Dockers & Kubernetes

PRACE Winter School 2021 @ Tel-Aviv/Israel

Mordechai Botrashvily (mordechai.botrashvily at gmail)
Company for Advanced Supercomputing Solutions LTD

09/12/2021

Agenda

- Welcome
- Motivation & Goals
- Before We Begin
- Parts
    1. Intro to Dockers & Containers
    2. Intro to Kubernetes (k8s)
- Wrap-up & Advanced Topics

Welcome

- 3rd PRACE Winter School held in Israel

- About Me
  - Computer science background
  - Physics / Geophysics / Applied Mathematics
  - Involved with GPU/HPC/Cloud infrastructures and computing

  - Participated in PRACE Summer of HPC 2013 (☺)

Motivation & Goals

Trends

- Software development is changing rapidly
- Researchers need to keep up with latest technologies to be competitive and productive

- Today’s compute tasks are getting more complex and heavier than before

- Terms like dockers/containers are very common
- Yet not easily accessible

Typical Scenario

- We developed an AI system (python + Torch + GPU + additional libraries)
- Our system is running on *Linux Ubuntu 18.04*
- Code is published as open-source on GitHub

- New versions are released every quarter
- Not necessarily compatible with previous ones

Difficulties

- How to let others run our codebase with minimal effort?
- Just give a list of installations?
- What if they have Windows or Macintosh?
- How to address other dependencies and versions?
- Or running on a given cluster?
  - Without ability to modify installed libraries

Dockers for Help

- Shipping complex software became an issue

- Dockers are meant to solve this
  - Create a single image that describes our environment
  - Can run everywhere
  - Easily modified and managed

- **Caveat:** additional overhead (virtualization + larger file sizes)

Kubernetes

- A complementary solution to dockers
- Cluster scale runtime environment for containers
- Extensive API
- Access to cloud resources (storage etc.)

- Very sophisticated/complex infrastructure

Goals

1. Basic familiarity with dockers & containers
   - CMD = **C**onsume, **m**odify, **d**eploy
   - Access local or cloud storage resources
   - Use container registry

2. Basic familiarity with Kubernetes
   - Using managed cloud services
   - Run dockers/containers in the cloud
   - Submitting compute jobs with python

Before We Begin

Preliminaries / Audience Assumptions

- OK with using Linux terminal + Ubuntu commands
- Python programming skills
- Beginners familiarity with AWS (services, console, cli)

- Workshop is OS independent (Windows / Linux / Mac)
- Make sure your browser is up-to-date (Edge, Firefox, Chrome)

Workshop Format

- Me:
  - Technical overview
  - Live demonstrations

- You:
  - Self paced tasks
  - Increasing level of complexity

Workshop Cloud Resources

- Using AWS as cloud provider

- Amazon was kind to provide free credit to resources
- Thank you ☺

- * All tasks can be accomplished with other cloud providers
  - Microsoft Azure, Google etc.

* Important Notes

- Many possibilities to run images (docker not mandatory)

- There are alternatives to Kubernetes (Singularity etc.)

- This workshop was designed to be practical and hands-on
- Around common technologies and tools
- A little informal

