thanks to virtualization it is also possible to run Linux containers on Windows and macOS hosts.
- This allows you to test your deployments on your development system, and also incorporate containers in your development workflow if you wish to do so.

Deployment

- Because Docker containers are lightweight, a single server or VM can run 10+ containers simultaneously/easily.
- Using containers may simplify the creation of highly distributed systems by allowing multiple applications, worker tasks and other processes to run autonomously on a single physical machine or across multiple virtual machines.
- This allows the deployment of nodes to be performed as the resources become available or when more nodes are needed, allowing a PaaS(Platform)-style of deployment and scaling for systems such as Apache Cassandra, MongoDB and Riak

- Docker enables the famous "***DevOps***" terms, meaning the combination of ***software development*** (Dev) and ***information technology operations*** (Ops)



Deployment

- Docker isn't the only container platform, but by far the most popular.
- There are two editions of Docker, a free community edition (CE, or Hub) and a subscription based enterprise edition (EE, or Enterprise).
  - www.docker.com
  - https://www.docker.com/products
  - https://www.docker.com/products/docker-hub
- Docker may require Hyper-V, a virtualization technology. The installer will enable this for you if necessary.

Deployment

- You can check docker installation with:

```
Thomass-MacBook-Pro:todo-api thomaxu$ docker version
Client: Docker Engine - Community
 Version:           18.09.2
 API version:       1.39
 Go version:        go1.10.8
 Git commit:        6247962
 Built:             Sun Feb 10 04:12:39 2019
 OS/Arch:           darwin/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          18.09.2
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.10.6
  Git commit:       6247962
  Built:            Sun Feb 10 04:13:06 2019
  OS/Arch:          linux/amd64
  Experimental:     false
```

Deployment

- The application is using the default SQLite database, which is supported by a file on disk. We have to switch to a real DB for production, and make the DB container *stateless*.
- If you have a container that has application code and no data, you can throw it away and replace it with a new one without any problems, the container becomes truly disposable, which is great in terms of simplifying the deployment of upgrades.
- The big picture of our virtual servers:
  - What's your "host computer"?
  - What's your "web server"?
  - What's your "database server"?
  - What's their network/IP?

Deployment

- Create your virtual network, so that the “containers” can access each other:

```
Thomass-MacBook-Pro:20190524_deploy thomaxu$ docker network create mynetwork
ad779d0205702a752764e4456df12d00484f49a9a3912cab02394ae907da702b
```

```
Thomass-MacBook-Pro:20190524_deploy thomaxu$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
005374e79230        bridge              bridge              local
f27e6b67fea6        host                host                local
ad779d020570        mynetwork           bridge              local
e791c6465c8e        none                null                local
```

De

```
Thomass-MacBook-Pro:20190524_deploy thomaxu$ docker network inspect mynetwork
[
    {
        "Name": "mynetwork",
        "Id": "ad779d0205702a752764e4456df12d00484f49a9a3912cab02394ae907da702b",
        "Created": "2019-05-23T05:30:43.7670554Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
```

De


```
"Containers": {
    "afb5f36024a855d612613c80e2bcd11f36e6ba5c1d115dc2c40537355fd70dc5": {
        "Name": "finalsystem",
        "EndpointID": "e6ae559f6fda05e8696c0823fe44ec3f089c09503e541d38a23349e9518a3e3f",
        "MacAddress": "02:42:ac:12:00:04",
        "IPv4Address": "172.18.0.4/16",
        "IPv6Address": ""
    },
    "ce417f9063a05e0d7aa8648345902c3975030bb0e8b446798a7b12821b85c0e1": {
        "Name": "myadmin",
        "EndpointID": "a5a16c4d1d7f9bef305774c2f7a094db5d91110a884563dca574188c72c7daea",
        "MacAddress": "02:42:ac:12:00:03",
        "IPv4Address": "172.18.0.3/16",
        "IPv6Address": ""
    },
    "fa81ac91feccae55e32bf3df8d2f2d06303c07dae6f3acd0ab375384ca522853": {
        "Name": "mysql",
        "EndpointID": "d4645d30fcc7a445777ab17b4062e6eddf1021750b36cd559a71ba73a634e528",
        "MacAddress": "02:42:ac:12:00:02",
        "IPv4Address": "172.18.0.2/16",
        "IPv6Address": ""
    }
},
"Options": {},
"Labels": {}
```

Deployment

- Then you need to build an image for docker, e.g. ”install an operating system and related software package”
- The script for building the image is a “Dockerfile”
  - We can also use the existing containers directly
  - The most powerful part of Docker
  - https://hub.docker.com/search/?type=image
  - https://hub.docker.com/r/mysql/mysql-server/
  - Very simple in fact!

