

COMS E6998 012/15099

# Practical Deep Learning Systems Performance

**Lecture 12 11/21/19**

# Logistics

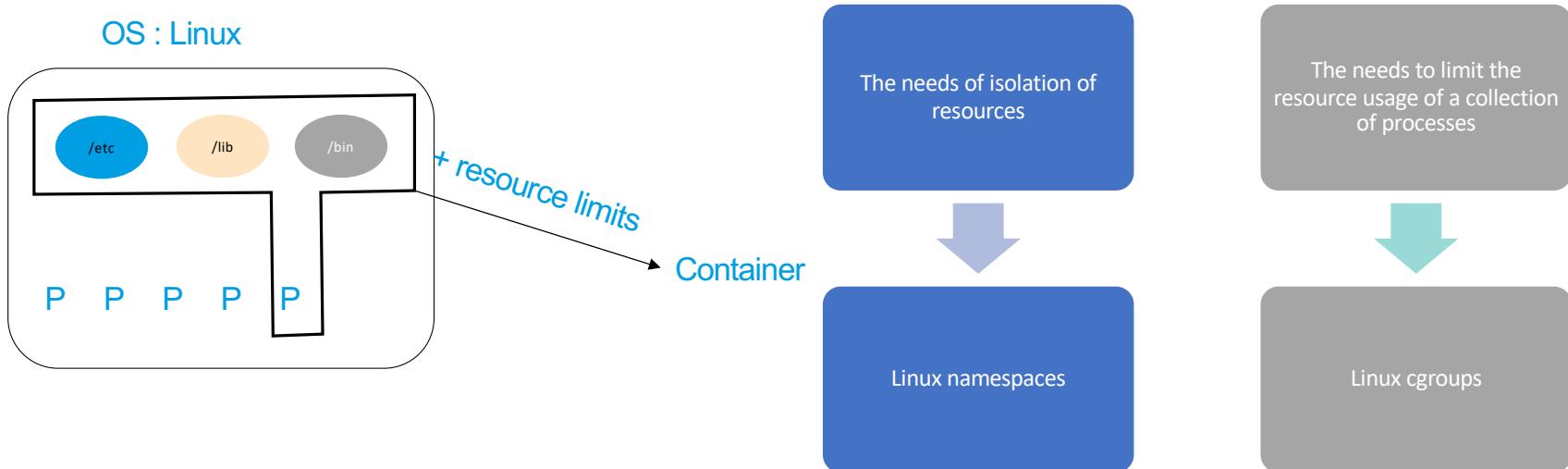
- Homework 4 is due by Nov 26 11:59 PM
  - Question 1: number of iterations can be less than 95K. Train for at least 200-300 epochs or until the model converges. Adjust learning rate schedule accordingly.
  - Question 4: take number of epochs to be 350.
  - Question 4: Work in a group to collect data. At most 5 in a group.
- Homework 5 will be posted soon. Due by Dec 13 11:59 PM
- Dec 10<sup>th</sup>: Deadline for 10% technical community participation
- Dec 9<sup>th</sup> and 10<sup>th</sup> : Project and seminar presentation

# Recall from last 2 lectures

- Lecture 10:
  - Deep Learning cluster schedulers
  - Optimus scheduler
    - Resource allocation as an optimization problem
    - Task placement
    - Sensitivity to prediction error
  - Gandiva scheduler
    - Time-share, packing, job migration, grow-shrink
    - Reactive mode and Introspective mode
- Lecture 11: Guest Lecture by Dr. Eitan Farchi, IBM Haifa Research Lab
  - ML experiment design
  - Data Drift

# What are Containers?

- Containers are an abstraction at the app layer that
  - packages code and dependencies together.
  - each running as isolated processes in user space.
  - Multiple containers can run on the same machine and share the OS kernel with other containers,

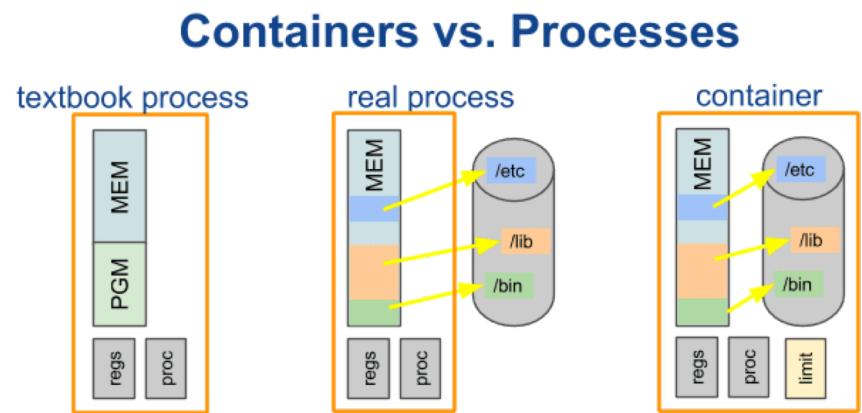


Tutorial: <https://www.youtube.com/watch?v=EnJ7qX9fkcU>  
<https://jvns.ca/blog/2016/10/10/what-even-is-a-container/>

Slide contents from Dr. Chen Wang, IBM Research

# Containers vs. Processes

- Textbook process
  - own address space
  - own program
  - own CPU state
  - own process table entry
- Real process:
  - memory mapped from the filesystem into the process address space
  - consists of dozens of shared libraries, programs and files that are shared.
  - super-dependent on its filesystem environment (e.g. locale info, shared libs and files)
- Container
  - an encapsulation of one program with all its dependencies.
  - with limits on the amount of resources the program can use.

