



利用 Kubernetes 實作 TensorFlow 分散式訓練與模 型服務分享

About Me

Software Engineer @ **inwin**STACK



☑ 早期在校主要撰寫於 Java 與 Objc 程式語言，並專注開發 iOS Mobile 應用程式，並具備四年開發經驗。次要研讀 Hadoop 與 Spark 資料運算框架，以及 Linux 相關技術。

☑ 工作期間則專注於 OpenStack、Ceph 與 Container 相關等 Cloud Native 技術與開源專案，閒暇之餘會參與相關專案社區及貢獻，並利用部落格、GitHub 與 GitBook 做筆記。

☒ 現在則在學習如何利用相關技術管理 Ethereum 礦場，以公司電費轉換提升自我年終獎金(誤)。



kairen(kyle.b@inwinstack.com)



<https://kairen.github.io/>

Agenda

Background

Why choose TensorFlow?

Distributed TensorFlow

Why TensorFlow + Kubernetes?

TensorFlow on Kubernetes

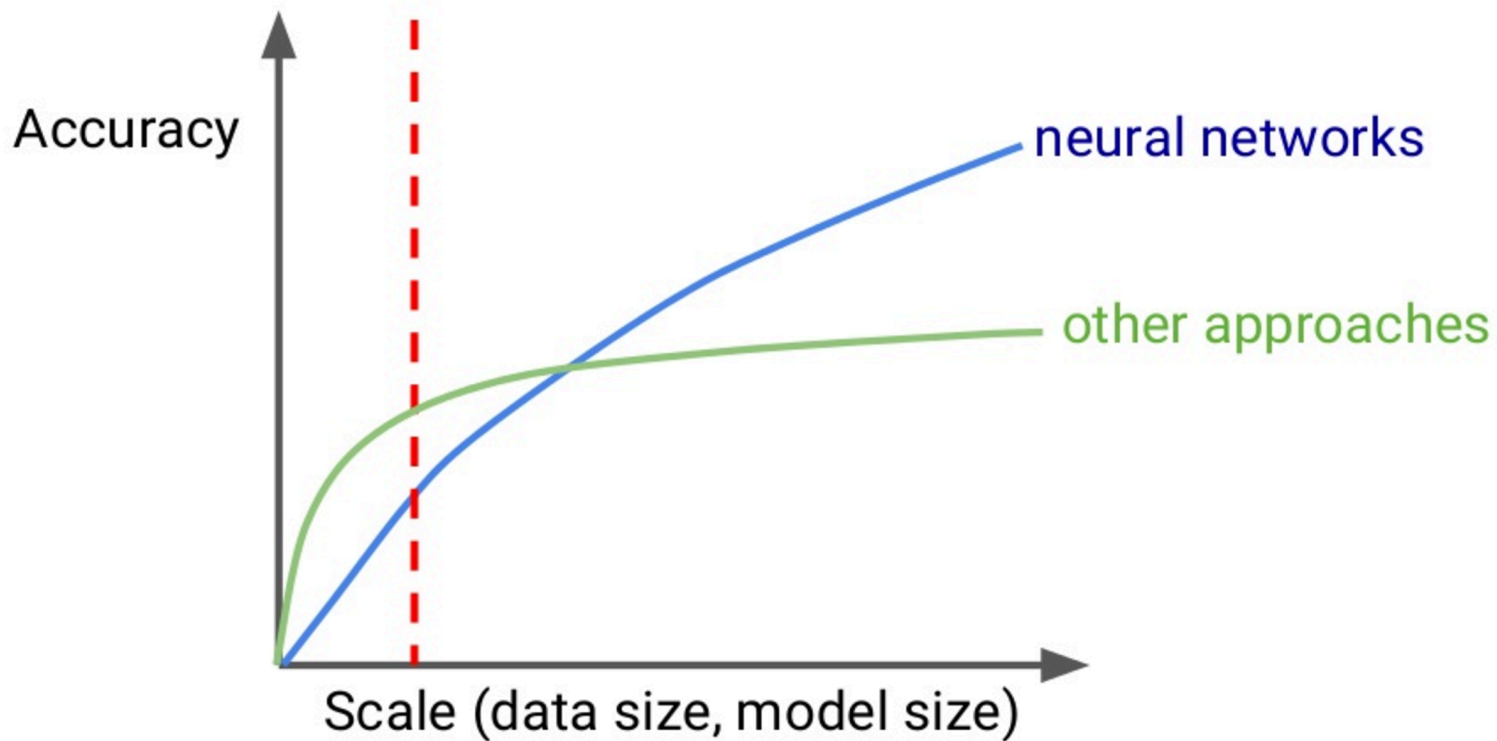




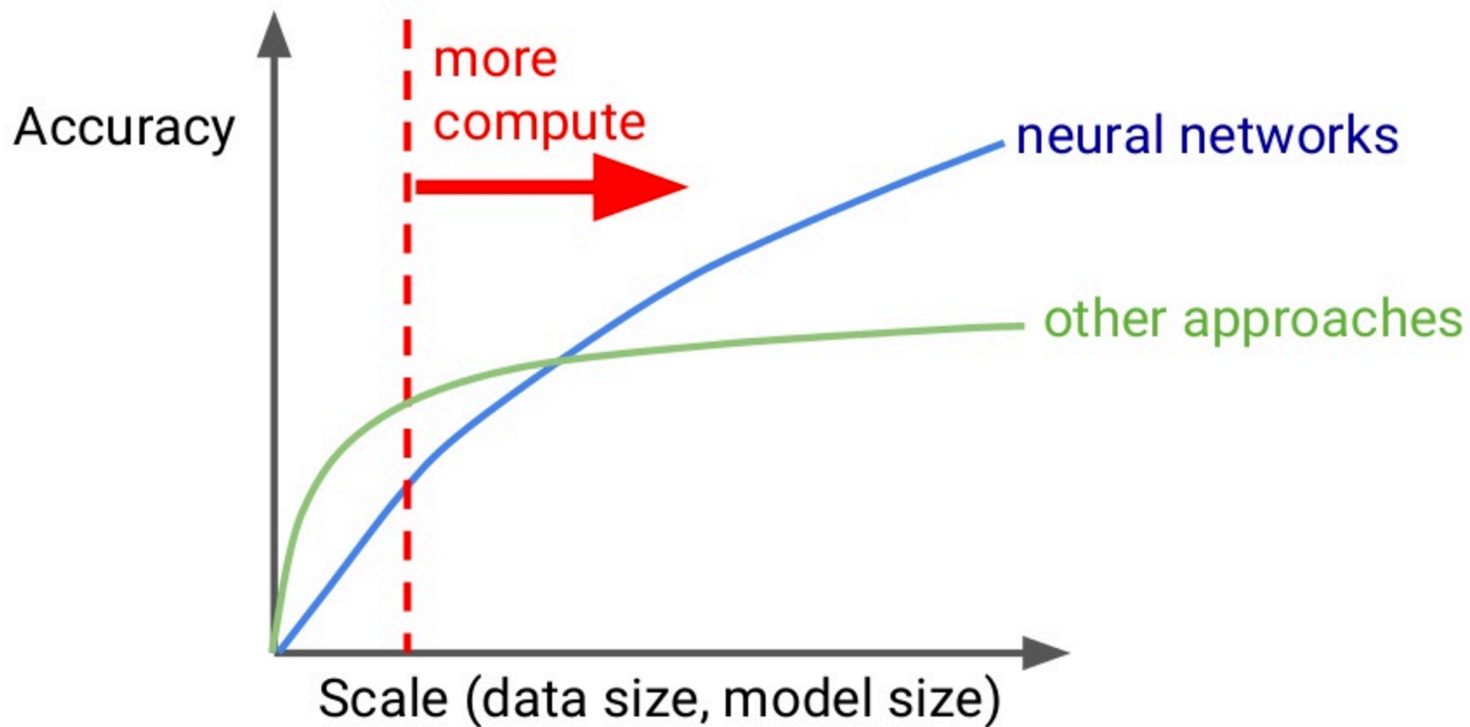
Background

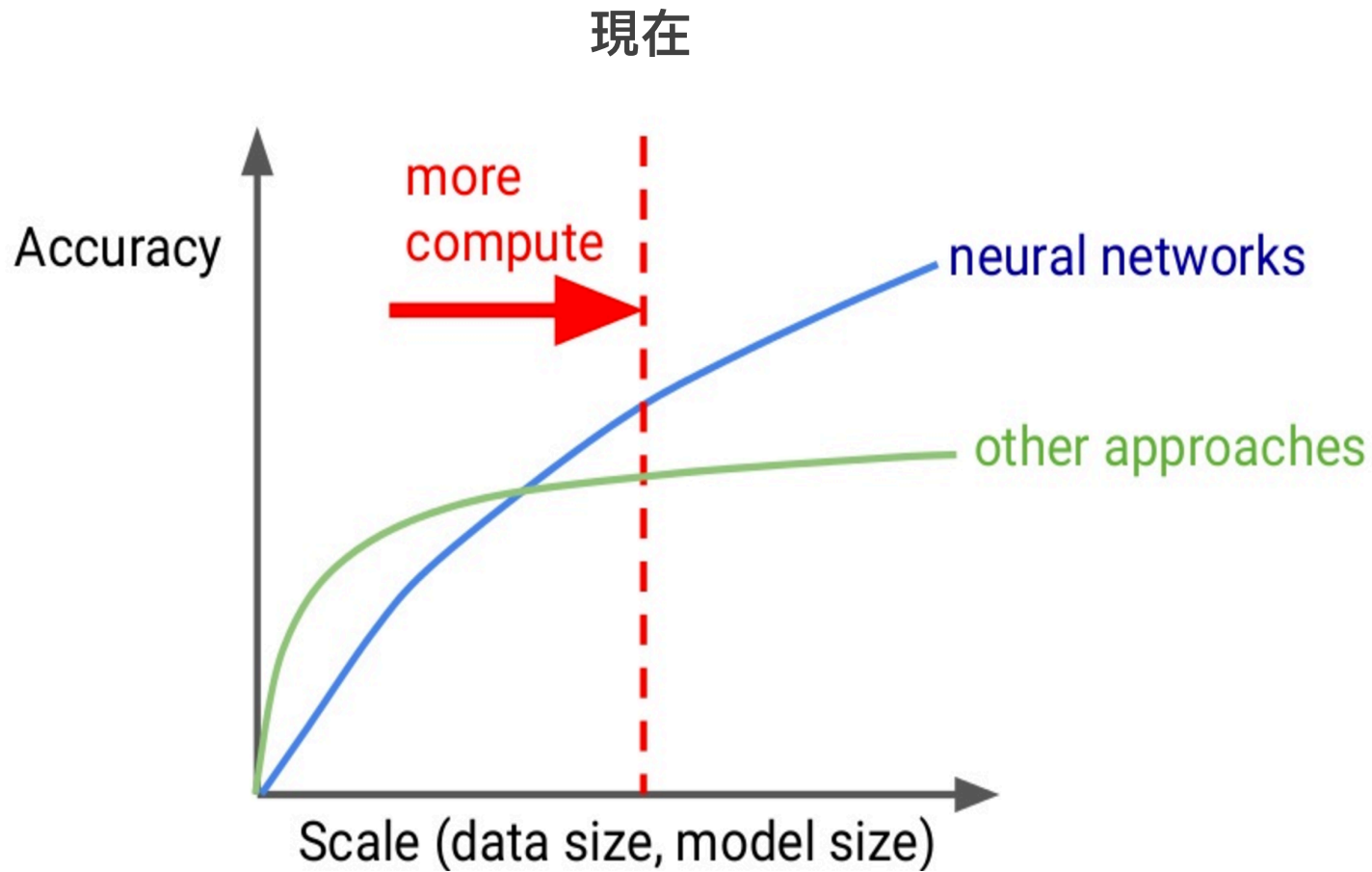
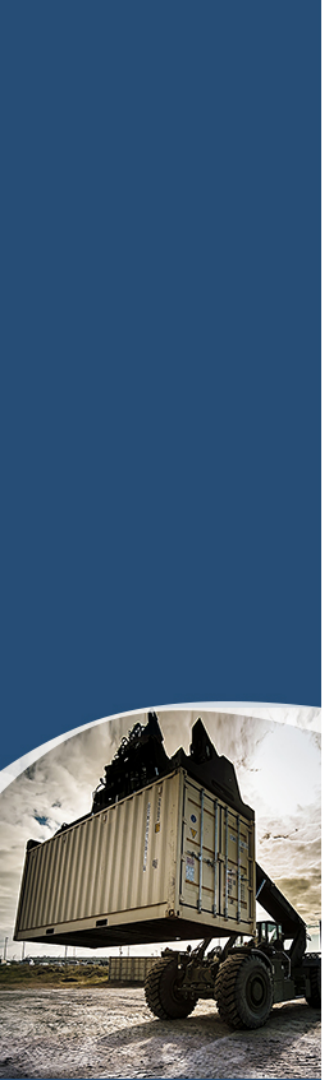
在 2016 年一場圍棋『人機大戰』中，人類李世乜敗給了 Google DeepMind 的 AlphaGo，這為全球帶來了一些震撼話題，甚至讓人聯想到『人工智慧將漸漸取代人』這樣的想法。