Deployment

- Just pull the mysql container from the docker server:
  - Join the virtual network
  - Map the port
  - Set the root password
  - Add user/password
  - Specify the version

```
Thomass-MacBook-Pro:20190524_deploy thomaxu$ docker run --name mysql --network mynetwork -p 3306:3306
 -d -e MYSQL_RANDOM_ROOT_PASSWORD=yes -e MYSQL_DATABASE=microblog -e MYSQL_USER=microblog -e MYSQL_PA
SSWORD=111111 mysql/mysql-server:5.7
fa81ac91feccae55e32bf3df8d2f2d06303c07dae6f3acd0ab375384ca522853
```

Deployment

- Now we have the DB server, next is the web server
- You can provide a bash shell script in the process as well
- We start with the “finalsystem” Python-Flask web service we built last time, based on Ubuntu 18.04 container/image:
- The “Dockerfile”:

```
Thomass-MacBook-Pro:20190524_deploy thomaxu$ ll
total 48
drwxr-xr-x  12 thomaxu  staff   384 May 23 18:53 .
drwxr-xr-x  15 thomaxu  staff   480 May 21 22:24 ..
-rw-r--r--   1 thomaxu  staff   846 May 22 14:57 Dockerfile
drwxr-xr-x   4 thomaxu  staff   128 May 21 22:28 __pycache__
drwxr-xr-x  10 thomaxu  staff   320 May 22 23:14 app
-rw-r--r--   1 thomaxu  staff   271 May 22 23:06 boot.sh
-rwxr-xr-x@  1 thomaxu  staff   620 May 23 15:12 config.py
-rwxr-xr-x@  1 thomaxu  staff   165 Jun  6  2018 finalsystem.py
drwxr-xr-x   3 thomaxu  staff    96 May 21 22:28 logs
drwxr-xr-x   8 thomaxu  staff   256 May 21 22:28 migrations
-rwxr-xr-x@  1 thomaxu  staff   568 May 21 23:17 requirements.txt
-rwxr-xr-x@  1 thomaxu  staff  3296 Jun  6  2018 tests.py
```

De

```
FROM ubuntu:18.04
RUN sed -i "s/archive.ubuntu./mirrors.aliyun./g" /etc/apt/sources.list
RUN sed -i "s/deb.debian.org/mirrors.aliyun.com/g" /etc/apt/sources.list
RUN sed -i "s/security.debian.org/mirrors.aliyun.com\/debian-security/g" /etc/apt/sources.list

FROM python:3.6-alpine

RUN adduser --gecos "Thomas Xu,RoomNumber,WorkPhone,HomePhone" --disabled-password thomas
WORKDIR /home/thomas

COPY requirements.txt requirements.txt
RUN python -m venv venv
RUN venv/bin/pip install -r requirements.txt -i https://mirrors.aliyun.com/pypi/simple/
RUN venv/bin/pip install gunicorn pymysql -i https://mirrors.aliyun.com/pypi/simple/

COPY app app
COPY migrations migrations
COPY finalsystem.py config.py boot.sh ./
RUN chmod +x boot.sh

ENV FLASK_APP finalsystem.py

RUN chown -R thomas:thomas ./
USER thomas

EXPOSE 5000
ENTRYPOINT ["./boot.sh"]
```

Deployment

- The “requirements.txt” file, specify the modules for Python-Flask to install:

```
alembic==0.9.6
blinker==1.4
certifi==2017.7.27.1
chardet==3.0.4
click==6.7
dominate==2.3.1
elasticsearch==6.1.1
Flask==1.0.2
Flask-Bootstrap==3.3.7.1
Flask-Login==0.4.0
Flask-Mail==0.9.1
Flask-Migrate==2.1.1
Flask-Moment==0.5.2
Flask-SQLAlchemy==2.3.2
Flask-WTF==0.14.2
guess_language-spirit==0.5.3
idna==2.6
itsdangerous==0.24
Jinja2==2.10
Mako==1.0.7
MarkupSafe==1.0
PyJWT==1.5.3
python-dateutil==2.6.1
python-dotenv==0.7.1
python-editor==1.0.3
pytz==2017.2
requests==2.18.4
```