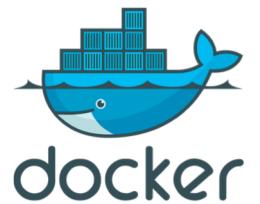


Slide contents from Dr. Chen Wang, IBM Research

# Docker

- Docker is a tool designed to make it easier to *create, deploy, and run* applications by using containers.

- **Docker image:** A Docker image is a file, comprised of multiple layers, used to execute code in a Docker container.
- **Dockerfile:** The Dockerfile is essentially the build instructions to build the image. A Dockerfile is a text document used to indicate to Docker a base image, the Docker settings you need, and a list of commands you would like to have executed to prepare and start your new container.



## Web Application



Layer 9 : **CMD** Start Pintail.ai website  
Layer 8 : **RUN** Commands to set up Pintail.ai Website  
Layer 7 : **ADD** Pintail.ai binaries  
Layer 6 : **FROM** Node.js - Alpine Linux

## Application Framework



Layer 5 : **RUN** Commands to set up Node.js  
Layer 4 : **ADD** Node.js binaries  
Layer 3 : **FROM** Alpine Linux

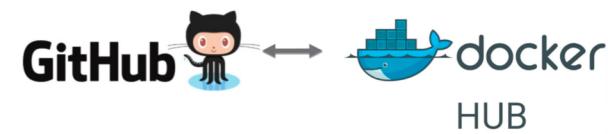
## Operating System



Layer 2 : **RUN** Commands to set up OS  
Layer 1 : **ADD** Operating System binaries  
Layer 0 : **FROM** Scratch

## Docker image repositories:

- Private: IBM Kubernetes Registry
- Public: Docker Hub



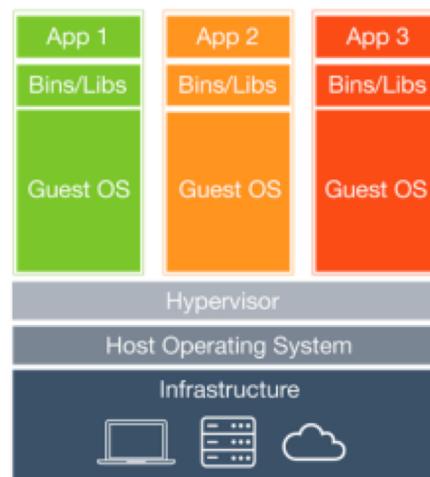
Slide contents from Dr. Chen Wang, IBM Research

# Container vs Virtual Machine

- Virtual machines
  - have a full OS
  - own memory management
  - own device drivers
  - own daemons
  - own binaries, libraries, and applications
- Containers
  - Share the host's OS kernel
  - Share the binaries and libraries in read-only mode.

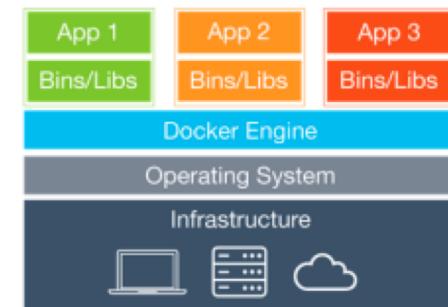
## How is this different from virtual machines?

Containers have similar resource isolation and allocation benefits as virtual machines but a different architectural approach allows them to be much more portable and efficient.



### Virtual Machines

Each virtual machine includes the application, the necessary binaries and libraries and an entire guest operating system - all of which may be tens of GBs in size.



### Containers

Containers include the application and all of its dependencies, but share the kernel with other containers. They run as an isolated process in userspace on the host operating system. They're also not tied to any specific infrastructure - Docker containers run on any computer, on any infrastructure and in any cloud.

Slide contents from Dr. Chen Wang, IBM Research

# Benefits of using Container

- Containerization allows our development teams to
  - move fast
    - Decoupling from environment
    - Compared to VMs, containers are far more lightweight.
  - deploy software efficiently
    - Consistent environment
    - Run Anywhere
    - Isolation
    - Easy version control
    - Agile development
  - and operate at an unprecedented scale
    - Kubernetes: Production-Grade Container Orchestration

