Spark Session: ft_server

updated: 18/03/2021

Project description:

> Discover Docker and set up your first web server.

Topics

1. Virtual Machines
2. Containers
3. Docker Tutorial

Virtual Machines

1. Traditionally, organizations would run applications on physical servers. This was both expensive to maintain and did not scale well.
 Virtualization was a solution to this. What is virtualization and what issues does it solve? (15 mins)
2. What is a virtual machine (VM)? What is a hypervisor? (5 mins)

Containers

1. VMs are, however, still resource-heavy as each VM is its own full machine with its own resource needs.
 Containerization provides a more lightweight alternative to that. What are containers? How are they similar yet different from VMs? (20 mins)

2. Where does Docker fit into this picture? Why would you use container technology like Docker? (10 mins)
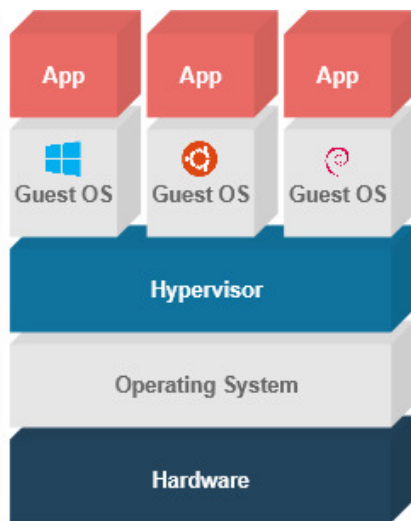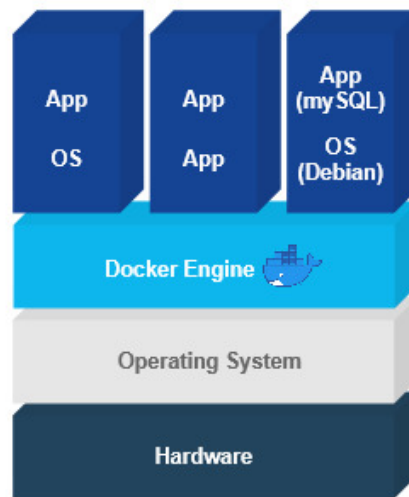
Break (5 mins)

Docker Tutorial

Set Up

For this session, we'll be using **Play with Docker** to follow Docker's own tutorial! Here's how to set it up: (10 mins)

1. If you don't have it already, register for a Docker ID: https://docs.docker.com/docker-id/
2. Go to https://labs.play-with-docker.com/ and log in.
3. Type the following command in the black PWD terminal:
 `docker run -dp 80:80 docker/getting-started:pwd` *(Copy-paste hint: Shift+Insert)*
4. Wait for it to start the container and click the **port 80** badge above the terminal.

Introduction to Docker

Tutorial page: Getting Started

1. Let's break down that `docker run -d -p 80:80 docker/getting-started:pwd` command you just executed. What does each part mean? (10 mins)
2. Why port 80? What is HTTP? (15 mins)
3. We've talked about what containers are. But what is a container image? (10 mins)

Break (5 mins)

Building an Image and Running a Container

Tutorial page: Our Application

Let's start learning how to use Docker by getting a little to-do list page up and running! (20 mins)

1. Follow the instructions under **"Getting our App into PWD"** to get the application source code into your Play with Docker environment.
 When you're done, do you see the files in `app/` as shown in the tutorial?
2. Now create a Dockerfile in the PWD environment with the code from the tutorial (you can use vim)
   - What is a Dockerfile?
3. Build the container image using `docker build -t docker-101 .`
   - What happened when you ran that command?
4. Start up the container using `docker run -dp 3000:3000 docker-101`
   - Recall what we discussed earlier. What does each component of this command mean?
5. Click on the port **3000** badge at the top of your PWD interface. Do you see your lovely empty to-do list? Try adding an item or two!