Deployment

- The “config.py” file, most important part is the database configuration, e.g. URI://username:password@IP/DBname

```
import os
basedir = os.path.abspath(os.path.dirname(__file__))

class Config(object):
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'you-will-never-guess'
    SQLALCHEMY_DATABASE_URI = 'mysql+pymysql://microblog:111111@172.18.0.2/microblog'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    MAIL_SERVER = os.environ.get('MAIL_SERVER')
    MAIL_PORT = int(os.environ.get('MAIL_PORT') or 25)
    MAIL_USE_TLS = os.environ.get('MAIL_USE_TLS') is not None
    MAIL_USERNAME = os.environ.get('MAIL_USERNAME')
    MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD')
    ADMINS = ['your-email@example.com']
    POSTS_PER_PAGE = 25
```

Deployment

- Now you have the required ***docker configuration files***, as well as the ***web service files***
- Next we build the container:
  - $ docker build -t finalsystem:latest .

De

```
Thomass-MacBook-Pro:20190524_deploy thomaxu$ docker build -t finalsystem:latest .
Sending build context to Docker daemon  110.6kB
Step 1/20 : FROM ubuntu:18.04
 ---> 7698f282e524
Step 2/20 : RUN sed -i "s/archive.ubuntu./mirrors.aliyun./g" /etc/apt/sources.list
 ---> Using cache
 ---> 858551dccc1d
Step 3/20 : RUN sed -i "s/deb.debian.org/mirrors.aliyun.com/g" /etc/apt/sources.list
 ---> Using cache
 ---> 4da3c5667a3f
Step 4/20 : RUN sed -i "s/security.debian.org/mirrors.aliyun.com\/debian-security/g" /etc/apt/sources
.list
 ---> Using cache
 ---> aa2f41f3e7cc
Step 5/20 : FROM python:3.6-alpine
 ---> 35bb01a3d284
Step 6/20 : RUN adduser --gecos "Thomas Xu,RoomNumber,WorkPhone,HomePhone" --disabled-password thomas
 ---> Using cache
 ---> f4651a3862eb
Step 7/20 : WORKDIR /home/thomas
 ---> Using cache
 ---> 62043b481153
Step 8/20 : COPY requirements.txt requirements.txt
 ---> Using cache
 ---> f10fb51a2901
Step 9/20 : RUN python -m venv venv
 ---> Using cache
 ---> 7a91e60ac1bb
```

De

```
Step 10/20 : RUN venv/bin/pip install -r requirements.txt -i https://mirrors.aliyun.com/pypi/simple/
 ---> Using cache
 ---> 249815a25577
Step 11/20 : RUN venv/bin/pip install gunicorn pymysql -i https://mirrors.aliyun.com/pypi/simple/
 ---> Using cache
 ---> c0d0788dfc77
Step 12/20 : COPY app app
 ---> Using cache
 ---> 90f55d2fc0db
Step 13/20 : COPY migrations migrations
 ---> Using cache
 ---> 8b2298387be3
Step 14/20 : COPY finalsystem.py config.py boot.sh ./
 ---> 78bf8f6ed598
Step 15/20 : RUN chmod +x boot.sh
 ---> Running in ac07cd6ce5a9
Removing intermediate container ac07cd6ce5a9
 ---> fcfad9de4716
Step 16/20 : ENV FLASK_APP finalsystem.py
 ---> Running in 3d6156ec5ba0
Removing intermediate container 3d6156ec5ba0
 ---> 8730b2cbd817
Step 17/20 : RUN chown -R thomas:thomas ./
 ---> Running in 626153934c89
Removing intermediate container 626153934c89
 ---> ea5ce24fcb60
Step 18/20 : USER thomas
 ---> Running in 85da25700e21
```

