NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Cloud Native Application - Containerization I

Cloud Infrastructure Engineering

Nanyang Technological University
& Skills Union - 2022/2023

Course Content

- Self Study Check In
- Containerization Concept
- Use case of Containerization
- Create Dockerfile
- Create Image and Run Container

Activity

Instructor

- Ask to use AWS use single region for all learner for easier monitoring

Q1: What are containers?

Q2: Which of the following about Docker is false?

A - The Dockerfile describes how a container image should be created.

B - The Dockerfile is able to launch an instance of container without creating images.

C - Docker Hub is a repository of docker images

D - Docker container is compatible with deployment orchestration tools like Kubernetes.

Q3: Which of the following does not describe the benefits of containerization?

A - Separation of responsibility / benefits of a managed service

B - Workload portability

C - Application Isolation

D - Lightweight

Containerization

What is containerization?

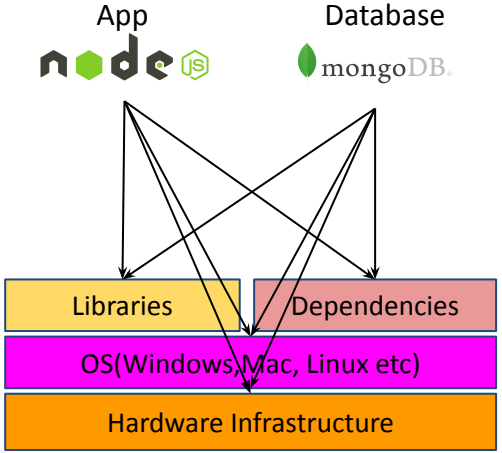
Containerization is a software deployment process that bundles an application's code with all the files and libraries it needs to run on any infrastructure. Traditionally, to run any application on your computer, you had to install the version that matched your machine's operating system.

For example, you needed to install the Windows version of a software package on a Windows machine. However, with containerization, you can create a single software package, or container, that runs on all types of devices and operating systems.

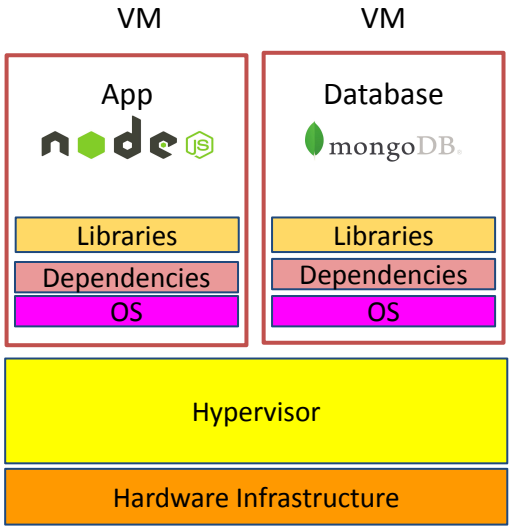
Containerization is a form of virtualization where applications run in isolated user spaces, called containers, while using the same shared operating system (OS). One of the benefits of containerization is that a container is essentially a fully packaged and portable computing environment.

Architectural Overview

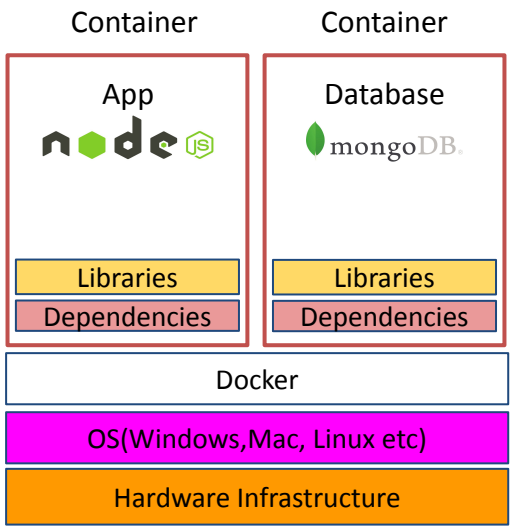
Single server setup / local set up



Virtual Machine(VM) set up



Container set up



What are the benefits of containerization?

Developers use containerization to build and deploy modern applications because of the following advantages.

- Portability
- Scalability
- Fault tolerance
- Agility

Portability

Software developers use containerization to deploy applications in multiple environments without rewriting the program code. They build an application once and deploy it on multiple operating systems. For example, they run the same containers on Linux and Windows operating systems. Developers also upgrade legacy application code to modern versions using containers for deployment

Scalability

Containers are lightweight software components that run efficiently. For example, a virtual machine can launch a containerized application faster because it doesn't need to boot an operating system. Therefore, software developers can easily add multiple containers for different applications on a single machine. The container cluster uses computing resources from the same shared operating system, but one container doesn't interfere with the operation of other containers.

Fault tolerance

Software development teams use containers to build fault-tolerant applications. They use multiple containers to run microservices on the cloud. Because containerized microservices operate in isolated user spaces, a single faulty container doesn't affect the other containers. This increases the resilience and availability of the application.