Part 1 – Intro to Dockers & Containers

Docker - An Introduction

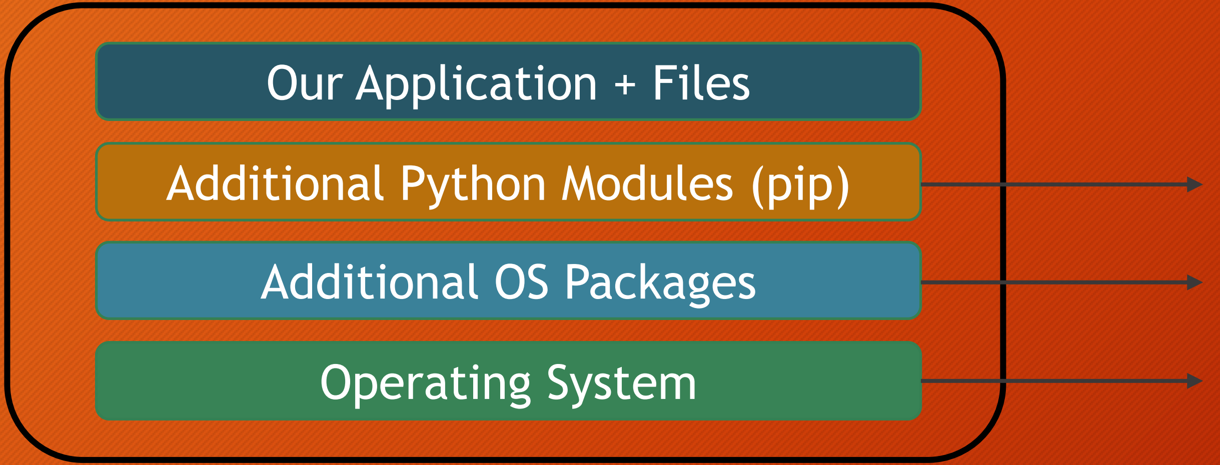
- Wraps together many resources in a layered approach
  - Operating system
  - Additional libraries and installations
  - Our application files and configuration

- Resulting in a single runnable image/file, describes everything

- For our purposes, dockers and containers are synonyms
- Checkout: https://docs.docker.com/

Docker / Container Diagram

- Rough docker image file layout



Our Application + Files

Additional Python Modules (pip)

Additional OS Packages

Operating System

e.g. sci/numpy

e.g. python3, OpenCV, drivers

e.g. Ubuntu 18.04

Live Demonstration – Docker Usage

- Using **Docker Desktop**
- Consuming basic Ubuntu 20.04 image
- Public repository/hub

Introduction to AWS Cloud9

- Web based IDE, integrated with AWS cloud
- Achieves uniformity and removes setup overhead
- Can be used to share environments between team members

- **For us:**
  - Contains all the necessary tools pre-installed
  - As if running on a local computer

Joint Demonstration – Docker Usage

- Similar to the previous demo, but on AWS with Cloud9
- Perform login and image run

- Using **Cloud9** (instead of **Docker Desktop**)
- Consuming basic Ubuntu 20.04 image
- Public repository/hub

Joint Demonstration – Docker Usage (#2)

- Type in terminal:

```
# docker run -it ubuntu:20.04 /bin/bash
```

- Follow similar output and try basic commands

- ***docker*** is the main cli for performing docker/container related tasks on the local machine

Short Summary

- Used docker ***run*** sub-command
- Ubuntu image is bare and very basic

- How to create a customized/reproducible image?

Composing a Docker – *Dockerfile*

- *Dockerfile* is a standard way to describe image contents

```
FROM ubuntu:20.04
RUN apt update
RUN apt install python3 python3-pip ipython3
```

- The **BOLD** directives above instruct docker what to perform
- Followed by specific instructions
  - Take a bare Ubuntu 20.04 image as basis
  - Obtain package information and install python3 packages

Docker – Task #1

- Build an Ubuntu 20.04 image with python3 and numpy
- Output random results to a file

- More specific details in the attached file

Docker – Task #1 – Fast Observations

- Image built by default without a meaningful identifier
    - Use *build* with *-t* (tagging) and check with *docker image list*
- Container data is volatile by default
    - Next we'll explore ways of sharing data between a container and the outside world
- Sidenote: each ***RUN*** directive describes an independent docker layer = good for caching

Sharing Data with Containers

- Main methods to explore:
    1. Copy commands against a running container
        - Very good for debugging
        - Works with cloud resources as well
    2. Mount a **local** folder inside a container
        - Great for development/debugging
        - Works only when running local containers
    3. Persistent volumes (not covered)
        - A special docker blob/disk, can re-attach to containers