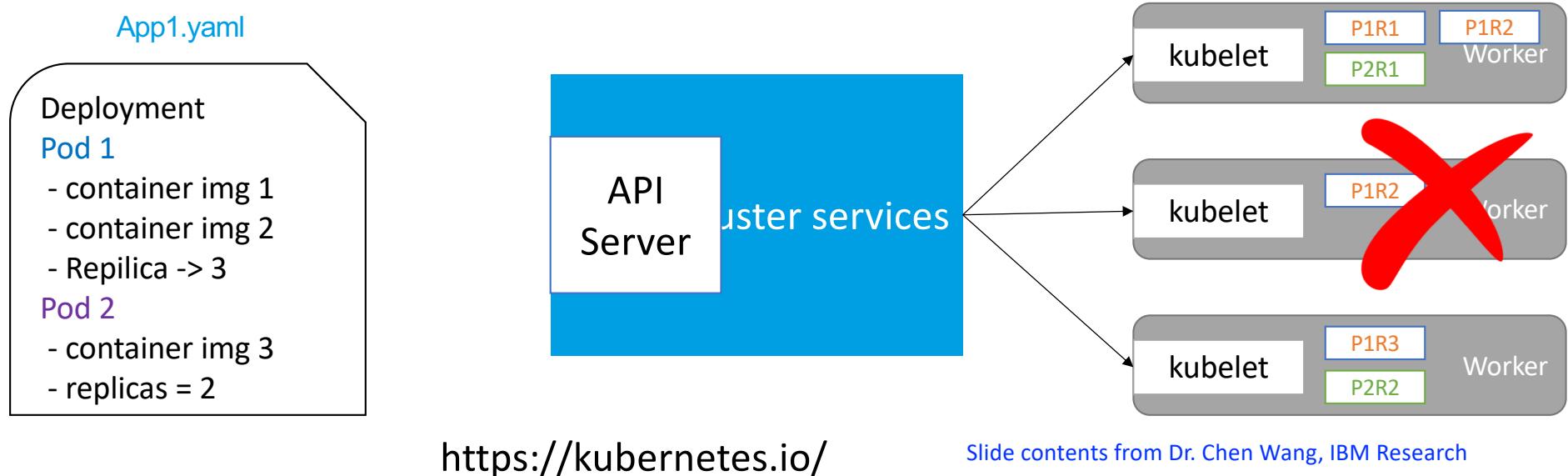
	CONTAINER BENEFITS	VIRTUAL MACHINE BENEFITS
Consistent Runtime Environment	✓	✓
Application Sandboxing	✓	✓
Small Size on Disk	✓	
Low Overhead	✓	

Slide contents from Dr. Chen Wang, IBM Research

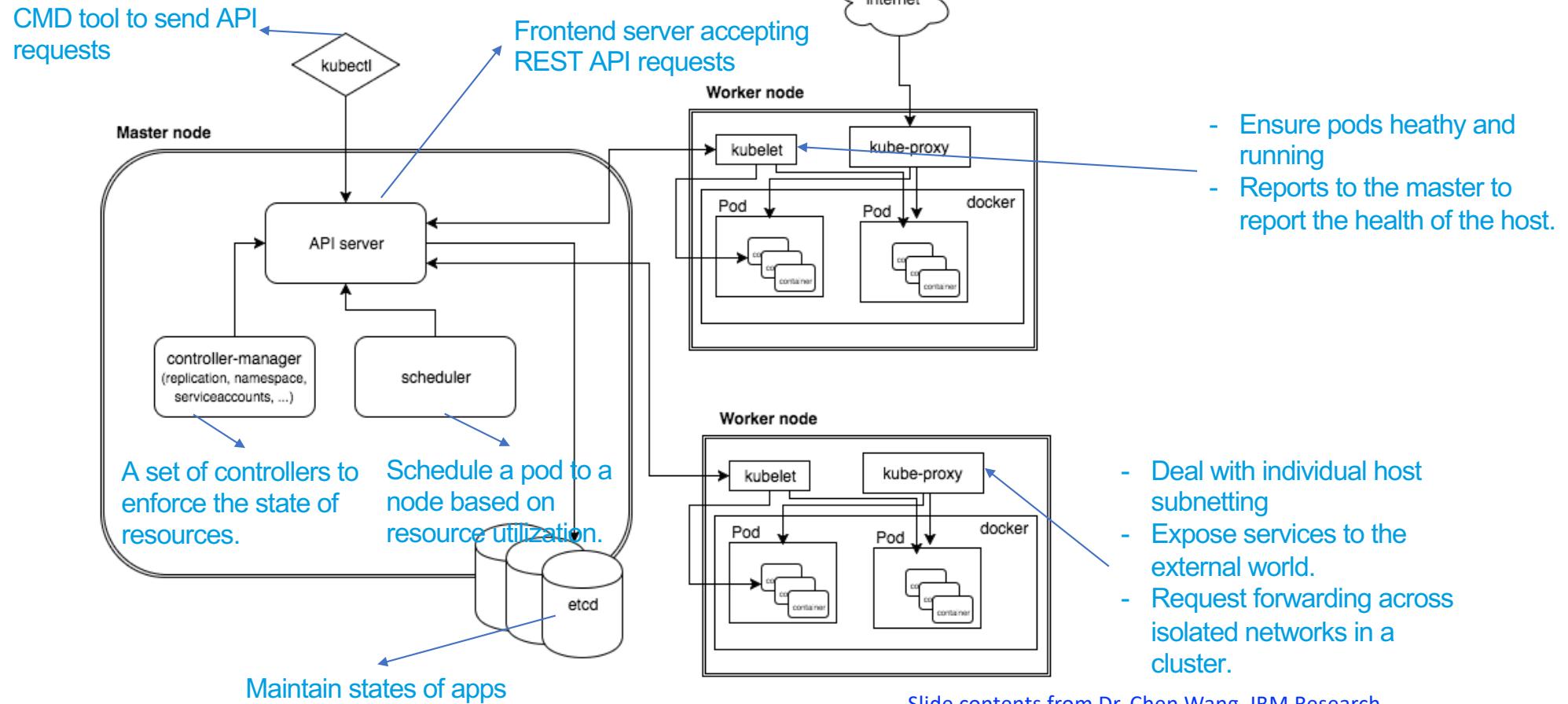
# What is Kubernetes (K8s)?