Agility

Containerized applications run in isolated computing environments. Software developers can troubleshoot and change the application code without interfering with the operating system, hardware, or other application services. They can shorten software release cycles and work on updates quickly with the container model.

Docker

What is Docker?

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run.

When to use Docker

MICROSERVICES

Build and scale distributed application architectures by taking advantage of standardized code deployments using Docker containers.

CONTINUOUS INTEGRATION & DELIVERY

Accelerate application delivery by standardizing environments and removing conflicts between language stacks and versions.

DATA PROCESSING

Provide big data processing as a service. Package data and analytics packages into portable containers that can be executed by non-technical users.

CONTAINERS AS A SERVICE

Build and ship distributed applications with content and infrastructure that is IT-managed and secured.

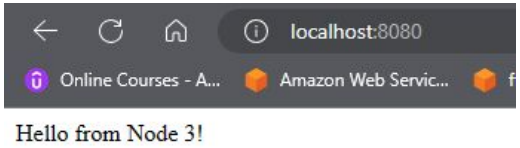
Break for 10-15 mins

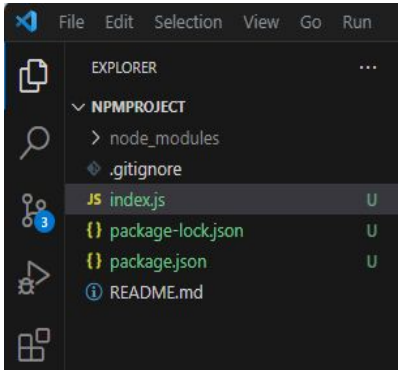
Activity

Dockerize our “Hello from Node!” application from the previous lesson.

Activity(If you missed previous lesson)

1. Create a new public git repository with README.md and .gitignore (Set to Node)
2. Clone your repository
3. cd into your repository locally
4. run npm init (Leave all to default -> press enter all the way)
5. run code . in terminal to open VSCode
6. Create a index.js file and paste the following content into it(jaezeu/npmproject (github.com))
7. Go back to **terminal** and run npm install express
8. Run node index.js in terminal
9. Verify that you see `Running on http://0.0.0.0:8080`
10. Open your browser and type localhost:8080

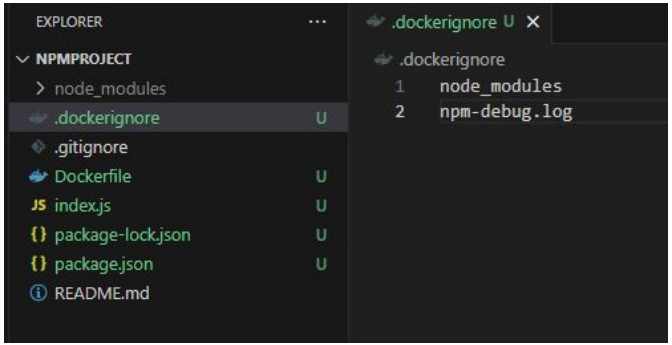




Activity

Instructor demonstrate on how to create Dockerfile file.

1. Git clone the repository you worked on in the previous lesson to your local computer(if you do not have)
2. Go to “index.js” line 13 and verify that HOST is set to “0.0.0.0”
3. Create a file named “Dockerfile” and fill content with the following (next slide)
4. Create `.dockerignore` file and add the following:



Activity

Instructor explain the Dockerfile file

```
FROM node:16-alpine

WORKDIR /my-app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 8080

CMD ["node", "index.js"]
```

Activity

Instructor explain the .dockerignore file

```
node_modules

npm-debug.log
```

Activity - Explanation about the code

Keyword	Purpose
`FROM`	The FROM instruction initializes a new build stage and sets the Base Image for subsequent instructions.
`WORKDIR`	The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it.
`COPY`	The COPY instruction copies files from a local source location to a destination.
`RUN`	The RUN instruction will execute any commands in a new layer on top of the current image and commit the results.
`EXPOSE`	The EXPOSE instruction exposes a particular port inside a Docker container
`CMD`	There can only be one CMD instruction in a Dockerfile. If you list more than one CMD then only the last CMD will take effect. The main purpose of a CMD is to provide defaults for an executing container.

Break for 5-10 mins

Create Image and Run Container

Activity

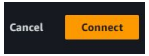
Instructor demonstrate on how to connect to EC2 and set up Docker

Step 1: Login to https://255945442255.signin.aws.amazon.com/console/ and once logged in, Type EC2 in the search bar on top

Step 2: Look for the Instance that is named after your name

Step 3: Right click on the Instance and select Connect.

