- Containerization
  It plays an important role in microservices and DevOps. It makes DevOps very easy. In the deployment procedure, we need to run different applications/microservices. They are developed from different languages like Java, Python and C, which require us to setup different deployment environment for them. We need different configuration and scripts, which is time-consuming and human error prone, and this makes maintaining these environments a big headache. Now, what we need is docker image. Docker enables standardization of how we package and deploy the application irrespective of the language or framework we used. Once you use containers, your infrastructure as code becomes really really simple.

  consistent deployment automation across different environments and different technologies. If all the microservices are built on containers, we can use standard procedures for executing, monitoring and scaling. We can make use of orchestration tools as well.

- Kubernete

  1. standardized application packaging: same packaging for all types of applications
  2. multi platform support: local machine, cloud
  3. light-weight and isolation: compared to VM

  image command

  - dock images
  - dock pull + imagename
  - docker search + imagename
  - docker image history + imageID: see layers inside a docker image
  - docker image inspect + imageID: inspect image details
  - docker image remove + imagename (docker rmi + iamgename)
  - docker rmi -f $(docker images -aq): remove all images

  container image

  - docker container rm + containerID (docker rm + containID)
  - docker container ls -a (docker ps -a)
  - docker container stop + containerID
  - docker container pause/unpause + containerID
  - docker rm -vf $(docker ps -aq), (docker container prune) : remove all containers
  - docker container logs -f: check running logs
  - docker container inspect + containerID

  system and stats

  - docker system prune -a: remove unused resource (stopped containers, build cache...)
  - docker stats + containerID

  network

  - docker network ls
  - docker network inspect +networkName

  compose

  - docker-compose up/down
  - docker-compose events
  - docker-compose config: useful to debug the .yml file
  - docker-compose images
  - docker-compose ps
  - docker-compose stop/kill

CMD vs. Entrypoint

  – CMD can be overrided when the image is run

- **docker compose**
  When microservices are launched, they are launched into the default bridge network. They cannot directly talk to each other, and to help them to talk with each other we need to establish a link between them.

  When we have multi-containers and want them to talk with each other, we can use docker compose. We can specify the configuration with multiple microservices in a simple YAML file and then launch up the entire configuration containing different containers with just one command.

- **GitHub action**
  We are living in a world of containerization We build Python artifacts, we build docker image and push that image to docker repository

  We use GitHub Actions to setup CI/CD pipelines. Compared to other tools, GitHub Actions is very straightforward. We don't need to manually configure and setup CI/CD, and we don't need to buy hardare. It has very powerful community, The GitHub Marketplace provides all kinds of free actions for us to buid the workflows.

  CI means when the code get changed. We should make sure all the changes work with the rest of the code when they are merged by undergoing automated testing. The testing include unit and integration tests to ensure that code iss functional. A CD pipeline goes one step further and deploys the built code into production.