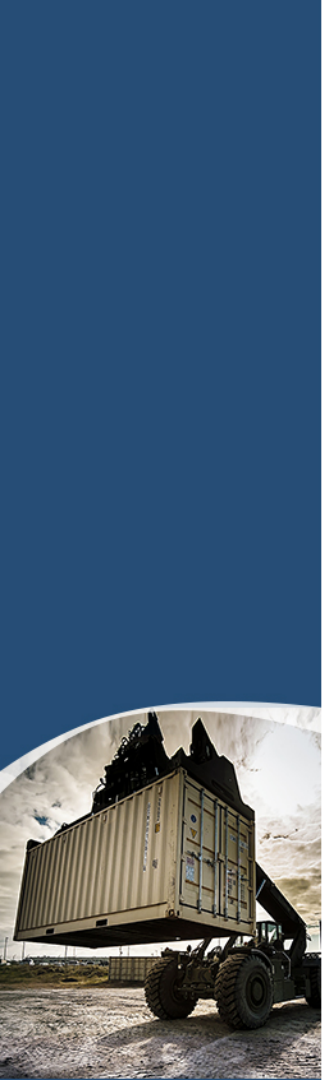
1980s and 1990s



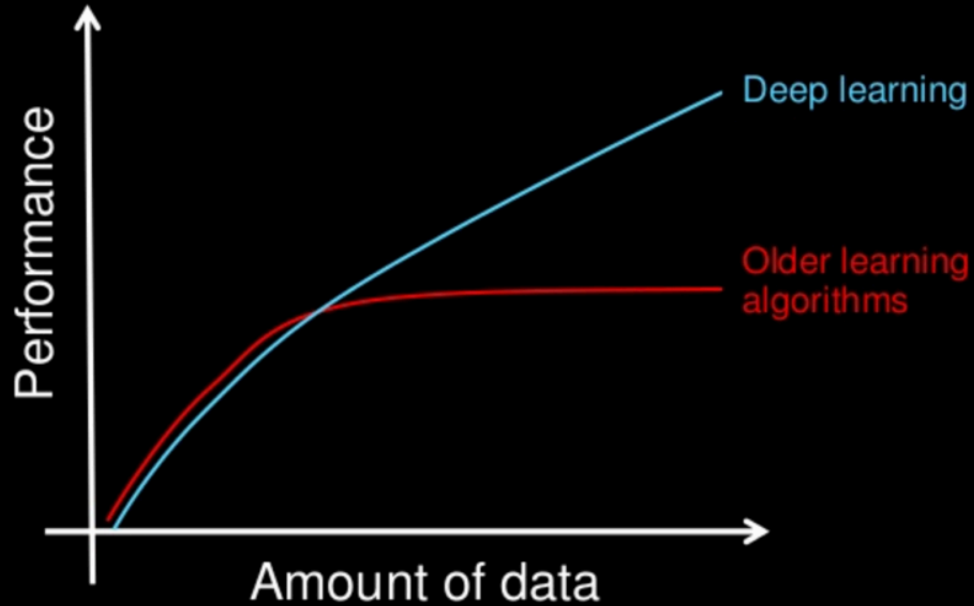
1980s and 1990s







Why deep learning



How do data science techniques scale with amount of data?

Slide by Andrew Ng, all rights reserved.