- Kubernetes is an open source system for managing containerized applications across multiple hosts, providing basic mechanisms for deployment, maintenance, and scaling of applications.

Kubernetes cluster services → “Desired State Management”



# Architecture of Kubernetes



Slide contents from Dr. Chen Wang, IBM Research

# Cloud Services running in Container Cloud

- Internal Customers of IBM Cloud Kubernetes Service: *Launched on May 23, 2017*
  - IBM Watson (Conversation, NLC, sentiment analysis, etc.)
  - IBM Deep Learning as a Service (DLaaS)
- Google: *Launched in 2015*
  - Almost everything from YouTube to Gmail.
  - Google Kubernetes Engine: <https://cloud.google.com/kubernetes-engine/>
- Alibaba:
  - 90% of their main business, especially e-commerce like Taobao and Tmall
  - Alibaba Cloud Container Service for Kubernetes: <https://www.alibabacloud.com/product/kubernetes>
- AWS: *Nov. 29, 2017*
  - Amazon Elastic Container Service for Kubernetes: <https://aws.amazon.com/eks/>
- Azure: *Oct. 27, 2017*
  - Azure Kubernetes service: <https://azure.microsoft.com/en-us/services/kubernetes-service/>

# Cloud Storage

- File, Block, Object Storage
- Medium article: <https://medium.com/ai-platforms-research/fundamentals-of-ibm-cloud-storage-solutions-8739f36f024e>

## Is K8s Enough for Data scientists and ML Developers ?

- Too “low-level” for use by data scientists
  - Training job vs pods, deployment, replica sets ...
- DL training workloads need specialized scheduling algorithms, not natively supported by K8s
- Cannot track DL specific job status: DOWNLOADING, PROCESSING, STORING, HALTED, RESUMED

# Distributed DL Platform Requirements

- Ease of Use
- Flexibility
  - Support of different frameworks
- Dependability
  - Highly available, reliable, and fault tolerant
- Efficiency
  - Negligible platform overhead; End-to-end job performance should be close to bare metal
- Scalability
  - End-to-end job performance should not degrade significantly with increasing load on the cluster

# IBM FfDL (Fabric for Deep Learning)

- Cloud-based distributed DL platform
- Design Goals: a distributed DL platform should be easy to use, flexible, dependable, scalable, and efficient.
- Open source: <https://github.com/IBM/FfDL>

# FfDL Design

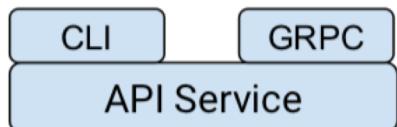
- Cloud-native platform architected as a set of loosely-coupled microservices communicating with each other using gRPC  
<https://grpc.io>

*“Cloud native is a term used to describe container-based environments. Cloud-native technologies are used to develop applications built with services packaged in containers, deployed as microservices and managed on elastic infrastructure through agile DevOps processes and continuous delivery workflows.”*

- Employs Kubernetes (K8S) for container orchestration and cluster management
- Flexibility through Containerization
  - Maintains docker images of different frameworks
  - Communication and synchronization methods internal to frameworks are preserved
  - FfDL takes user code, and automatically instantiates Docker containers (from its library of Docker images) containing all necessary packages to run the training job

# FfDL Components

## FfDL Core Services



Training Metrics Service



## FfDL Platform Services

- MongoDB
  - Stores job metadata as well as job history (for a specific user or client organization)
  - Data is long-lived and spans several jobs
- Etcd:
  - Store status updates and messages
  - Data is short lived (a DL job's data is erased after it terminates)
- Training Metrics Service
  - collects metric about both training jobs and FfDL microservices
- API Service
  - gRPC and REST end points
  - FfDL CLI interacts with API service