Step 4: In the next page, Verify that the username is “ec2-user” and click Connect



Step 5: Once you are logged in to the terminal, type the following commands:

```
sudo su

yum update -y

yum install git -y

yum install docker -y

systemctl start docker

systemctl status docker
```

Activity

Instructor demonstrate on how to create Image and Run Container

Step 1: Launch EC2 instance connect

Step 2: git clone your “hello from node” repo into the EC2

Step 3: cd to your repo

Step 4: Run docker build -t mysampleapp . to build an image with the repository name mysampleapp

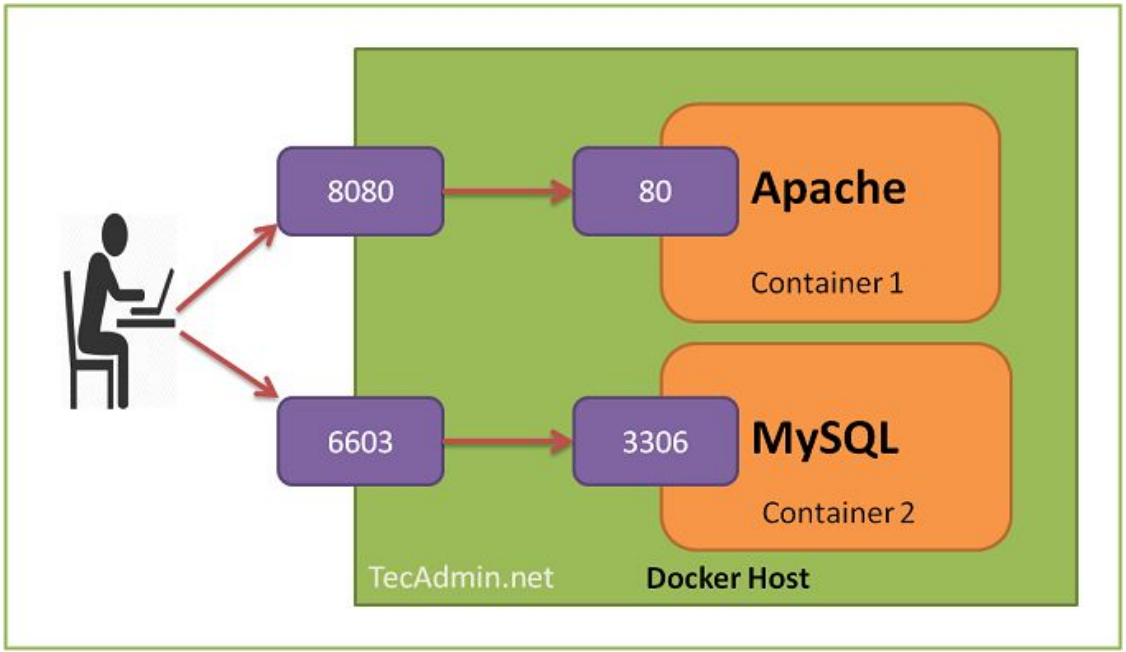
Step 4: Run docker image ls to list all images. You should see a table with IMAGE ID as one of the column. You will need that to run image as container in the next step.

Step 5: Run docker run -d -p 8080:8080 mysampleapp:latest to start the container

Step 6: Run the following command , curl http://0.0.0.0:8080

Step 7: After Step 6, you should see a “Hello from Node!” response displayed in the terminal

Activity


8080
80
Apache
Container 1
6603
3306
MySQL
Container 2
TecAdmin.net
Docker Host

Most Commands for Docker

Commands	Description
`docker image ls`	List all images that are being created
`docker image rm <image id>`	Remove specific image by ID
`docker build <options>`	Build an image from the Dockerfile.
`docker pull`	Pulls a image from dockerhub and stores it in the host
`docker ps`	List all running containers
`docker ps -a`	List all stopped and running containers
`docker stop <container id>`	Stop a running container
`docker rm <container id>`	Remove a container
`docker logs <container id>`	View logs of a particular container

[Dockerizing a Node.js web app | Node.js (nodejs.org)]

Container Image Names

Convention

{registry}/{repository}:{tag}

Examples

docker.io/httpd:2.4.57 <- Each version of an application image is defined by a different tag

docker.io/httpd:2.4.56

docker.io/myuser/myapplication:v1.0-alpine <- Sometimes there are subpaths (e.g. personal repository on dockerhub)

xxxx.dkr.ecr.ap-southeast-1.amazonaws.com/node-hello-world:latest

xxxx.dkr.ecr.ap-southeast-1.amazonaws.com/node-hello-world <- Sometimes `latest` tag is implied if not stated. Best to avoid such usage

Bonus Example

ubuntu@sha256:26c68657ccce2cb0a31b330cb0be2b5e108d467f641c62e13ab40cbec258c68d

Questions?

Activity

Learner:

- Clean up AWS.
- Remove/delete/terminate all service/ resources that you created.

Instructor

- Clean up AWS.
- Remove/delete/terminate all service/ resources that you created.
- Check the AWS account after learner clean up.

END

Activity

Take 5-10 Mins for Learners create new repository.

Activity

Instructor demonstrate on how to create Dockerfile file with the sample-app.

Sample app is available here: https://github.com/su-ntu-ctp/6m-cloud-3.4-cloud-native-application-containerization-i/tree/main/sample-app

1. Create new repository
2. Pull the repository to local computer
3. Fill the dockerfile file content with the following (next slide)
4. Create `.dockerignore` file and add `node_modules` to the content.

Activity

Instructor explain the Dockerfile file

```
FROM node:16-alpine

WORKDIR /app

ENV PORT=3000

COPY ["package.json", "package-lock.json*", "./"]

RUN npm install

COPY . .

CMD ["npm", "start"]
```