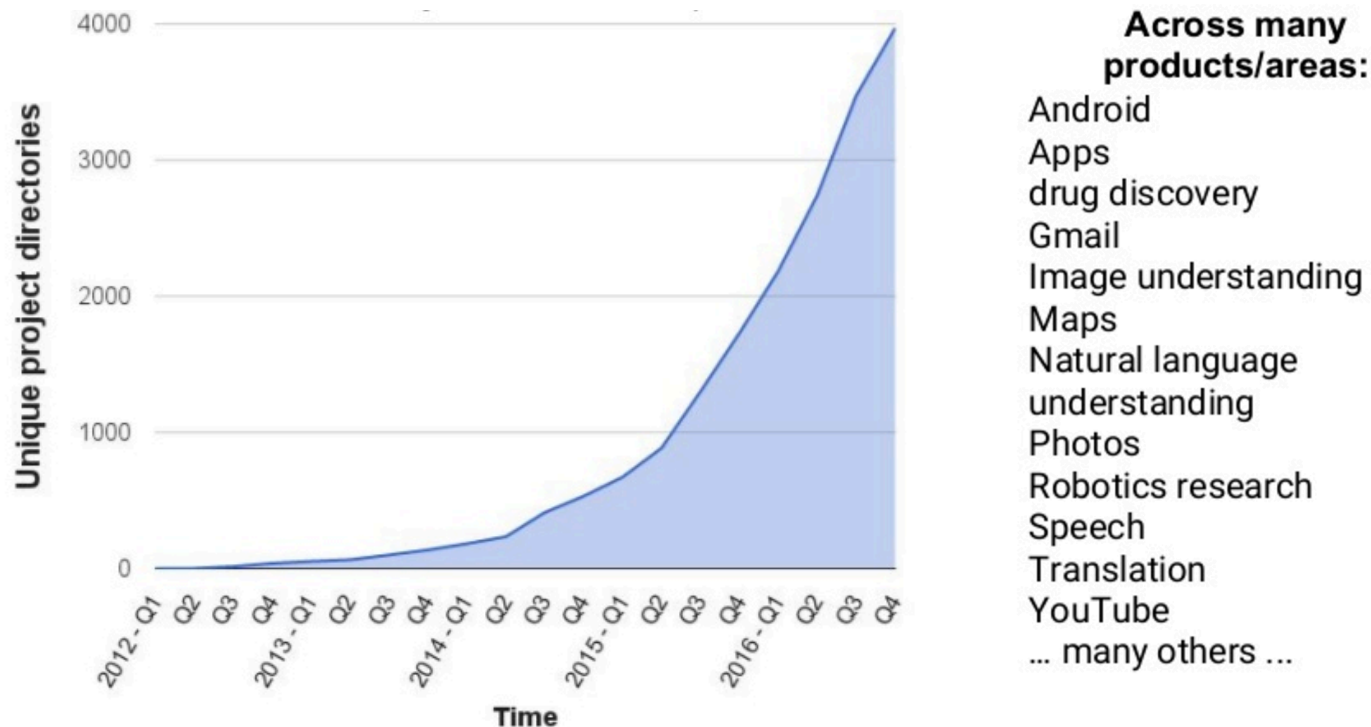




00:48:07

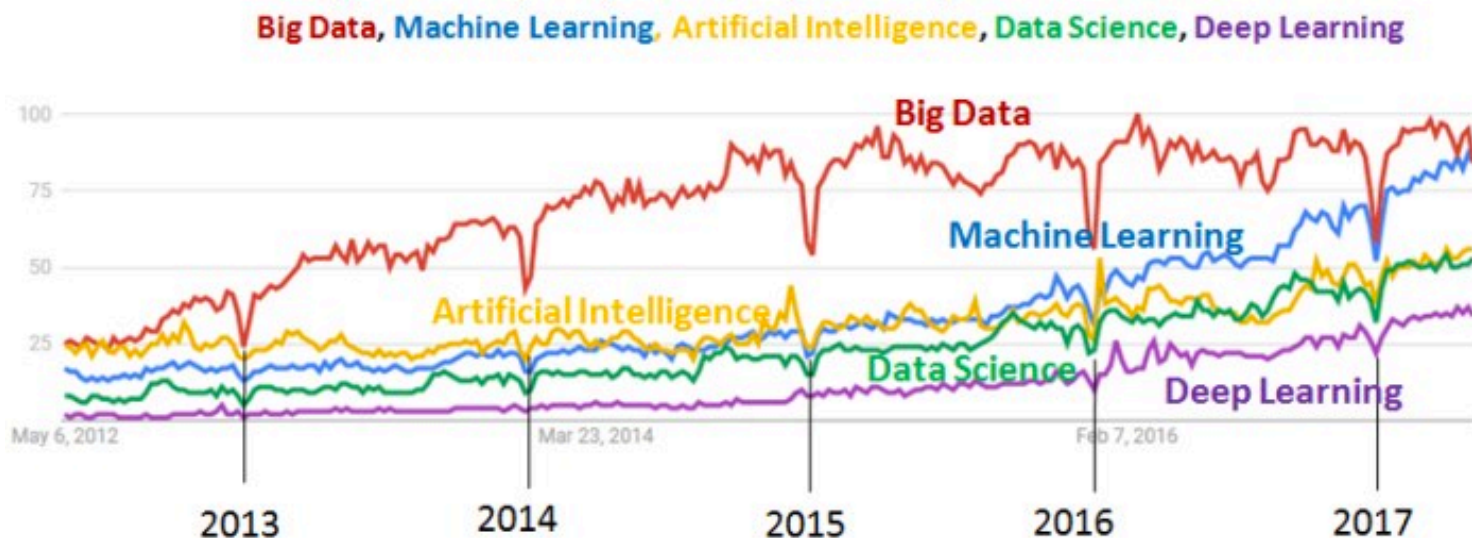
Growing Use of Deep Learning at Google

Of directories containing model description files



Google Trend

May 2012 - April 2017, Worldwide





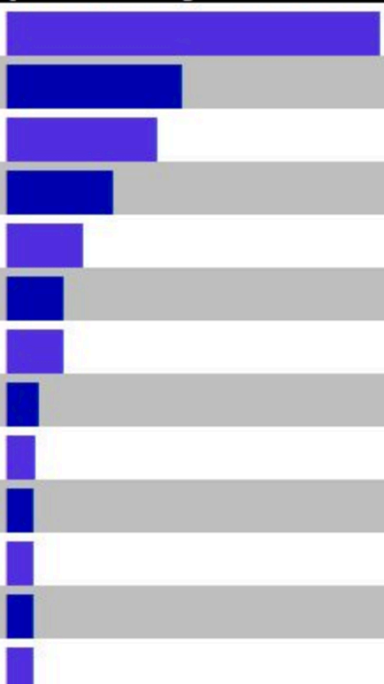
AI / Machine & Deep Learning



Accumulated GitHub metrics

April 12, 2017

Aggregate popularity $(30 \cdot \text{contrib} + 10 \cdot \text{issues} + 5 \cdot \text{forks}) \cdot 1e-3$

#1:	209.25		tensorflow/tensorflow
#2:	95.91		BVLG/caffe
#3:	82.36		fchollet/keras
#4:	61.69		dmlc/mxnet
#5:	41.20		Theano/Theano
#6:	35.00		deeplearning4j/deeplearning4j
#7:	32.17		Microsoft/CNTK
#8:	18.73		torch/torch7
#9:	17.29		baidu/paddle
#10:	15.14		pytorch/pytorch
#11:	14.22		pfnet/chainer
#12:	14.05		NVIDIA/DIGITS
#13:	12.62		tflearn/tflearn



Deep Learning libraries: GitHub activity

from February 11 to April 12, 2017

new contributors from 2017-02-11 to 2017-04-12

#1:	131	tensorflow/tensorflow
#2:	63	fchollet/keras
#3:	51	pytorch/pytorch
#4:	49	dmlc/mxnet
#5:	18	Theano/Theano
#6:	11	BVLC/caffe
#7:	11	Microsoft/CNTK
#8:	9	tflearn/tflearn
#9:	9	pfnet/chainer
#10:	8	torch/torch7
#11:	5	deeplearning4j/deeplearning4j
#12:	4	NVIDIA/DIGITS
#13:	3	baidu/paddle

new forks from 2017-02-11 to 2017-04-12

#1:	4192	tensorflow/tensorflow
#2:	991	fchollet/keras
#3:	810	BVLC/caffe
#4:	517	deeplearning4j/deeplearning4j
#5:	414	dmlc/mxnet
#6:	307	pytorch/pytorch
#7:	244	Microsoft/CNTK
#8:	211	tflearn/tflearn
#9:	134	torch/torch7
#10:	131	Theano/Theano
#11:	116	baidu/paddle
#12:	88	NVIDIA/DIGITS
#13:	55	pfnet/chainer

new issues from 2017-02-11 to 2017-04-12

#1:	1175	tensorflow/tensorflow
#2:	568	fchollet/keras
#3:	499	dmlc/mxnet
#4:	286	pytorch/pytorch
#5:	257	Microsoft/CNTK
#6:	239	deeplearning4j/deeplearning4j
#7:	219	baidu/paddle
#8:	173	Theano/Theano
#9:	171	BVLC/caffe
#10:	112	NVIDIA/DIGITS
#11:	84	tflearn/tflearn
#12:	57	pfnet/chainer
#13:	47	torch/torch7

aggregate activity from 2017-02-11 to 2017-04-12

#1:	36.64	tensorflow/tensorflow
#2:	12.52	fchollet/keras
#3:	8.53	dmlc/mxnet
#4:	6.09	BVLC/caffe
#5:	5.92	pytorch/pytorch
#6:	5.12	deeplearning4j/deeplearning4j
#7:	4.12	Microsoft/CNTK
#8:	2.93	Theano/Theano
#9:	2.86	baidu/paddle
#10:	2.17	tflearn/tflearn
#11:	1.68	NVIDIA/DIGITS
#12:	1.38	torch/torch7
#13:	1.12	pfnet/chainer





Why choose TensorFlow?

為何選擇 TensorFlow ? 是因為 Google 的開源專案 ? 還是因為其特性呢 ?



Open, standard
software for
general machine
learning.



- 2015 年底，Google 在 GitHub 開源了 TensorFlow。
- 2016 年四月，TensorFlow 釋出了**分散式運算**版本。
- 2017 年一月，**1.0 預覽版本**釋出，在眾多開發者貢獻下，其功能越來越穩定。
- 2017 年三月，在開發者大會中，釋出用於工業的 **1.0 正式版本**。
- 現在 TensorFlow 已經來到 1.2 版本，增加了一些特性，與優化效能，並且更穩定。





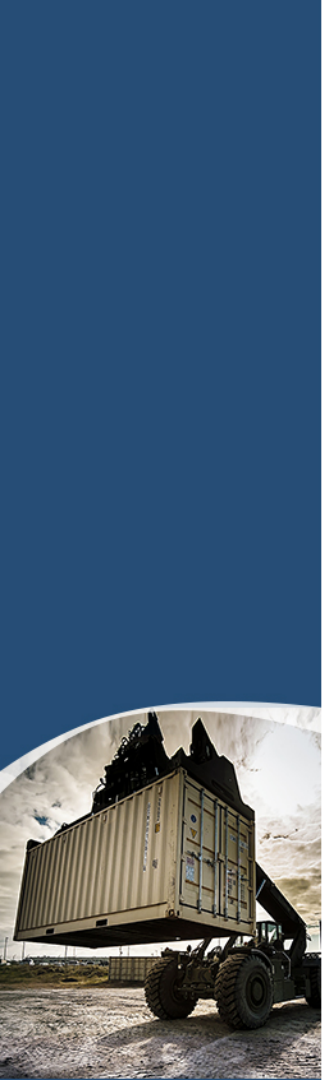
Great for Deep
Learning in
particular.

TensorFlow Pros

Deep Learning framework from Google:

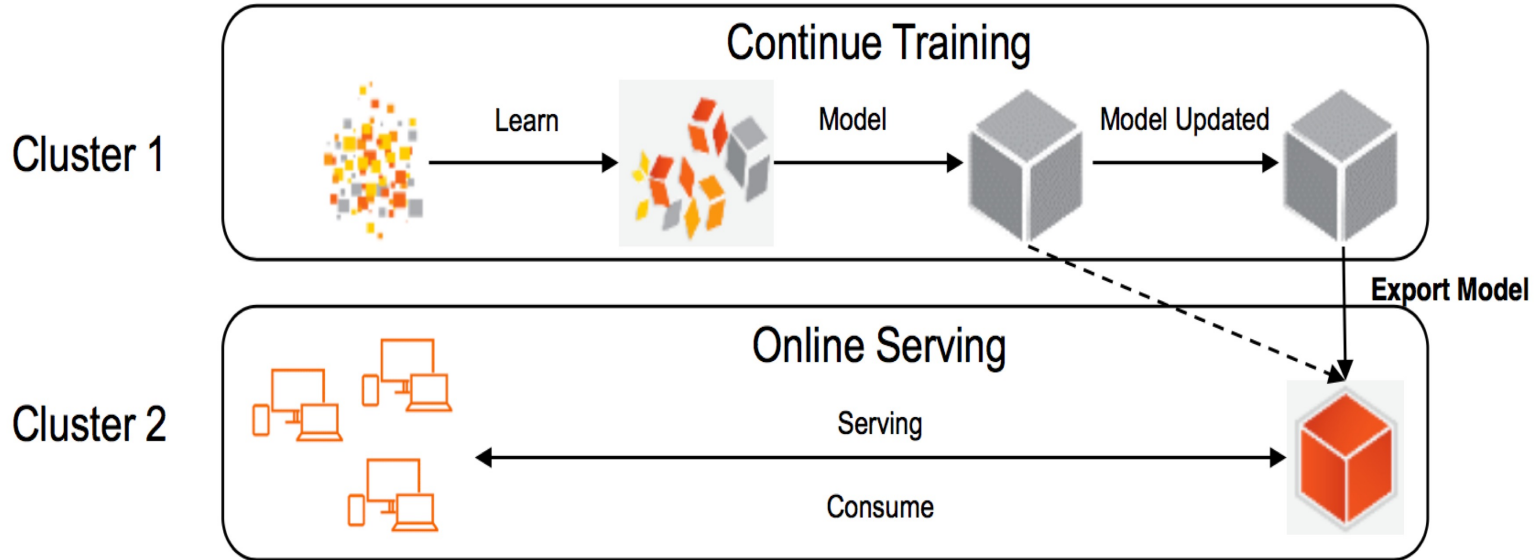
- 支援GPU/CPU/TPU，等異構平台。
- 前端支援多種開發程式語言，如 Python/C++/Go 等。
- 後端透過 C++ 與 CUDA 等開發而成。
- 除了能夠執行深度學習演算法外，還能夠實現許多常見演算法，如隨機森林與支援向量機等。
- 內建支援分散式訓練與服務。
- DNN building block, ckpt/queue/...
- Docker 與 Kubernetes 支援。