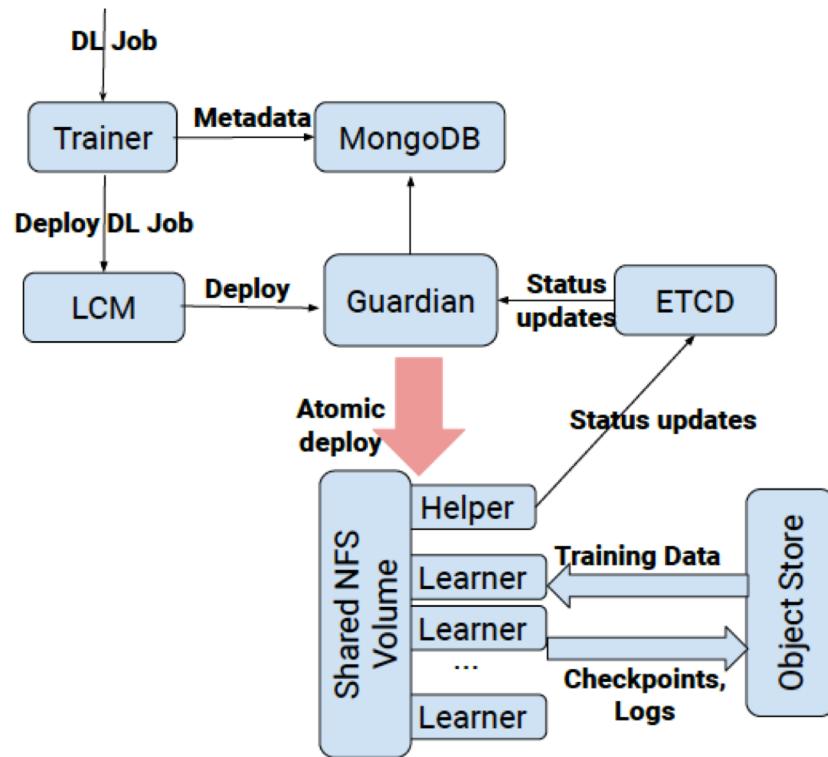
# Life Cycle Manager (LCM)

- Responsible for the job from submission to completion or failure.
- Takes care of deployment, monitoring, garbage collection, and user-initiated job termination
- LCM performs some of these functions directly, but creates delegates for others
  - Scalability
  - Delegates avoid LCM becoming a single point of failure
- Atomic provisioning of DL jobs
  - Either the whole job is provisioned with the requisite resources or none

## *Guardian: LCM delegate for a job's lifecycle*

- Guardian is an LCM launched delegate for atomic deployment and further monitoring of each DL job.
- LCM instantiates Guardian with all the metadata of the DL job.
- Guardian executes the multi-step process of actually deploying the DL job by interacting with K8S.
- Guardian monitors the progress of successfully deployed jobs
- Unsuccessful job deployment are marked as FAILED by Guardian in MongoDB

# FfDL Job Deployment Flow



# Training job specification

- YAML file
- [https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/ml\\_dlaas\\_e2e\\_example.html](https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/ml_dlaas_e2e_example.html)

# Scalability and Elasticity

- Each microservice is replicated
  - Number of replicas chosen based on the size of the cluster as well as the expected number of concurrent training jobs
- Replicated microservices are K8S deployments or replica sets
  - gRPC requests are automatically load balanced by K8S among the available replicas
- Replica set for each microservice can be elastically scaled as physical machines are added to or removed from the cluster.
- Guardian for job deployment ensures that FfDL can handle thousands of deployment requests concurrently.
- Guardians consume only a fraction of a CPU and need little RAM.

# T-shirt size of learners

CPU-threads	thpt-1P100	thpt-1V100
2	65.96	106.46
4	66.14	106.5
8	65.67	107.24
16		107.45
28		107.47

**Table 4.** Throughput (images/sec) scaling of VGG-16/Caffe with CPU threads for learners with 1 P100 and 1 V100 and batch size 75.

CPU-threads	InceptionV3	Resnet-50	VGG-16
16	217.8 (86.8%)	345.3 (93.3%)	216.2 (98.7%)
28	223.6 (90.5%)	345.8 (92.7%)	216.2 (97.3%)

**Table 6.** Throughput (images/sec) scaling of TensorFlow with CPU threads for learners with 1 V100 and batch size 128. Also shown in parentheses is the GPU utilization.

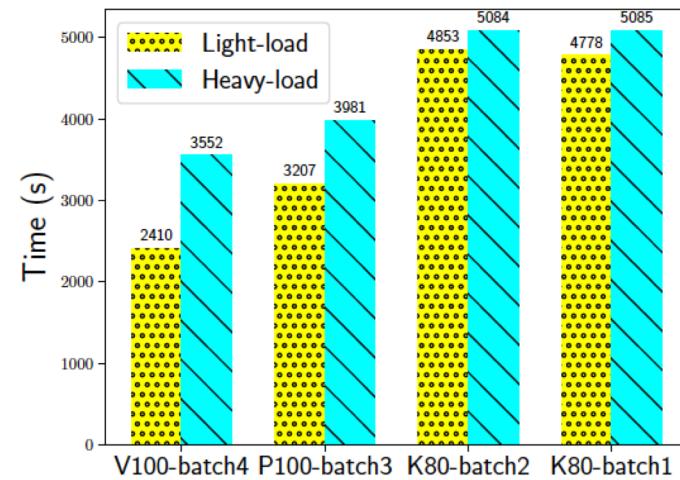
GPU-type	CPU	memory (GB)
1-K80	4	24
2-K80	8	48
4-K80	16	96
1-P100	8	24
2-P100	16	48
1-V100	26	24
2-V100	42	48

**Table 5.** T-shirt size recommendation for FfDL jobs.

# Scale Testing of FfDL Cluster

GPU-type-batch#	jobs-LL	jobs-HL	start time
K80-batch1	30	300	first 1 min
K80-batch2	24	240	after 15 min
P100-batch3	11	110	after 30 min
V100-batch4	5	50	after 32 min