Deployment

```
Step 18/20 : USER thomas
 ---> Running in 85da25700e21
Removing intermediate container 85da25700e21
 ---> b1d90138635f
Step 19/20 : EXPOSE 5000
 ---> Running in a7071dd1a23f
Removing intermediate container a7071dd1a23f
 ---> fa03b05bc2cb
Step 20/20 : ENTRYPOINT ["./boot.sh"]
 ---> Running in a50e92bcad41
Removing intermediate container a50e92bcad41
 ---> 22de848e74c3
Successfully built 22de848e74c3
Successfully tagged finalsystem:latest
```

Deployment

- Now you can start the web service (finalsystem) container:
  - The two containers should be in the same virtual network (mynetwork)
  - The port from the container (5000) is mapped to the host machine (8000)
  - The container will be removed after its stopped

```
Thomass-MacBook-Pro:20190524_deploy thomaxu$ docker run --name finalsystem --network mynetwork -d -p
8000:5000 --rm finalsystem:latest finalsystem:latest
a813cff0de8186c611a987503fd375173b0a0387ac7a931f5e1e666cf86c9a3c
```

Deployment

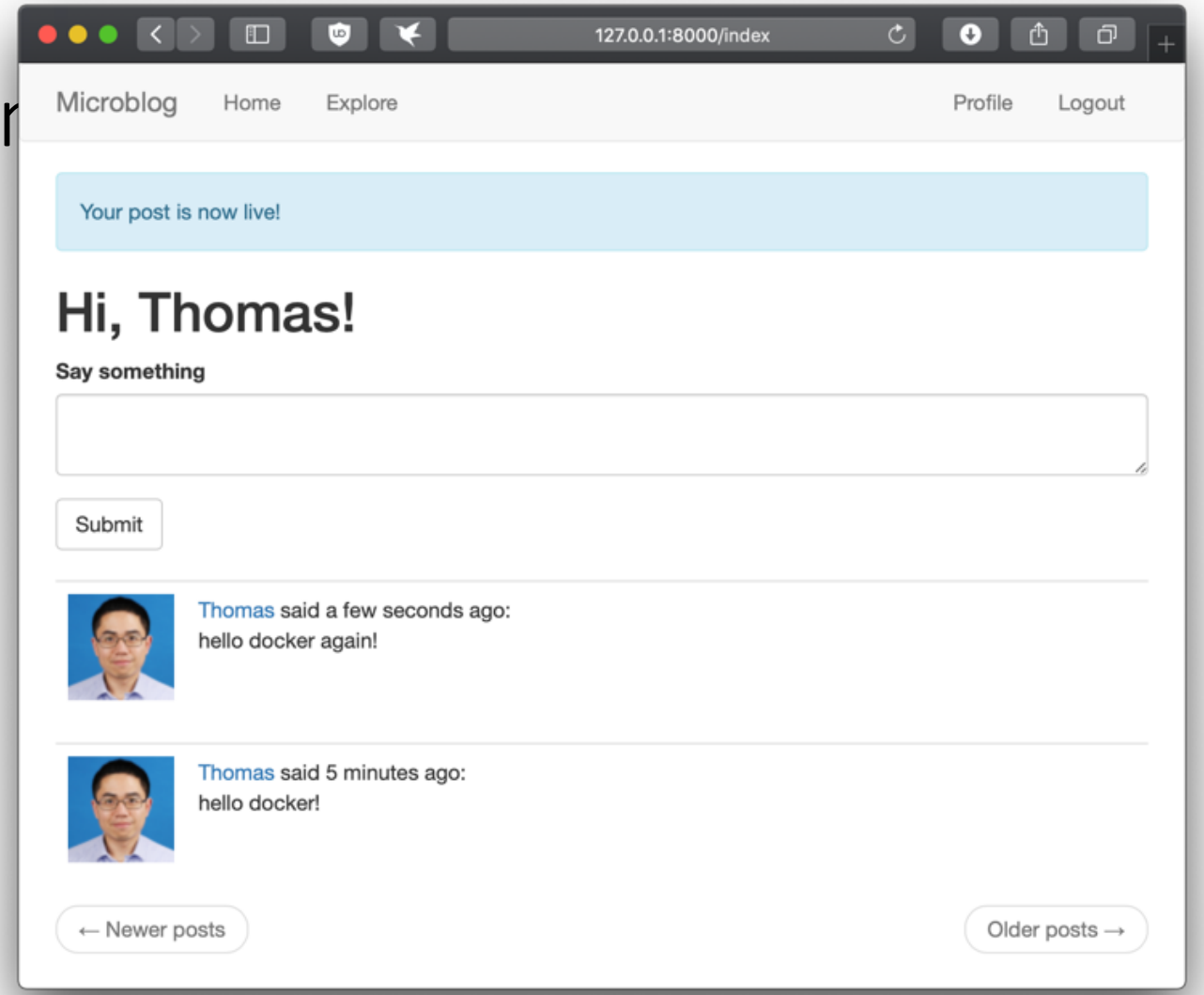
- Check your container’s status:
  - Running container: $ docker ps
  - All container: $ docker ps -a

```
Thomass-MacBook-Pro:20190524_deploy thomaxu$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS
            PORTS                                      NAMES
a813cff0de81        finalsystem:latest             "./boot.sh finalsyst…"   4 hours ago         Up 4 hours
            0.0.0.0:8000->5000/tcp                     finalsystem
ce417f9063a0        phpmyadmin/phpmyadmin          "/run.sh supervisord…"   4 hours ago         Up 4 hours
            9000/tcp, 0.0.0.0:8080->80/tcp             myadmin
fa81ac91fecc        mysql/mysql-server:5.7         "/entrypoint.sh mysq…"   4 hours ago         Up 4 hours
(healthy)   0.0.0.0:3306->3306/tcp, 33060/tcp          mysql
```