Connect Research and Production

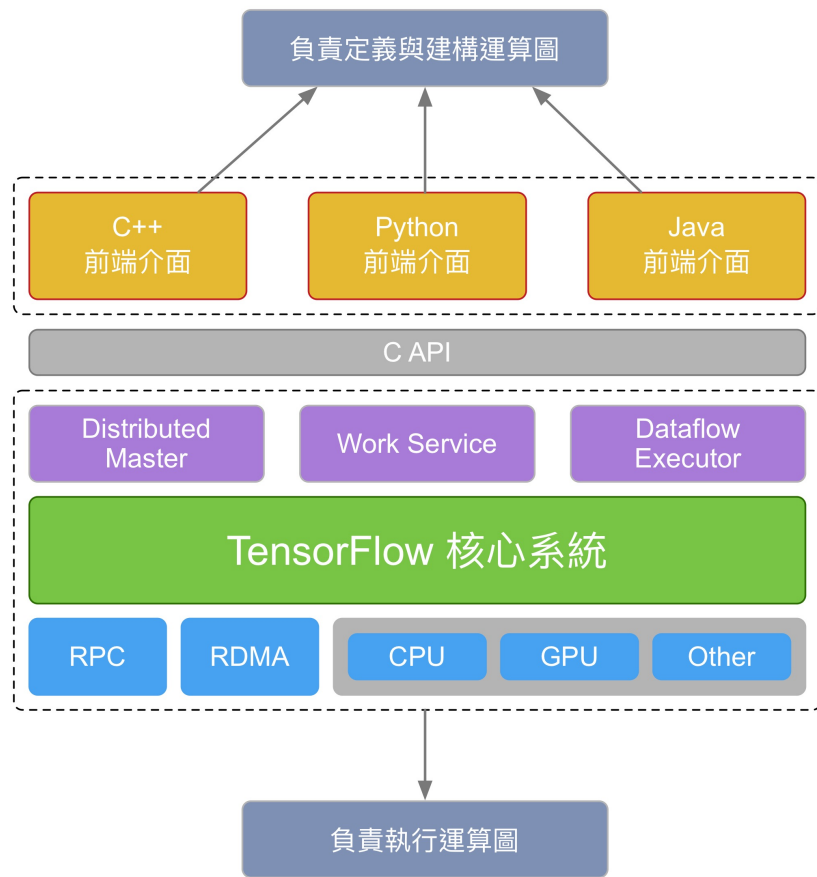
Providing Training -> Serving model



TensorFlow Architecture

TensorFlow 一般分為**前端 API 程式開發介面**與**後端執行系統**。

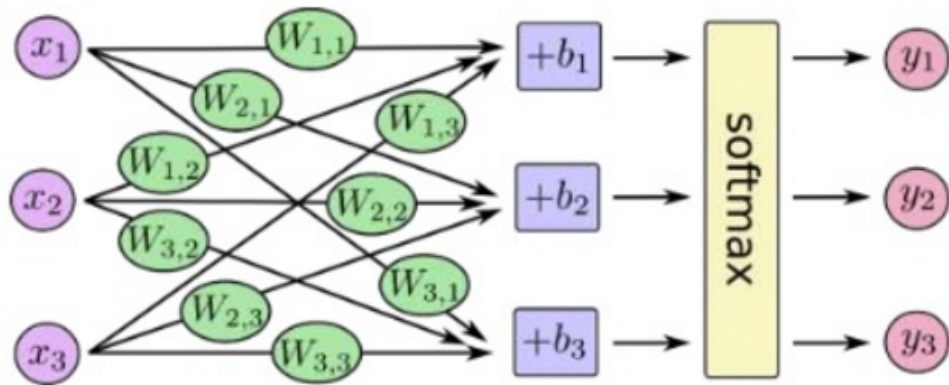
兩者分別提供**定義建構運算圖**與**利用硬體資源執行運算圖**。



Tensor + Flow

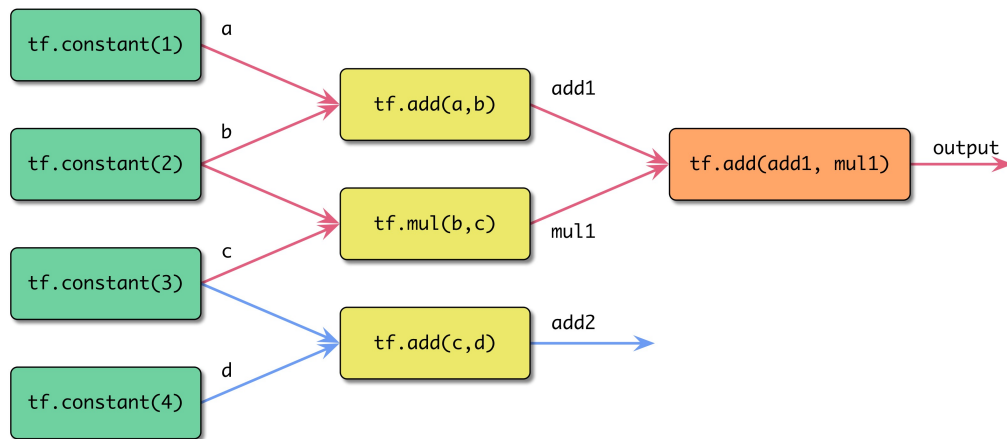
TensorFlow 其實在意思上是要用兩個部分來解釋，Tensor 與 Flow：

- **Tensor**：是中文翻譯是張量，其實就是一個n維度的陣列或列表。
如一維 Tensor 就是向量，二維 Tensor 就是矩陣等等。
- **Flow**：是指 Graph 運算過程中的資料流。



Dataflow Graphs

資料流圖(Data Flow Graphs)是一種有向圖的節點(Node)與邊(Edge)來描述計算過程。



```
a = tf.constant(1)
b = tf.constant(2)
c = tf.constant(3)
d = tf.constant(4)
```

```
add1 = tf.add(a, b)
mul1 = tf.multiply(b, c)
add2 = tf.add(c, d)
output = tf.add(add1, mul1)
```

```
with tf.Session() as
sess:
    print
sess.run(output)
```

Companies using TensorFlow

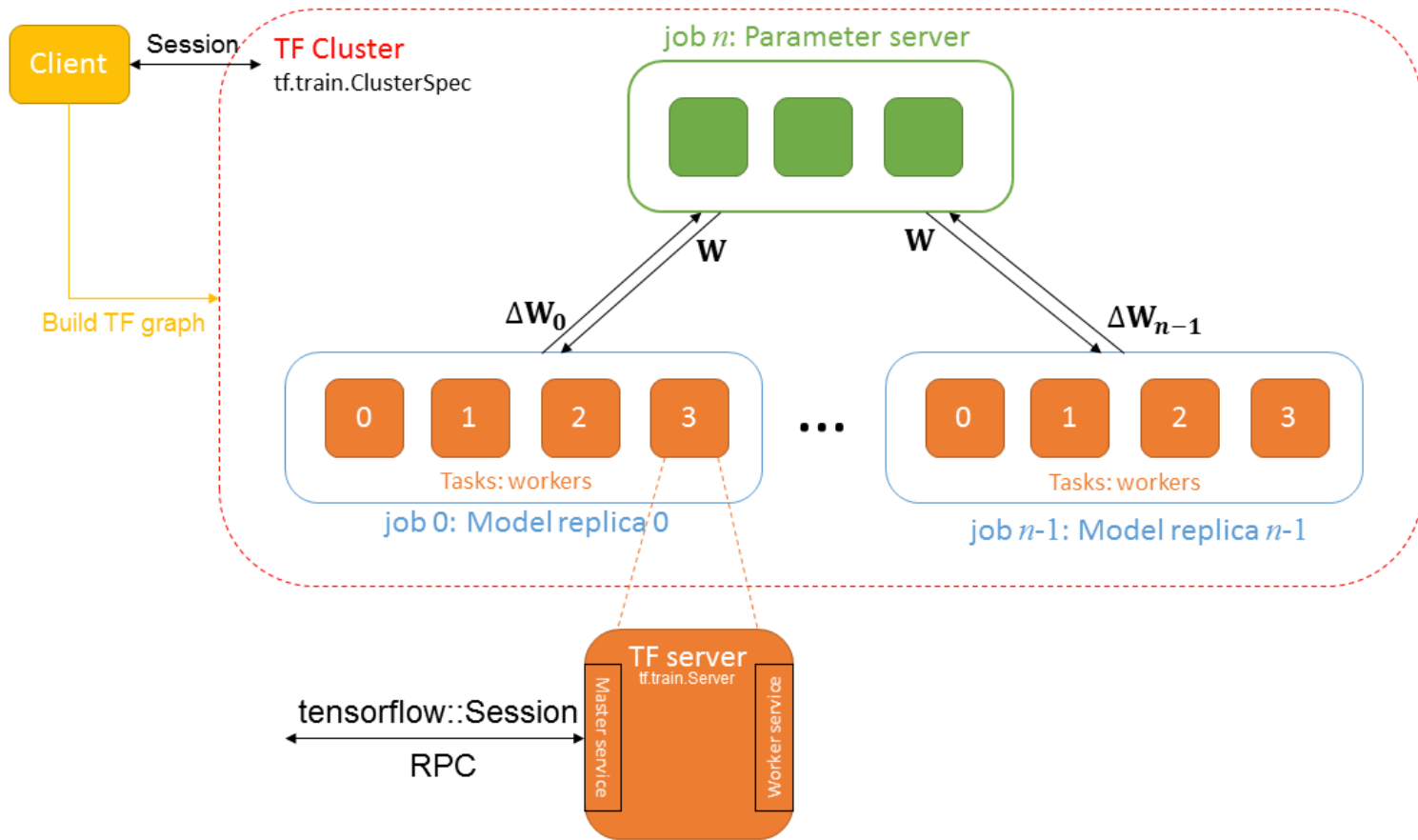




Distributed TensorFlow

去年四月中旬 Google 釋出 TensorFlow 0.8，新增加分散式運算能力，使 TensorFlow 可在數百台的機器上執行訓練程序，建立各種機器學習模型，將原本要耗費數天或數個星期的模型訓練過程縮短到數小時。

Distributed TensorFlow Architecture



TensorFlow cluster concept

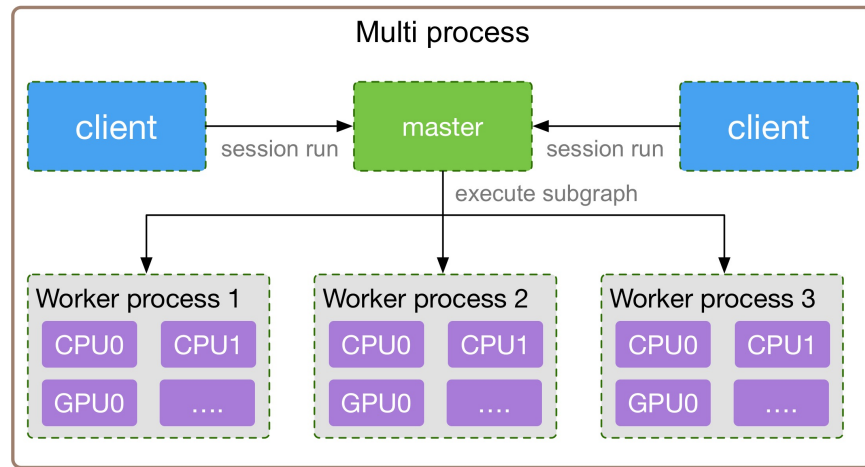
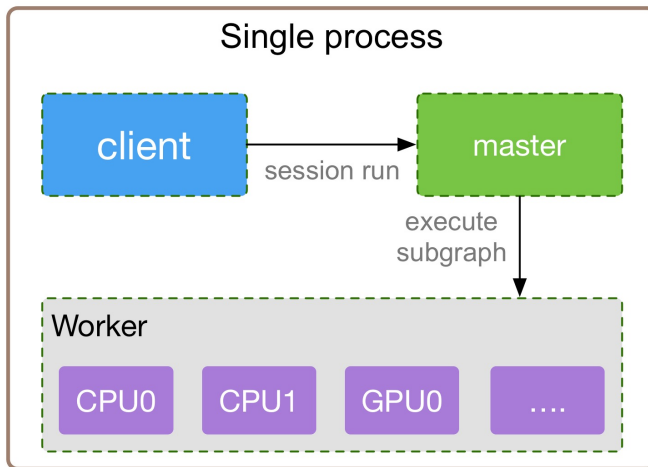
TensorFlow 的叢集就是一組工作任務，每個任務是一個服務，服務分成了 Master 與 Worker，並提供給 Client 進行操作：