Docker Copy

- A special *cp* sub-command
- Move files between local file-system and a running container
- Very easy to use

- Check here: docker cp | Docker Documentation

Docker Bind Mounts

- A method similar to Linux mount
- Additional parameter when starting a new container

```
# docker run -it --mount
type=bind,source=<ext_folder>,target=/mnt ubuntu
```

- Our external folder will be available as */mnt* inside the container

- Check here: Use bind mounts | Docker Documentation

Docker – Task #2

- Use the resulting image of Task #1
- Output random results to a file

- Use *cp* command to copy the output file to the Cloud9 file-system
- Use bind mounts to share data directly with container

- More specific details in the attached file

Docker – Task #2 – Conclusions

- Both methods work well
- Which one is most preferred?

Towards a Docker “Service”

- Up until now - using docker instances interactively

- Most workloads assume headless (standalone) operation
  - How to prepare for it?

- Introducing additional *Dockerfile* directives:
  - COPY = instructs docker to copy external files into target image
  - WORKDIR = changes parent folder for file operations inside image
  - ENTRYPOINT = specifies default executable and args to run when launching

Docker – Task #3

- Creating fresh docker image
- Consuming customized git repository to serve AI inputs (JARVIS)
  - A simple bot that processes text input and replies

- HINT: Pay close attention to details
  - Some instructions will be specific and some abstract

Docker – Task #3 - Summary

- An end-to-end docker image (even if toy-model)
- Was a good example why dockers are useful
    - AIML support with Ubuntu LTS isn’t flawless
    - Only a specific combination of package versions works

Going Public – The Last Mile

- Covered most useful docker operations
- How to share image with others?
    1. Send them Dockerfile? – maybe, but build time can be significant
    2. Use container registry service – **YES!**

- A registry service offloads build operations and image storage to the cloud
- There are many registries (used the default till now)

Docker/Container Registry Role

- Standardized service
- Provides image storage, versioning + tagging (and more)
- Use *docker* utility to connect

- May offload image build operations to the cloud

- Checkout: DockerHub and free pricing plan (implicit use till now)

Connecting a Registry

- So far, used implicit registry service (docker hub)
- To explicitly specify a different service:
  1. Use the full URL of the image to consume (more later)
  2. Use *docker login* sub-command

- Most cloud services provide a secure way to login (option 2)
- Example with AWS CLI

Working With a Registry

- Once logged in, two sub-commands of interest:
  1. *docker pull* - get an image from remote service to local computer
  2. *docker push* - send a local image to the remote service

- It is that simple to publish and consume a hosted image

- Specific instructions for AWS

Docker – Task #4

- Push previously created image to a registry
- (Optional) Share image with a colleague

Docker – A Short Summary

- Very useful tool
- Has pros & cons
- Requires technical understanding

- Once set, usually an automated build process (CI/CD)

Questions?

Part 2 – Intro to Kubernetes

Why Going Cloud?

- In part 1, most operations were local (except ECR)
- Just install Docker Desktop and replace Cloud9 env
- Running a container like every virtual machine on our PC

- **Note from me:** highly advised to do so after the workshop ends

Task Parallelization on Clusters

- Given a cluster with many processors
- Want to distribute work “simply”/efficiently

- Many traditional frameworks exist: SLURM, LSF, MPI etc.
- Non are container specific
- In fact, they are very generic indeed

Kubernetes (k8s)

- k8s is a special framework
- Providing a container runtime for clusters:
  - Deploying
  - Managing versions and updates
  - Scaling, load-balancing and so much more
- Can run independently or as cloud hosted service (i.e. EKS on AWS)

k8s - CAUTION

- Very sophisticated stack, takes time to master

- We’ll assume everything is setup
- Official AWS documentation and tutorial

Use Cases

- Very common with live services (e.g. web-apis / nginx)
- Or performing CI/CD tasks (automated build/testing/deployment)
- Most online resources are targeting that