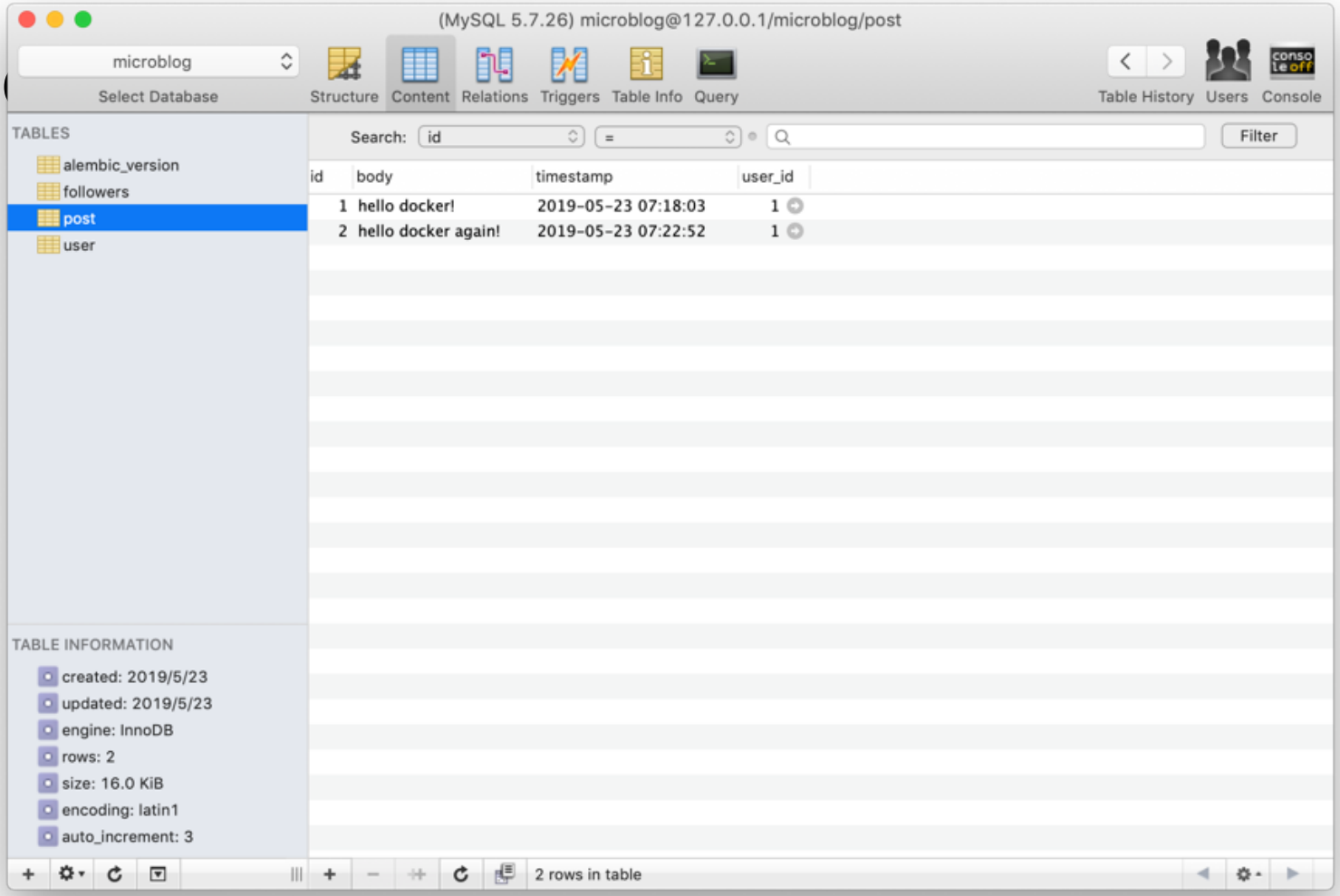
Deployment

- Stop a container:
  - $ docker stop #ID
- Remove a container:
  - $ docker remove #ID
- Check the file system and execute command inside container:
  - $ docker exec -it finalsystem ls
- Check the port info:
  - $ docker port finalsystem
- Check bash and log info inside container:
  - $ docker run -it finalsystem /bin/bash

Deployr


127.0.0.1:8000/index
Microblog
Home
Explore
Profile
Logout
Your post is now live!
Hi, Thomas!
Say something
Submit
Thomas said a few seconds ago:
hello docker again!
Thomas said 5 minutes ago:
hello docker!
← Newer posts
Older posts →

Depl


(MySQL 5.7.26) microblog@127.0.0.1/microblog/post
microblog
Select Database
Structure
Content
Relations
Triggers
Table Info
Query
Table History
Users
Console
TABLES
alembic_version
followers
post
user
Search:
id
=
Filter
id
body
timestamp
user_id
1 hello docker! 2019-05-23 07:18:03 1
2 hello docker again! 2019-05-23 07:22:52 1
TABLE INFORMATION
created: 2019/5/23
updated: 2019/5/23
engine: InnoDB
rows: 2
size: 16.0 KiB
encoding: latin1
auto_increment: 3
2 rows in table

De

```
Management Commands:
  builder     Manage builds
  config      Manage Docker configs
  container   Manage containers
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
```

De

```
import      Import the contents from a tarball to create a filesystem image
info        Display system-wide information
inspect     Return low-level information on Docker objects
kill        Kill one or more running containers
load        Load an image from a tar archive or STDIN
login       Log in to a Docker registry
logout      Log out from a Docker registry
logs        Fetch the logs of a container