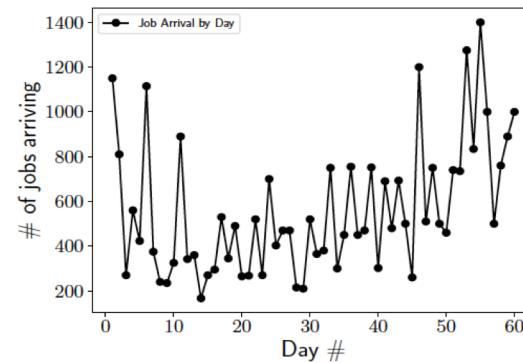
**Table 7.** Light-load (LL) and heavy-load (HL) job mix



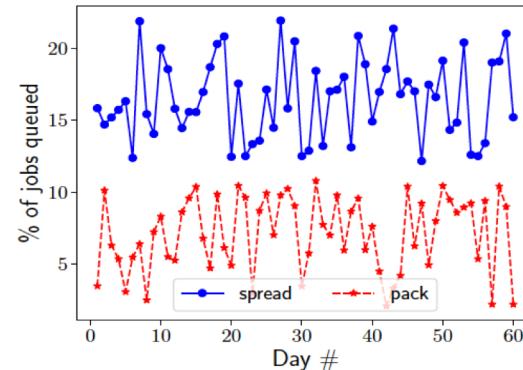
**Figure 5.** E2E job runtime by GPU-type for light-load and heavy-load scenarios

# Job Scheduling in FfDL

- Spread
  - Default placement policy of K8S pods
  - Spread distributes pods over the cluster, and avoids placing two pods which are replicas of the same workload on the same physical machine
- Problems:
  - Increased communication costs
  - Increased fragmentation
- Pack
  - Pods from a DL job are packed into as few physical machines as possible
  - Implemented as an extension to K8S scheduler



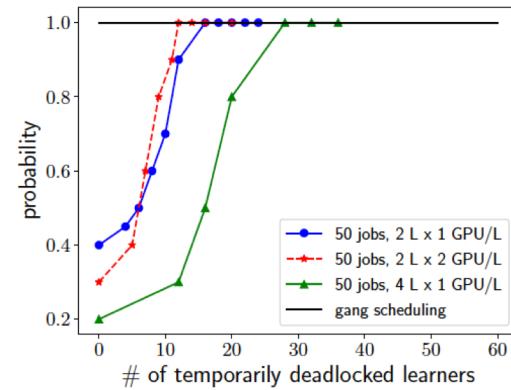
(a) Arrival of jobs by day over a 60 day period at a production cluster



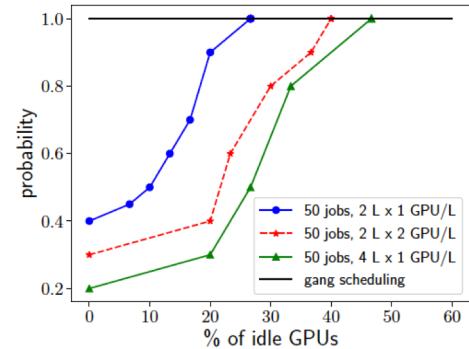
(b) Percentage of arriving jobs that would be queued for over 15 mins, in SPREAD vs. PACK

# Gang Scheduling

- DL job can have multiple pods
- K8S scheduler doesn't consider one whole job while scheduling, it considers each of the learner pods individually
- Scheduling pods of a job individually can cause temporary deadlocks; learners are waiting and holding hardware resources while waiting for other pods (learners) of the job to get deployed
- Gang scheduler schedules all pods that belong to a DL job holistically, as a group/gang



(a) CDF of the probability of temporarily deadlocked learners with and without gang scheduling



(b) CDF of the probability of idle GPUs with and without gang scheduling

# Kubeflow

- <https://www.kubeflow.org/docs/about/kubeflow/>
- Kubeflow is *the machine learning toolkit for Kubernetes*.
- Goal: to making deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable
- Kubeflow started as an open sourcing of the way Google ran [TensorFlow](#) internally, based on a pipeline called [TensorFlow Extended](#).
- Multi-architecture, multi-cloud framework for running entire machine learning pipelines

# Kubeflow Logical Components

- Jupyter Notebooks
- Hyperparameter Tuning
- Pipelines
- Serving
- Training

# Kubeflow Pipelines

- A platform for building, deploying, and managing multi-step ML workflows based on Docker containers.
- The Kubeflow Pipelines platform consists of:
  - A user interface (UI) for managing and tracking experiments, jobs, and runs.
  - An engine for scheduling multi-step ML workflows.
  - An SDK for defining and manipulating pipelines and components.
  - Notebooks for interacting with the system using the SDK.

# Pipeline

- Description of ML workflow
- Pipeline is formed using pipeline components
- Pipeline includes
  - Components and their inputs and outputs
  - Definition of the inputs (parameters) required to run the pipeline
- <https://www.kubeflow.org/docs/pipelines/overview/pipelines-overview/>

# Pipeline Components

- <https://www.kubeflow.org/docs/pipelines/overview/concepts/component/>
- A *pipeline component* is a self-contained set of user code, packaged as a [Docker image](#), that performs one step in the pipeline.
- A component for data preprocessing, data transformation, model training, data visualization...
- A component is analogous to a function, in that it has a name, parameters, return values, and a body.
- Each component in a pipeline executes independently.
- The components do not run in the same process and cannot directly share in-memory data.