- Our goal is more modest:
  - Utilize k8s as a simple job scheduler
  - Like traditional cluster frameworks
  - Less common
  - Matches research needs for one-shot/short-lived simulations

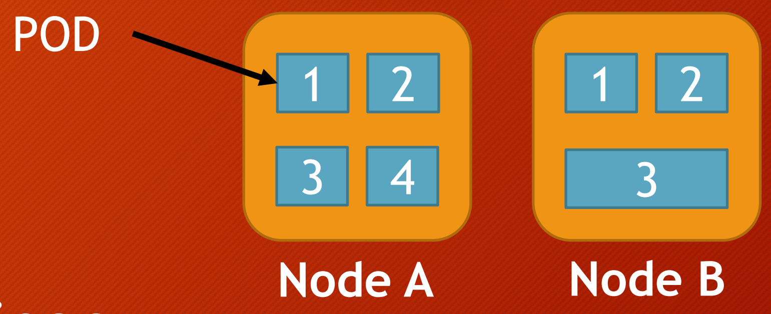
Alternatives to Kubernetes

- There are better cloud services to run ad-hoc containers
- Checkout AWS Lambda
- Similar in other cloud vendors

- k8s is still very useful
- Given on part for mind-opening, educational purpose
- Hard-work appreciated ☺

Diving In – k8s Useful Terminology

- **Node** = actual server with multiple CPUs
- **Pod** = a running container instance



- **Service** = long-lived, managed software in a container
- **Job** = Short-lived, dedicated task software in a container

- A service is always up-and-running, can be paused/resumed
- When a job finishes, its lifetime is over, need to resubmit

k8s CLI Utility

- Recall *docker* that we used for related tasks
- k8s has its own main utility: *kubectl* (Kubernetes Control)
  1. Install in Cloud9
  2. May download separately to your PC
  3. Or use cloud CLI tools to obtain it

- Some may have heard on *helm*, but it’s outside the scope

k8s – Task #1

- Experiment with kubectl
- Get basic information on cluster resources

- Run a simple interactive image in a pod
- Run an interactive JARVIS image from a private repository
- Copy files between a pod and a local/Cloud9 environment

Programmatic Access To k8s

- kubectl can be great, but is still a manual tool

- We’ll use python for programmatic access to k8s
  - Libraries to many other languages exist Client Libraries | Kubernetes

- Useful python client resources:
  - Documentation & source (GitHub)
  - Examples (GitHub)

k8s Python API Usage

- Python client is available in the special package ***kubernetes***

- Credentials must be loaded to authenticate & communicate with a cluster
- Our Cloud9 environment is pre-configured
- Every cloud environment has special instructions to obtain them

k8s – Task #2

- Use python client to k8s
- Obtain and output basic cluster info on nodes/pods

Deploying Jobs

- Job is a one-shot entity

- One can manage & deploy jobs using kubectl
  - Need to create a YAML file and *apply* manually

- Our goal is to automate this task
- Creating jobs in python and deploying to the cluster
  - Deploy = submit for work

k8s – Task #3 - FINAL

- Advanced processing flow with k8s jobs

- Modify our JARVIS image to include several input files
- Use python to create multiple jobs based on a private image

- Optional
  - Monitor jobs status
  - Copy results from finished jobs to local environment

k8s – A Short Summary

- k8s API requires a little effort to understand
- Relay on examples and existing clients
- Documentation may not cover all topics

Questions?

Wrap-up & Advanced Topics

Wrap-up

- Both dockers and k8s are huge topics
- Takes time to master
- Introduced tools and basic operations

- k8s is still evolving

Advanced Topics / Tasks

- Dockers
  - Repeat tasks with a docker installation on your PC/Mac
  - Consume and customize richer images
    - Pre-built with ML/AI etc.

- k8s
  - Become familiar with resource limits
  - Understand *secrets*
  - Experiment with minikube (local k8s server)
  - Attach cloud storage to container
  - Auto-scaling
  - Node type filtering

Thank You ☺