Docker toolbox:

- Docker for Windows an Mac OS X exist but only for 64bit enterprise / professional license
- Docker Toolbox can be used on older machines, and that is the only reason why this will be used
- Docker Toolbox can be downloaded from
  https://docs.docker.com/toolbox/toolbox_install_windows/
  https://docs.docker.com/toolbox/toolbox_install_mac/

- Toolbox includes these Docker tools:
  - Docker Machine for running docker-machine commands
  - Docker Engine for running the docker commands
  - Docker Compose for running the docker-compose commands
  - Kitematic, the Docker GUI
  - A shell preconfigured for a Docker command-line environment
  - Oracle VirtualBox

Boot2Docker

- Boot2Docker is a lightweight Linux distribution made specifically to run Docker containers.
- It runs completely from RAM, is a ~45MB download and boots quickly.
- Using `$ docker-machine create` command will:
    - download the lightweight Linux distribution (boot2docker)
    - create and start a VirtualBox VM with Docker running
    - Create a SSH key
    - Obtain an IP address for machine

```
$ docker-machine create --driver virtualbox manager1
```

- In the command above we are using the --driver flag with virtualbox to create our machine
- **manager1** is the name of the machine which will be created.
- Once machine started we will see “Docker is up and running”
- To view all machines information use `$ docker-machine ls`

https://github.com/boot2docker/boot2docker

Docker Swarm

- Swarm is Docker's own container orchestration tool.
- It uses the standard Docker API and networking, making it easy to drop into an environment where you're already working with Docker containers.
- Docker Swarm is a clustering and scheduling tool for Docker containers.
- With Swarm, IT administrators and developers can establish and manage a cluster of Docker nodes as a single virtual system.
- Swarm uses scheduling capabilities to ensure there are sufficient resources for distributed containers.
- Swarm assigns containers to underlying nodes and optimizes resources by automatically scheduling container workloads to run on the most appropriate host.
- Creating worker machines is done by using

```
$ docker-machine create --driver virtualbox worker1
```

To see machines, we can use:

```
$ docker-machine ls
```

To run existing machines, we can use:

```
$ docker-machine start <machine name>
```

To stop existing machines, we can use:

```
$ docker-machine stop <machine name>
```

Creating a swarm:

- SSH into our manager machine:

```
$ docker-machine ssh <machine name>
```

- Initialize the swarm:

```
$ docker swarm init --advertise-addr <MANAGER_IP>
```

- To check swarm status:

```
$ docker node ls
```

- To find out the **join** command for a **worker**, use:

```
$ docker swarm join-token worker
```

- To find out the **join** command for a **manager**, use:

```
$ docker swarm join-token manager
```

- To join into the swarm we will need to perform a few steps:

1. open a new **Docker Quick start terminal**
2. SSH into one of the worker machines:
   a.
```
$ docker-machine ssh worker1
```
3. Copy the “Docker swarm join..” from earlier **excluding the port !!!**
4. And paste it inside the Worker1 terminal, and you will see the following output:

```
This node joined a swarm as a worker.
```

- To join another worker machine following steps 1-4.
- To validate all nodes are ready, use command:

```
$ docker node ls
```

- Now that we have our swarm up and running, it is time to schedule our containers on it.
- The beauty of the orchestration layer is that we are going to focus on the app and not worry about where the application is going to run.
- To start a service, we will need to have the following:
    - What is the Docker image that you want to run. In our case, we will run the standard **Nginx** image that is officially available from the Docker hub.
    - We will expose our service on port 80.
    - We can specify the number of containers (instances) to launch. This is specified via the replicas parameter – in our case it will be 2 (workers number)
    - Give a name for our service.

```
$ docker service create --replicas 2 -p 80:80 --name web nginx
```

    - Once we run it, swarm will create the service and prepare the replicas.
- On the host, open any browser and type one of the workers IP address.
- If everything worked well, you should see Nginx page:

Scaling:

- Scaling is done via the Docker service scale command.
- Scaling is used when we want to easily let our service to run on multiple instances
- To increase the amount of containers to 5 use:

```
$ docker service scale web=5
```

Maintenance:

- We can inspect the nodes anytime via the Docker node inspect command.
- For example if we are already on the node we want to check, we can use the name self for the node.

```
$ docker node inspect self
```

- We can also check up another node, using node name. For e.g.

```
$ docker node inspect worker1
```

- To remove the service, simply use:

```
$ docker service rm web
```

- In case you have an updated Docker image to roll out to the nodes, all you need to do is use the service update command.

```
$ docker service update --image <imagename>:<version> web
```

Draining nodes:

- Sometimes, we have to bring the node down for some maintenance reasons.
- This meant by setting the Availability to **Drain mode**.
- When we give that command, the Manager will stop tasks running on that node and launches the replicas on other nodes with **ACTIVE** availability.
- So what we are expecting is that the Manager will bring the 2 containers running on worker2 and schedule them on the other nodes (manager1 or worker1).
- This is done by updating the node setting availability to “drain”.

```
$ docker node update --availability drain worker2
```

- If we will run the status command, we will see the following:

```
$ docker service ps web
```

- We can see that the containers on drained node are being rescheduled on other workers.
- This is required because we have asked for 5 replicas to be running in an earlier scaling exercise.

https://docs.docker.com/engine/swarm/

YAML:

- YAML is a human-readable data serialization language.
- It is commonly used for configuration files, but could be used in many applications where data is being stored (e.g. debugging output) or transmitted (e.g. document headers).
- YAML targets many of the same communications applications as XML but has a minimal syntax which intentionally breaks compatibility.
- It uses both Python-style indentation to indicate nesting, and a more compact format that uses [] for lists and {} for maps.
- This files can have either the .yml or .yaml extenstions.
- yml files can sometimes be annoying to write because:
    - YAML is case sensitive.
    - YAML does not allow the use of tabs.
    - Spaces are used instead as tabs are not universally supported.
- If you feel stuck use a YAML validator which can be found freely online, e.g: http://www.yamllint.com/

http://yaml.org/

Docker compose:

- Compose is a tool for defining and running multi-container Docker applications.
- With Compose, you use a Compose file to configure your application's services.
- Then, using a single command, you create and start all the services from your configuration.
- Compose is great for development, testing, and staging environments, as well as CI workflows.
- Using Compose is basically a three-step process.
    - Define your app's environment with a **Dockerfile** so it can be reproduced anywhere.
    - Define the services that make up your app in **docker-compose.yml** so they can be run together in an isolated environment.
    - Lastly, run **docker-compose up** and Compose will start and run your entire app.
- Volumes key can be added to the compose file (yml) to mount the project directory (current directory) on the host to /code inside the container, allowing us to modify the code on the fly, without having to rebuild the image.

https://docs.docker.com/compose/

Kitematic:

- Kitematic is an open source project built to simplify and streamline using Docker on a Mac or PC.
- Kitematic automates the Docker installation and setup process and provides an intuitive graphical user interface (GUI) for running Docker containers.
- Kitematic integrates with Docker Machine to provision a VirtualBox VM and install the Docker Engine locally on your machine.
- Kitematic is a part of the Docker toolbox bundle.

https://kitematic.com/