# Component Specification

## Example of a component specification

A component specification takes the form of a YAML file, `component.yaml`. Below is an example:

```
name: xgboost4j - Train classifier
description: Trains a boosted tree ensemble classifier using xgboost4j

inputs:
- {name: Training data}
- {name: Rounds, type: Integer, default: '30', help: Number of training rounds}

outputs:
- {name: Trained model, type: XGBoost model, help: Trained XGBoost model}

implementation:
container:
  image: gcr.io/ml-pipeline/xgboost-classifier-train@sha256:b3a64d57
  command: [
    /ml/train.py,
    --train-set, {inputPath: Training data},
    --rounds, {inputValue: Rounds},
    --out-model, {outputPath: Trained model},
  ]
```

<https://www.kubeflow.org/docs/pipelines/reference/component-spec/>

Metadata (name, description), input and output interfaces, implementation (docker image url)

Real world component specifications

[Link here](#)

# Example Pipeline

- <https://www.kubeflow.org/docs/pipelines/overview/pipelines-overview/>
- [https://github.com/kubeflow/pipelines/tree/master/samples/core/xgboost\\_training\\_cm](https://github.com/kubeflow/pipelines/tree/master/samples/core/xgboost_training_cm)

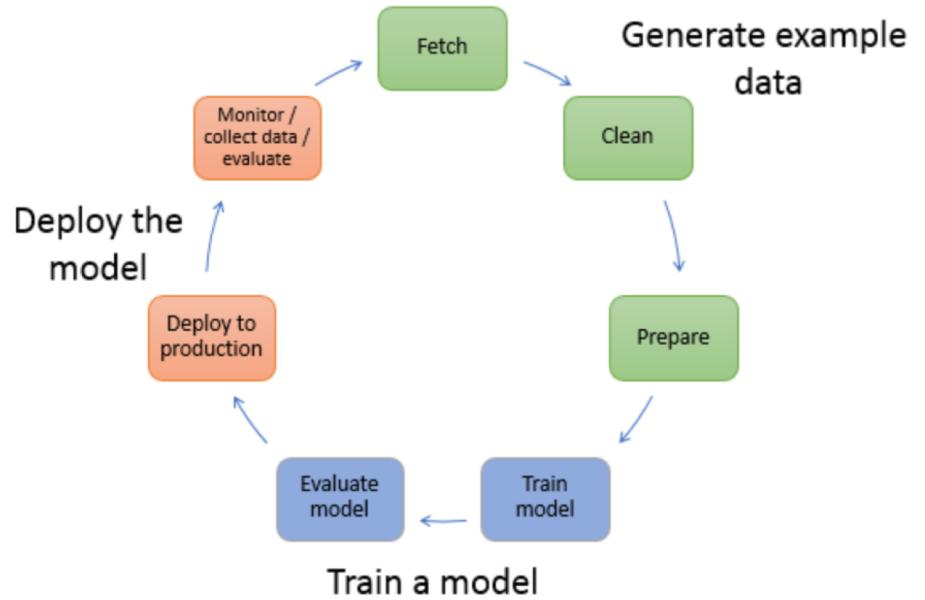
# Kubeflow Tutorial Using GCP

- <https://codelabs.developers.google.com/codelabs/kubeflow-introduction/index.html#0>