- **Client**：是用於建立 TensorFlow 計算 Graph，並建立與叢集進行互動的 `tensorflow::Session` 行程，一般由 Python 或 C++ 實作，單一客戶端可以同時連接多個 TF 伺服器連接，同時也能被多個 TF 伺服器連接。
- **Master Service**：是一個 RPC 服務行程，用來遠端連線一系列分散式裝置，主要提供 `tensorflow::Session` 介面，並負責透過 Worker Service 與工作的任務進行溝通。
- **Worker Service**：是一個可以使用本地裝置(CPU 或 GPU)對部分 Graph 進行運算的 RPC 邏輯，透過 `worker_service.proto` 介面來實作，所有 TensorFlow 伺服器均包含了 Worker Service 邏輯。



TensorFlow training mode

TensorFlow 在執行時，會分為單機模式與分散式模式。單機模式下 client、master 與 worker 都在同一部機器上執行。而分散式模式則可以拆分 client、master 等在不同機器上執行。



TensorFlow Job

TensorFlow 的工作(Job)可拆成多個相同功能的任務(Task)，這些工作又分成 Parameter Server 與 Worker：

- **Parameter Server(ps):**主要根據梯度更新變數，並儲存於 `tf.Variable`，可理解成只儲存 TF Model 的變數，並存放 Variable 副本。
- **Worker:**通常稱為計算節點，主要執行密集型的 Graph 運算資源，並根據變數運算梯度。存放 Graph 副本。

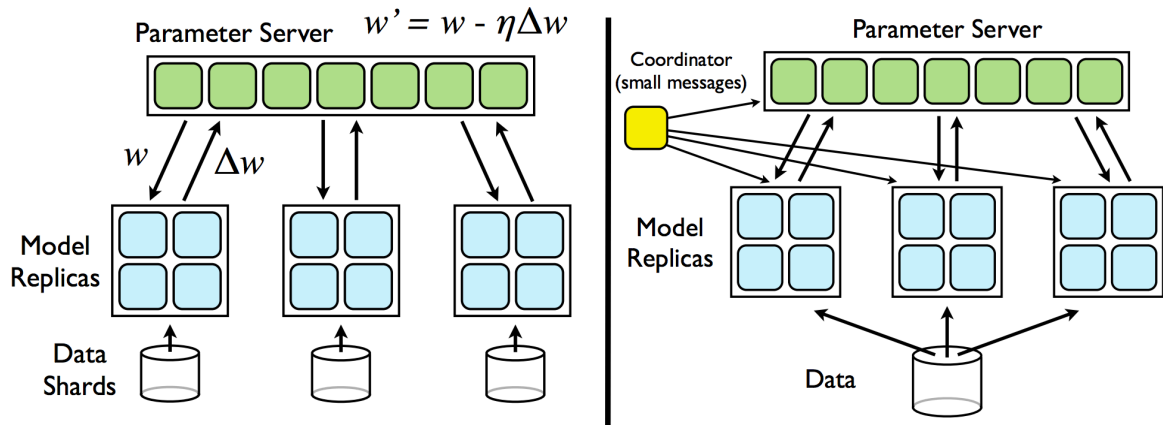
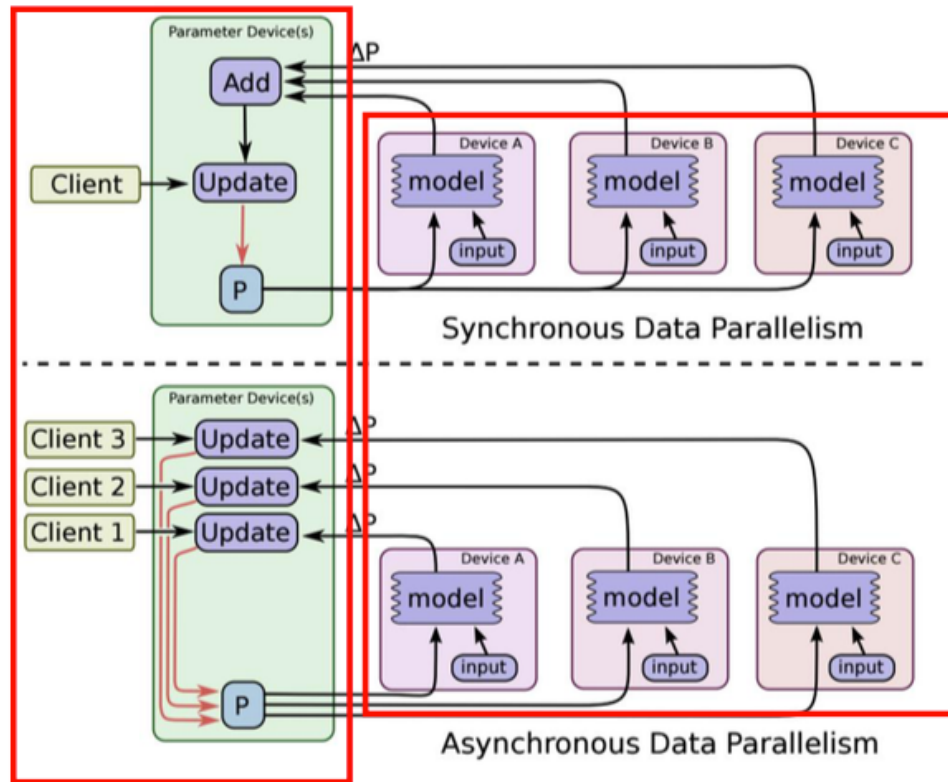


Image from: Large Scale Distributed Deep Networks

Replication and training mode

- **Replication**
 - In-graph
 - Between-graph
- **Training**
 - Synchronous
 - Asynchronous

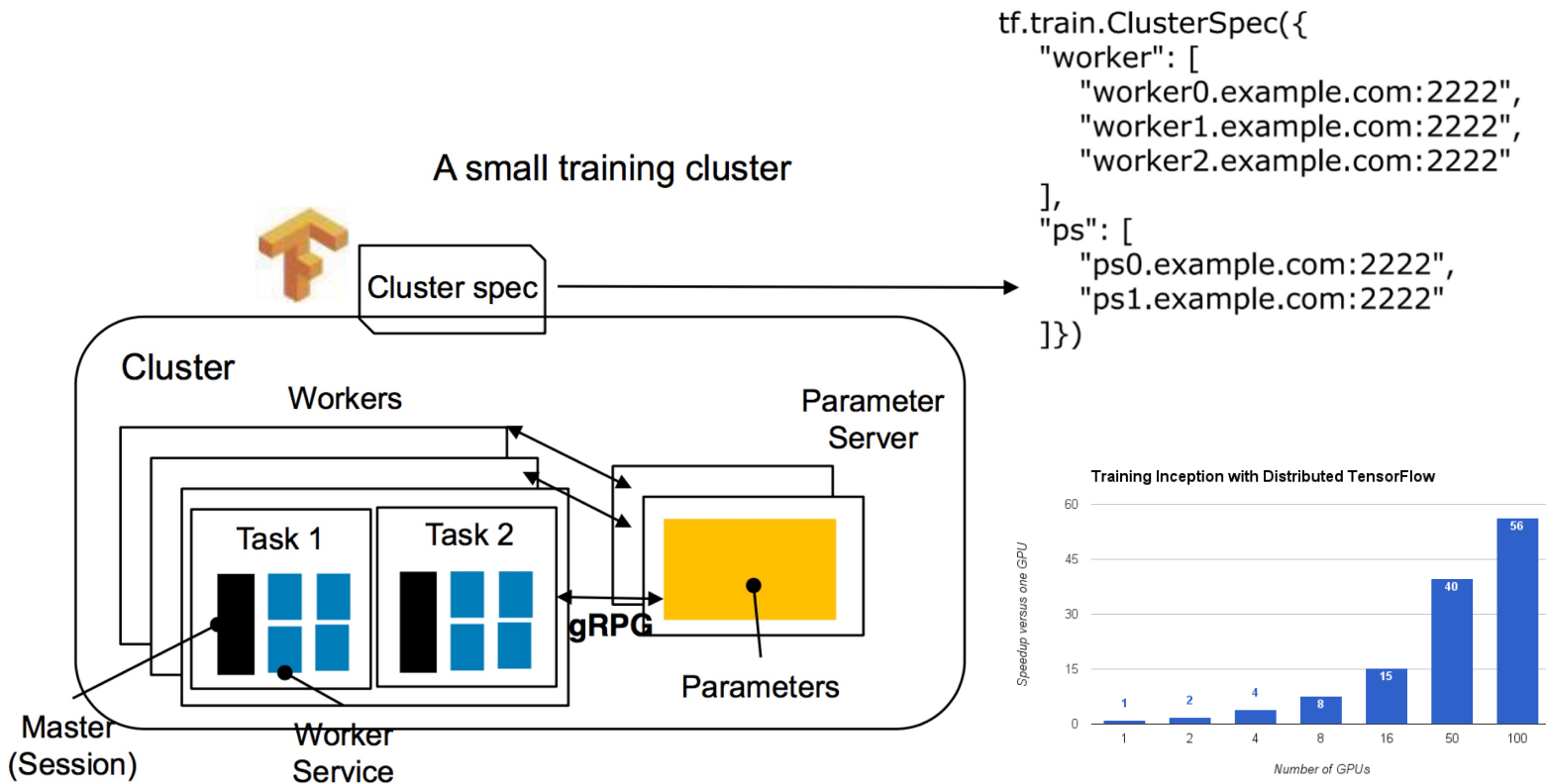


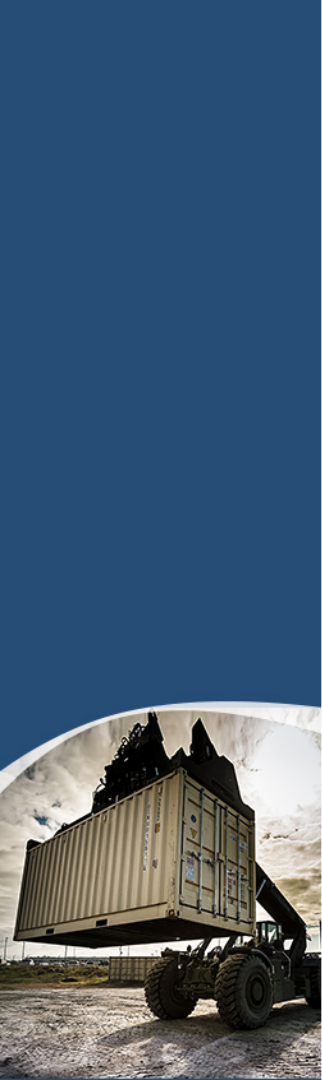
Distribution of different sizes

- **小型規模訓練**，這種資料與參數量不多時，可以用一個 CPU 來同時執行兩種任務。
- **中型規模訓練**，資料量較大，但參數量不多時，計算梯度的工作負載較高，而參數更新負載較低，所以計算梯度交給若干個 CPU 或 GPU 去執行，而更新參數則交給一個 CPU 即可。
- **大型規模訓練**，資料與參數量多時，不僅計算梯度需要部署多個 CPU 或 GPU，連更新參數也要不說到多個 CPU 中。