pause       Pause all processes within one or more containers
port        List port mappings or a specific mapping for the container
ps          List containers
pull        Pull an image or a repository from a registry
push        Push an image or a repository to a registry
rename      Rename a container
restart     Restart one or more containers
rm          Remove one or more containers
rmi         Remove one or more images
run         Run a command in a new container
save        Save one or more images to a tar archive (streamed to STDOUT by default)
search      Search the Docker Hub for images
start       Start one or more stopped containers
stats       Display a live stream of container(s) resource usage statistics
stop        Stop one or more running containers
tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
top         Display the running processes of a container
unpause     Unpause all processes within one or more containers
update      Update configuration of one or more containers
version     Show the Docker version information
```

Deployment

- More things than deployment:
  - Cloud server
  - Domain name
  - Email server
  - Database
  - Storage
  - Valid certificate
  - Security monitoring
  - Performance monitoring
  - Backup
  - Scalability

Exercise

- Run & Understand the Docker container platform
- Make your own web service project running on docker
- Add more containers as you need, explore the official docker hub
- Explore how to manage tens+ of containers with orchestration (Kubernetes, Mesos)
- Explore other tools for DevOps automation, e.g. Ansible, Jenkins, Git
- Explore the micro-service software architecture for Docker/Kubernetes/DevOps