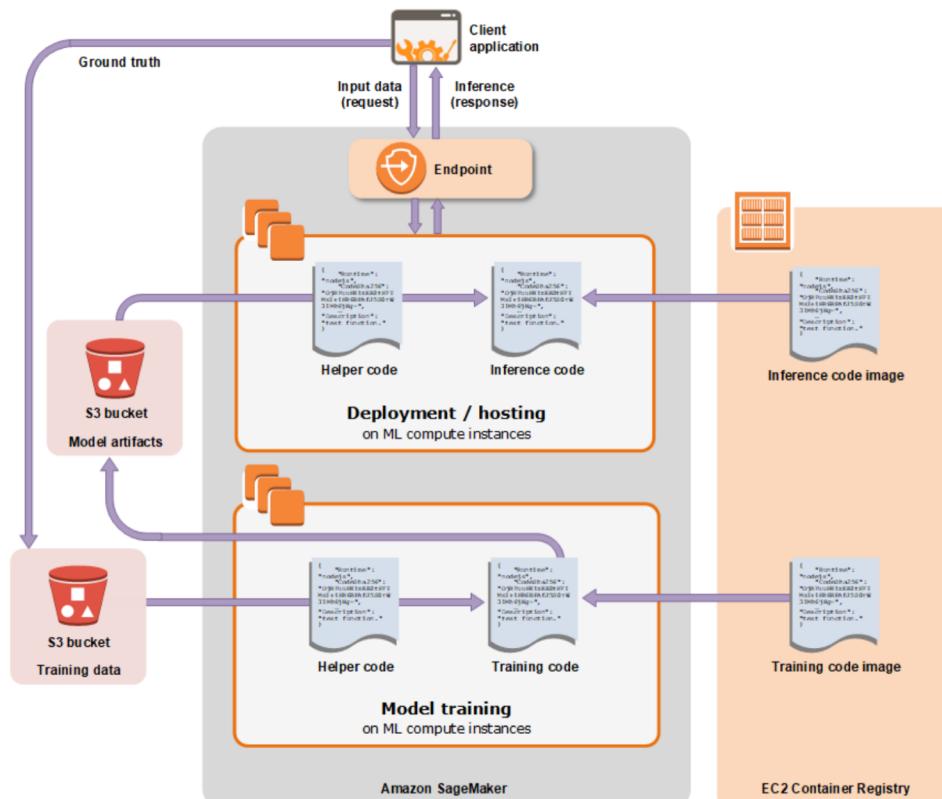
# Amazon Sagemaker

- Fully managed machine learning service by Amazon
- Supports:
  - Quick and easy building and training of ML models
  - Model deployment in production-ready hosted environment

## Typical ML Workflow



# Train with Amazon Sagemaker



## Training algorithm options

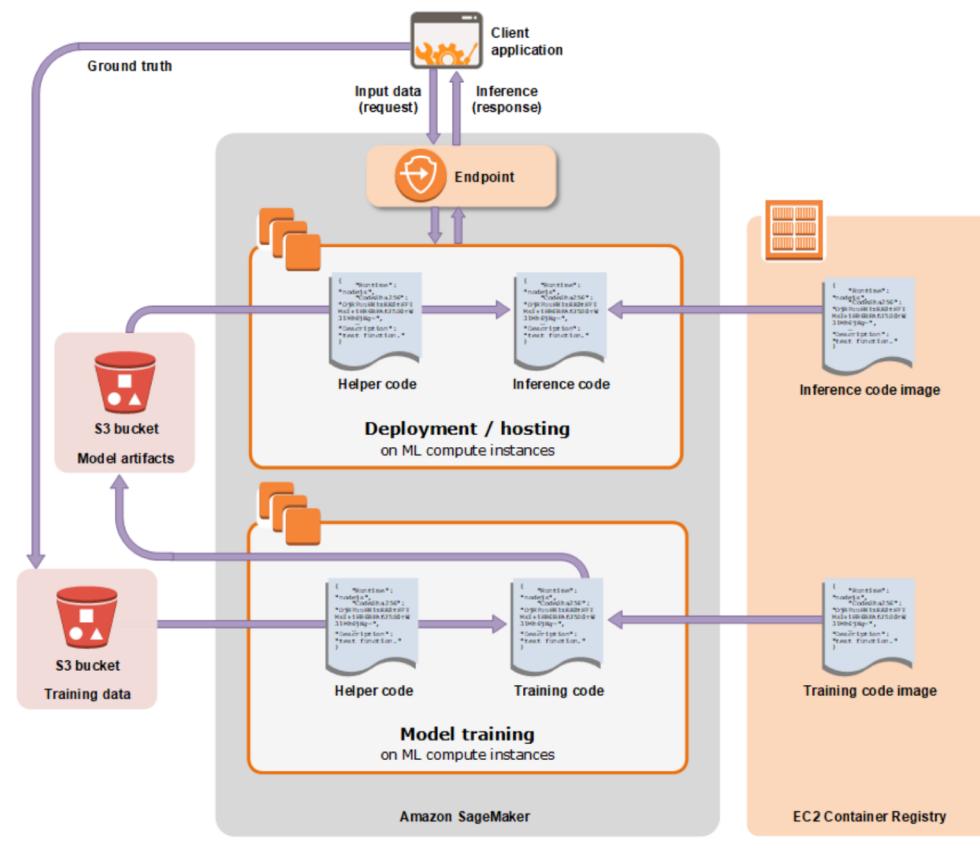
- Use an out-of-the-box algorithm provided by Amazon
  - Pre-built docker images
- Use Apache Spark MLlib with Amazon Sagemaker
- Custom python code to train with DL frameworks
  - Tensorflow and Apache MXNet
- Use your own custom algorithm in any programming language and framework
  - Package as a docker image and register it
- Use an algorithm from AWS Marketplace

<https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-training.html>

# CreateTrainingJob API

- [https://docs.aws.amazon.com/sagemaker/latest/dg/API\\_CreateTrainingJob.html](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html)

# Deploy on Amazon Sagemaker Hosting Services



<https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-hosting.html>

# Other ML Platforms

- [Azure ML](#)
- [Google Cloud ML Engine](#)
- [Amazon Sagemaker RL](#)

# Reference Papers

- Jayaram et al. [FfDL: A Flexible Multi-tenant Deep Learning Platform.](#) Middleware 2019.
- Bhattacharjee et al. [IBM Deep Learning Service](#). IBM Journal of Research and Development. 2017.

# Blogs, Videos, Code Links

- Janakiram. [10 KEY ATTRIBUTES OF CLOUD-NATIVE APPLICATIONS](#)
- AWS Documentation. [How Amazon Sagemaker Works](#)
- James Quigley. [Microservices, Docker, and Kubernetes](#)
- Ben Corrie. [What is a container?. Video](#)
- Microsoft Azure. [Kubernetes Learning Path](#). A hands-on course.
- Kubeflow. [Getting Started with Kubeflow](#)