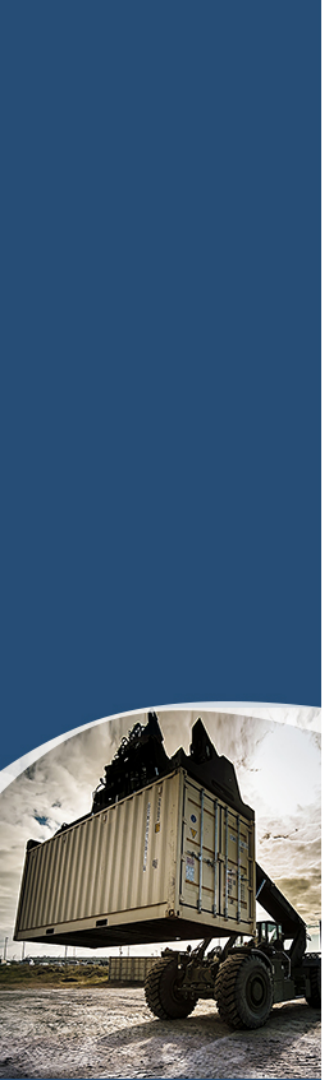
Example cluster spec





```
# in worker0
clusterSpec = tf.train.ClusterSpec({      // declare cluster spec
    "worker": ["192.168.1.100:2222", ...],
    "ps": ["192.168.1.101:2222", ...]})
server = tf.train.Server(clusterSpec, job_name = "worker", task_index = 0)
with tf.Session(server.target) as sess // create session using server.target
    # do something ...
```





```
# in ps 0
clusterSpec = tf.train.ClusterSpec({      // declare cluster spec
  "worker": ["192.168.1.100:2222", ...],
  "ps": ["192.168.1.101:2222", ...]})
server = tf.train.Server(clusterSpec, job_name = "ps", task_index = 0)
# wait for incoming connections ...
server.join()    // waiting for connection...
```



Distributed TensorFlow Workflow

We need shell script..? Or Ansible...?

Write the code

Copy to server

SSH to server1

OOM

SSH to server2

OOM

Run and Test

Port

Run and Test

Port

Disk

Disk

Memory

Copy to server

Update the code

Memory

...

...



然而生產環境中，我們將可能有多台機器需要被偏配，且環境可能是複雜的，需要網路、運算與儲存。

SO WE CHOOSE



kubernetes

MANAGED SERVICE PROVIDER





Why TensorFlow + Kubernetes?

在生產環境中，TensorFlow 的分散式運算並未支援資源與任務 scheduler，然而 Kubernetes 的特性與功能正好是一個最佳選擇。

Kubernetes

Kubernetes 是 Google 開源的**容器(Container)**分散式管理系統，是 Google 十幾年以來大規模應用容器技術的經驗累積和昇華的一個重要成果，是建於 Container(OCI標準容器)之上的容器叢集排程服務，簡稱為K8s('k' + 8 letters + 's')。

目前為 CNCF 專案之一。



kubernetes
MANAGED SERVICE PROVIDER



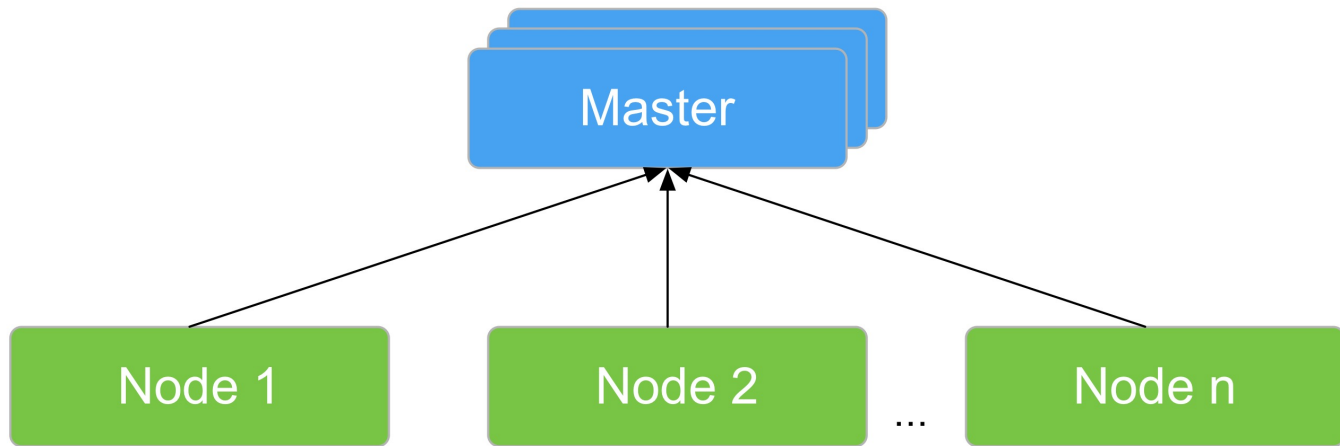
v0.9.4



Kubernetes Architecture

Kubernetes 屬於分散式架構系統，主要由兩種節點角色組成：

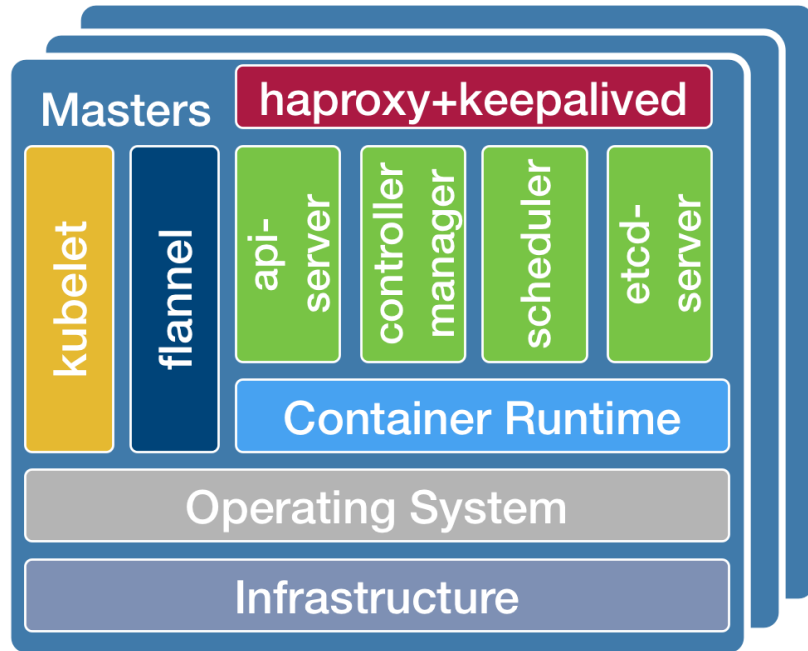
- **Masters** – 主要工作為提供 API 與管理工作節點，可視為**主節點**。
- **Nodes(Minions)** – 主要執行容器應用的節點，上面會執行許多容器。



Kubernetes Master

Kubernetes Master 包含了四個基本組件：

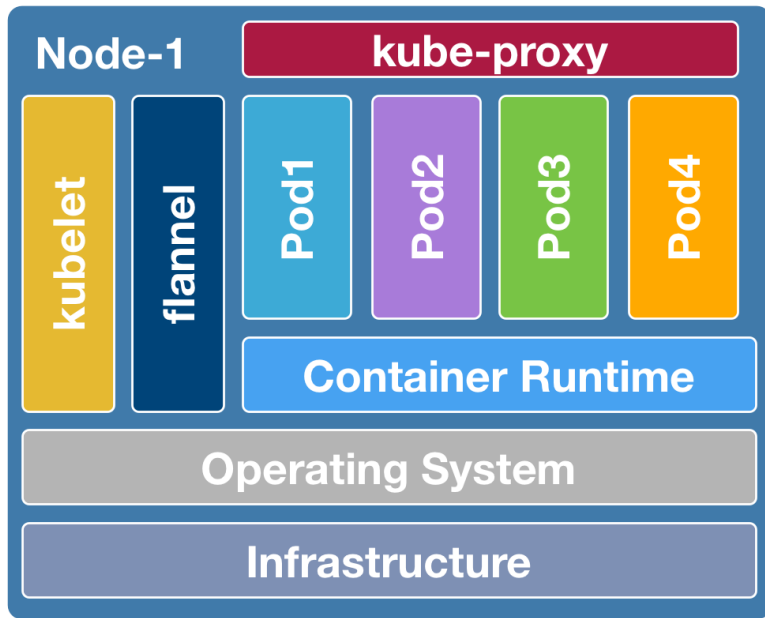
- **Etcd**: 主要為 Kubernetes 的資訊儲存與服務發現功能。
- **API Server**: 以 REST APIs 介面方式提供所有業務邏輯CURD操作。
- **Controller Manager Server**: 所有其他叢集級功能都是透過控制管理器 (Controller Manager)來操作。
- **Scheduler**: 負責整個分散式系統的資源排程。



Kubernetes Node(Minion)

Kubernetes Node 包含了三個基本組件：

- **Kubelet:**負責管理的映像檔、容器與資料 Volume等操作。也是連接 Master 的橋樑。
- **Proxy:** 為了解決外部網路能夠群曲跨機器叢集中容器提供的應用服務而設計。支援 TCP,UDP stream forwarding 或 round robin TCP,UDP forwarding。
- **Container:**基於 OCI Container Runtime 來執行應用程式容器實例。