Making Changes in our Container

Tutorial page: Updating our App

Let's see how we can implement a change in our image. (15 mins)

1. Change the code on line **56** in `app.js` as specified in the tutorial.
2. Let's try building this newest version of the image with
 `docker build -t docker-101 .`
3. What happens if you execute `docker run -dp 3000:3000 docker-101` like last time?
4. So we first need to stop our old container that still has the outdated code.
   - How can we see the currently running containers?
   - How can we stop a specific container?
 (Hint: do you have to type the entire container ID?)
5. Now that we've stopped the old container, we have to remove it. Execute the `docker rm` command shown in the tutorial.
   - There's also a way to stop and remove a container using a single command. What is it?
6. Try running your updated application again. Do you see your updated text?
   - But notice that all of the items you added to your previous list are gone! Let's address that next.

Persisting the Data

Tutorial page: Persisting our DB

Now let's learn about why our application data is wiped each time we launch it and how to prevent that. (30 mins)

1. Each container has its own filesystem and gets their own space to create and manipulate these files.\ Follow the instructions on the tutorial page to see how a file created in one container isn't available in another.
   - Look at the first `docker run` command. What does each part do? Run it to start the ubuntu container and create the `data.txt` file.
   - Run the `docker exec` command shown, using the container ID *(hint: `docker ps`)*. Do you see a random number?

- - - **Extra step:** now try running `docker exec <container-id> ls` . Do you see the data.txt file?
    - Let's run a second container using the same image. Execute the `docker run` command shown. Notice that there's no `data.txt` file here.
    - You can remove the first container now using `docker rm -f <container-id>` .
2. What are container volumes?
3. Let's try creating and using a **named volume**. Do you know what that is? Run the `docker volume create` command shown.
4. Start your `docker-101` container again, but this time specifying a volume mount with the `-v` flag.
    - What does the `docker run` command shown do exactly?
5. Once your to-do list is up, try adding a few items to the list.
6. Now remove the container using `docker rm -f` .
7. Start a new container using the same `docker run` command you ran in step 4.
 If you open the app again, do you see the list items you added earlier?
8. Now let's see where Docker stores that data when you use a named volume.\ Run the `docker volume inspect` command shown in the tutorial.
    - What is a mountpoint?

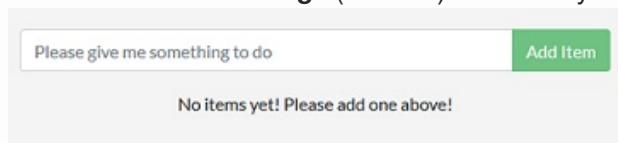Congratulations, now you know how to use volumes in Docker to preserve your application data!
Named volumes are one of the main types of volumes in Docker, **bind mounts** being the other.\
Feel free to try out the tutorial section on bind mounts on your own to see how you can rapidly implement changes in your source code by specifying the mountpoint.

Docker Environment Variables

Docker's `ENV` instruction lets you use **environment variables** to set and modify configuration information to running containers. This way, you don't have to manually edit your configuration files!

1. First off, how would you declare an environment variable in your Dockerfile? (5 mins)
2. Now let's try to change the "New Item" placeholder text in your to-do list! Using environment variables, we can dynamically change what text is displayed. (15 mins)\ Go into your Dockerfile and add the following things:
    - declare an environment variable called `NEW` and give it a default value ("What would you like to do today?", for example)
    - replace the last `CMD` instruction with this: *(make sure the instruction is on one line)*
      `CMD sed -i "99 s/New Item/${NEW}/g" /app/src/static/js/app.js && node /app/src/index.js`
3. Build your image with `docker build -t docker-101 .` and run it with
 `docker run -dp 3000:3000 docker-101` . Is the New Item text now replaced with the default value you gave `$NEW` ? (5 mins)
4. Now stop the container. Say we want to change the empty message to something else, "Please give me something to do" for example. How can we do that through `docker run` , **without changing the Dockerfile or rebuilding**? (15 mins)
 Did you manage to change your placeholder text?

5. Let's go back to that `CMD sed` instruction I helpfully provided earlier. (15 mins)

- What does each part of the `sed` command mean?
- Why did I use `CMD` and not `RUN`?
- Can you have multiple `CMD` instructions in a single Dockerfile?

Now you know how to use environment variables to allow for more dynamic environment configuration.\
This will come in handy for certain parts of the project that require toggling settings or passing credentials.