Kubernetes Storage

- Volumes
- Persistent Volumes
- Many Volume Plugin
- Storage Class
- Dynamic Provisioning

Volume Plugin	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWElasticBlockStore	✓	-	-
AzureFile	✓	✓	✓
AzureDisk	✓	-	-
CephFS	✓	✓	✓
Cinder	✓	-	-
FC	✓	✓	-
FlexVolume	✓	✓	-
Flocker	✓	-	-
GCEPersistentDisk	✓	✓	-
Glusterfs	✓	✓	✓
HostPath	✓	-	-
iSCSI	✓	✓	-
PhotonPersistentDisk	✓	-	-
Quobyte	✓	✓	✓
NFS	✓	✓	✓
RBD	✓	✓	-
VsphereVolume	✓	-	-
PortworxVolume	✓	-	✓
ScaleIO	✓	✓	-
StorageOS	✓	-	-



Kubernetes Pros

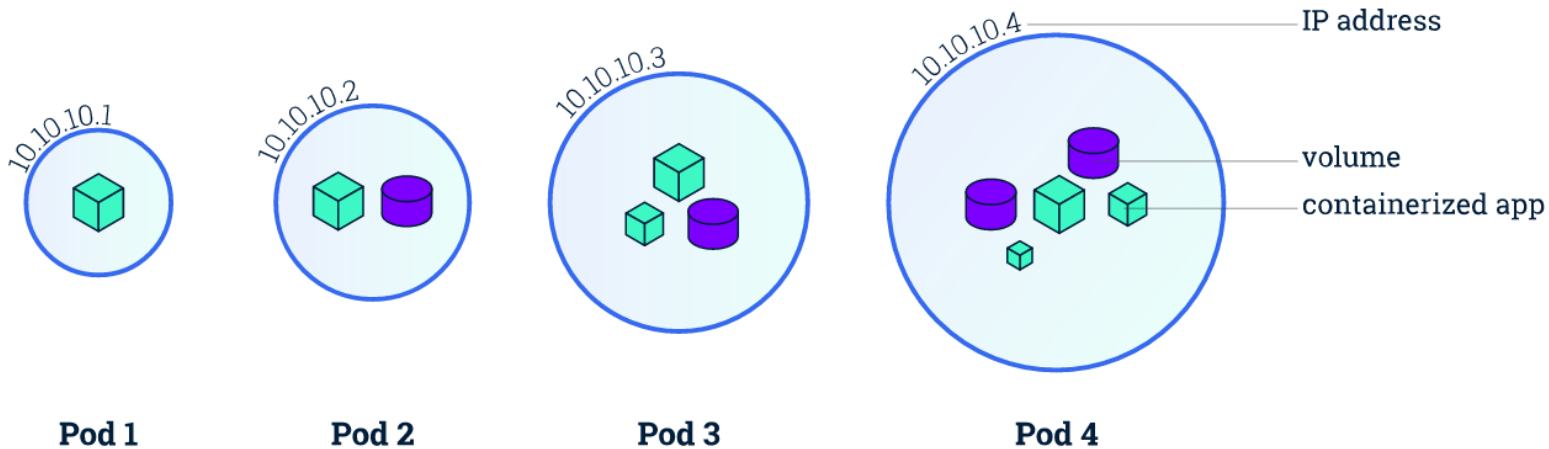
Kubernetes 管理跨區域與主機的容器節點，提供基本部署、維運、管理，以及執行各項應用程式。Kubernetes 為IT人員帶來以下幾項好處：

- 管理與操作簡單，不需要太多複雜元件設定，且支援 Heapster/Grafana/Influx 監控服務。
- 可管理與排程叢集資源。
- 簡單地擴展、遷移與升級Kubernetes元件。
- 提供負載平衡、容錯、**Namespace**、**Auto scale**與本地讀寫等功能。
- 支援各種雲端平台部署與作業系統，**No vendor lock-in**。
- 支援異質環境，如 CPU、GPU等。



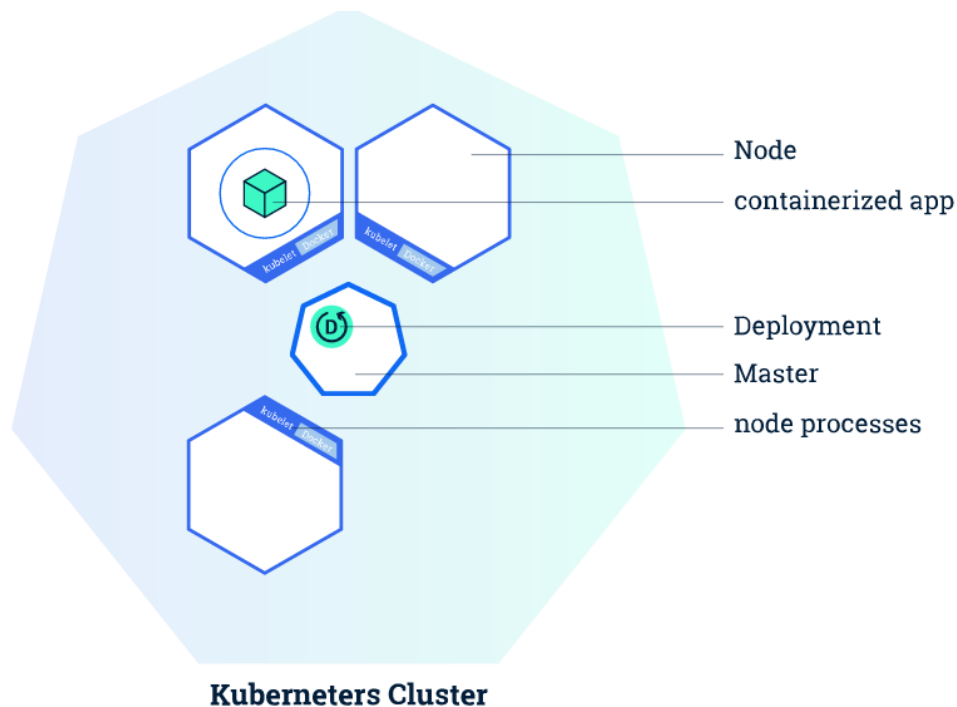
Kubernetes - Pod

Pod 是 K8s 中最小的部署單位，可以將一個或多個容器組成一個 Pod，Pod 所包含的容器只會運行在同一個 Host 上，並共享 network namespaces、IP port，每個 Pod 的 network mapping 都是由 container pause 所封裝處理的。



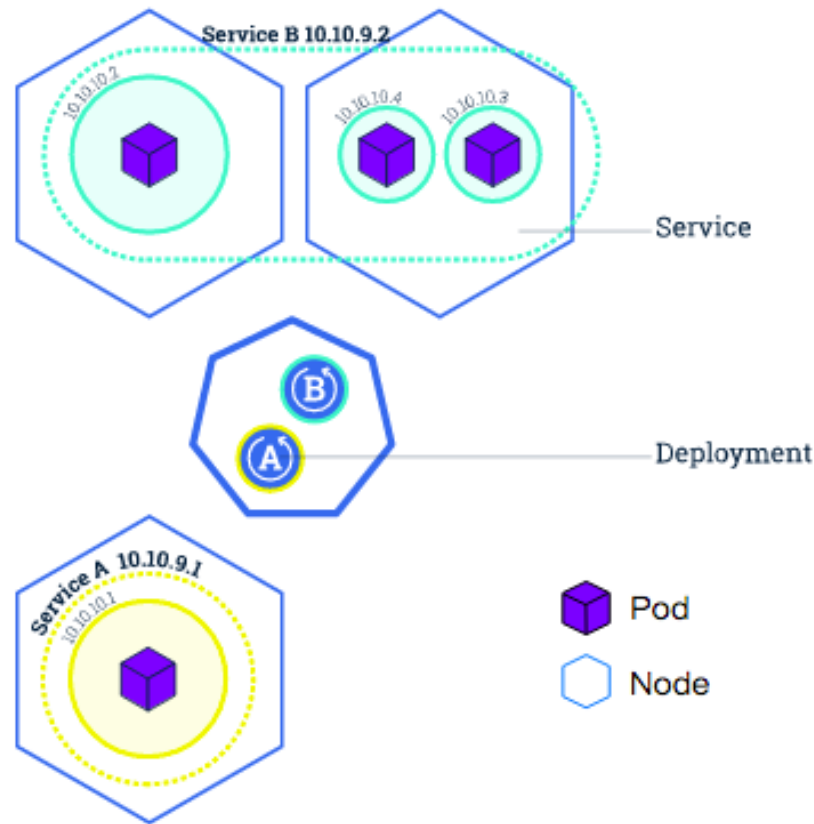
Kubernetes - Deployments

Deployments(Replication Controllers)是部署群組 Pods，其會確保 K8s 系統中指定數量副本的 Pod 存活，也可以利用 Label 及 Rolling update 來進行副本的擴展或更新。



Kubernetes - Service

Service 是 pod 服務的抽象層，具有簡單負載平衡的功能，由於 pod 的數量與IP 不會固定，可將一群 pod 指派成 service 並給予一個固定IP (cluster IP)，service 會將請求至 cluster IP 的流量導入其對應 pod。



TensorFlow on Kubernetes Pros(1)

叢集管理

- 適合大規模管理與擴展。
- 透過 Kubernetes DNS 機制來解析伺服器位址。
- 透過 Replication Controller 來管理故障重啟問題。
- Kubernetes 提供 Monitoring 與 Logging 等功能。
- 能夠支援使用 CPU 與 GPU 排程來指定節點。
- 可隱藏系統工程師與資料科學家之間的管道。



TensorFlow on Kubernetes Pros(2)

行程生命週期管理

- 透解決目前 TensorFlow 自動結束問題。
- 解決無法區分正常完成還是故障退出等問題。
- 解決需要手動管理行程問題。
- 模型服務版本控管。

共享儲存解決方案

- 可使用多種儲存服務，如 NFS、Ceph 等分散式儲存系統。



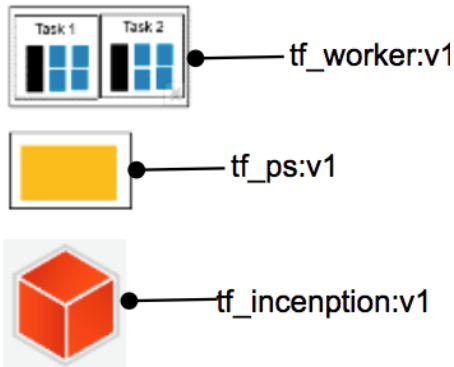


TensorFlow on Kubernetes

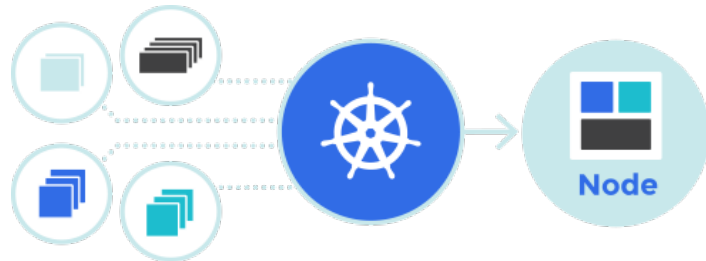
基於上述原因與需求使我們選擇利用 Kubernetes 來 TensorFlow 實作與管理分散式訓練與模型服務，接下來我們將針對一些簡單案例來說明如何透過 Kubernetes 進行 TensorFlow 叢集訓練。

Easy integrated with Kubernetes

Step1: Package Workers & Parameters
general code into Docker images.



Step3: Define the cluster Deployment and Service,..., etc to run.

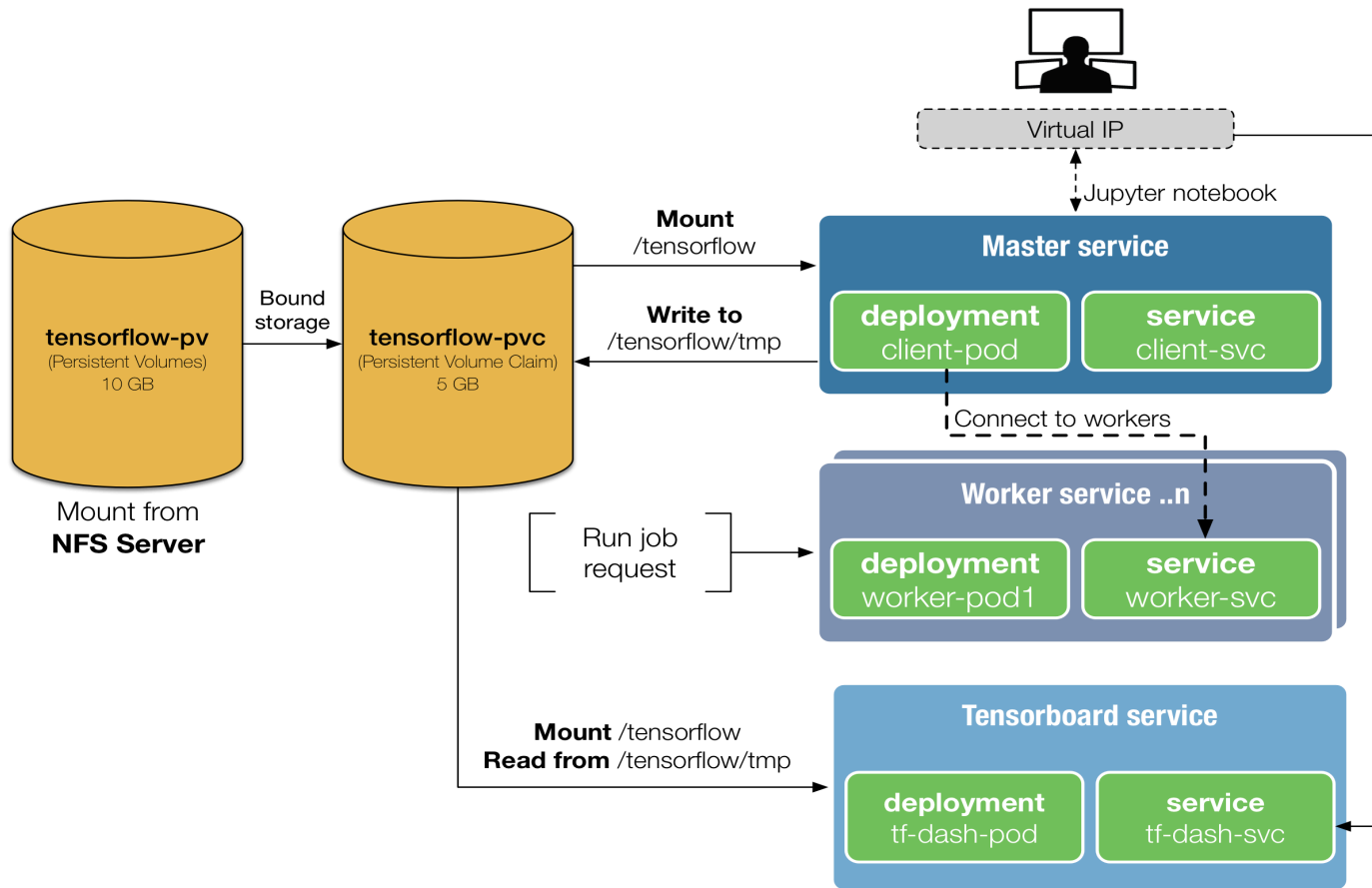


Step2: Create the clusters according to
TensorFlow's cluster spec into ConfigMap.

```
tf.train.ClusterSpec({  
  "worker": [  
    "worker0.example.com:2222",  
    "worker1.example.com:2222",  
    "worker2.example.com:2222"  
  ],  
  "ps": [  
    "ps0.example.com:2222",  
    "ps1.example.com:2222"  
  ]  
})  
  
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: cluster-config  
  namespace: default  
data:  
  ps: "ps0.default.svc.cluster.local:2222"  
  worker: "worker0.default.svc.cluster.local:2222,worker1.default.svc.cluster.local:2222"
```



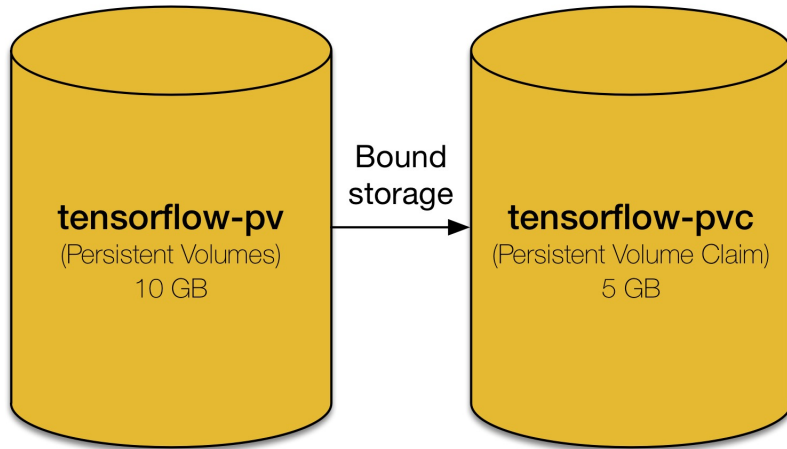
Simple TensorFlow on K8s



K8s PV and PVC

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: lab01-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    path: <Path>
    server: <NFS Server Address>
```

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: lab01-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

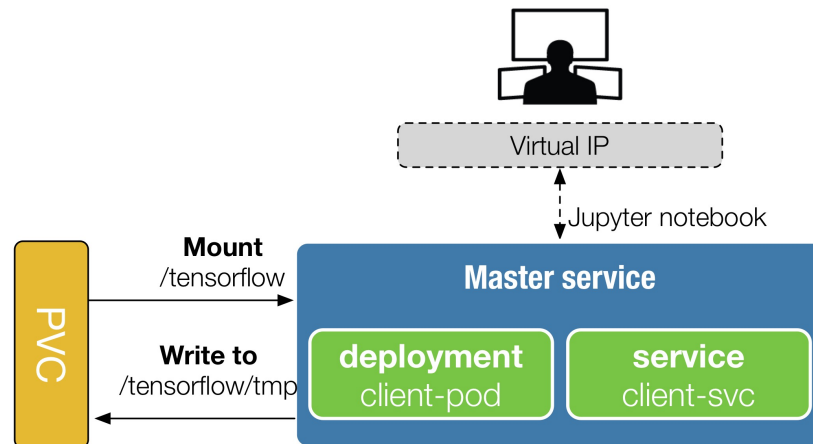


Mount from
NFS Server



Master service

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: tf-client
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: tf-client
    spec:
      containers:
        - name: tf-client
          image: example/tensorflow:1.2.1
          ports:
            - containerPort: 8888
          env:
            - name: TOKEN
              value: "p@ssw0rd"
          command: ["/bin/sh", "-c"]
          args:
            - "/run_jupyter.sh --NotebookApp.token=${TOKEN};"
          volumeMounts:
            - name: tmp
              mountPath: /tmp/train
      volumes:
        - name: tmp
          persistentVolumeClaim:
            claimName: tensorflow-pvc
```

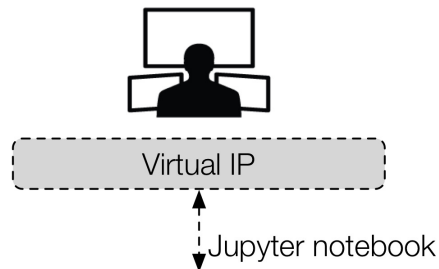


```
apiVersion: v1
kind: Service
metadata:
  name: tf-client
spec:
  selector:
    app: tf-client
  type: LoadBalancer
  externalIPs:
    - <external ip>
  ports:
    - port: 8888
      targetPort: 8888
```

Worker service

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: tf-worker
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: tf-worker
        role: worker
    spec:
      containers:
        - name: tf-worker
          image: example/tensorflow:1.2.1
          ports:
            - containerPort: 2222
          command: ["/bin/sh", "-c"]
          args:
            - "python /opt/basic_server.py;"
```

```
apiVersion: v1
kind: Service
metadata:
  name: tf-worker
spec:
  selector:
    app: tf-worker
  type: ClusterIP
  ports:
    - port: 2222
      targetPort: 2222
```



Master service

Send graph session to worker

Worker service ..n

deployment
worker-pod1

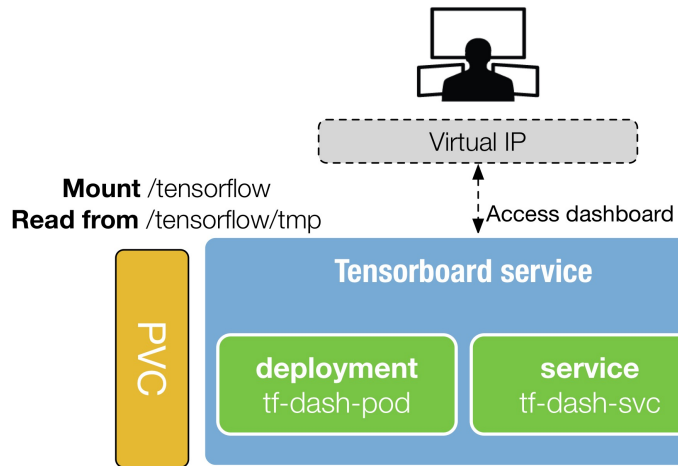
service
worker-svc



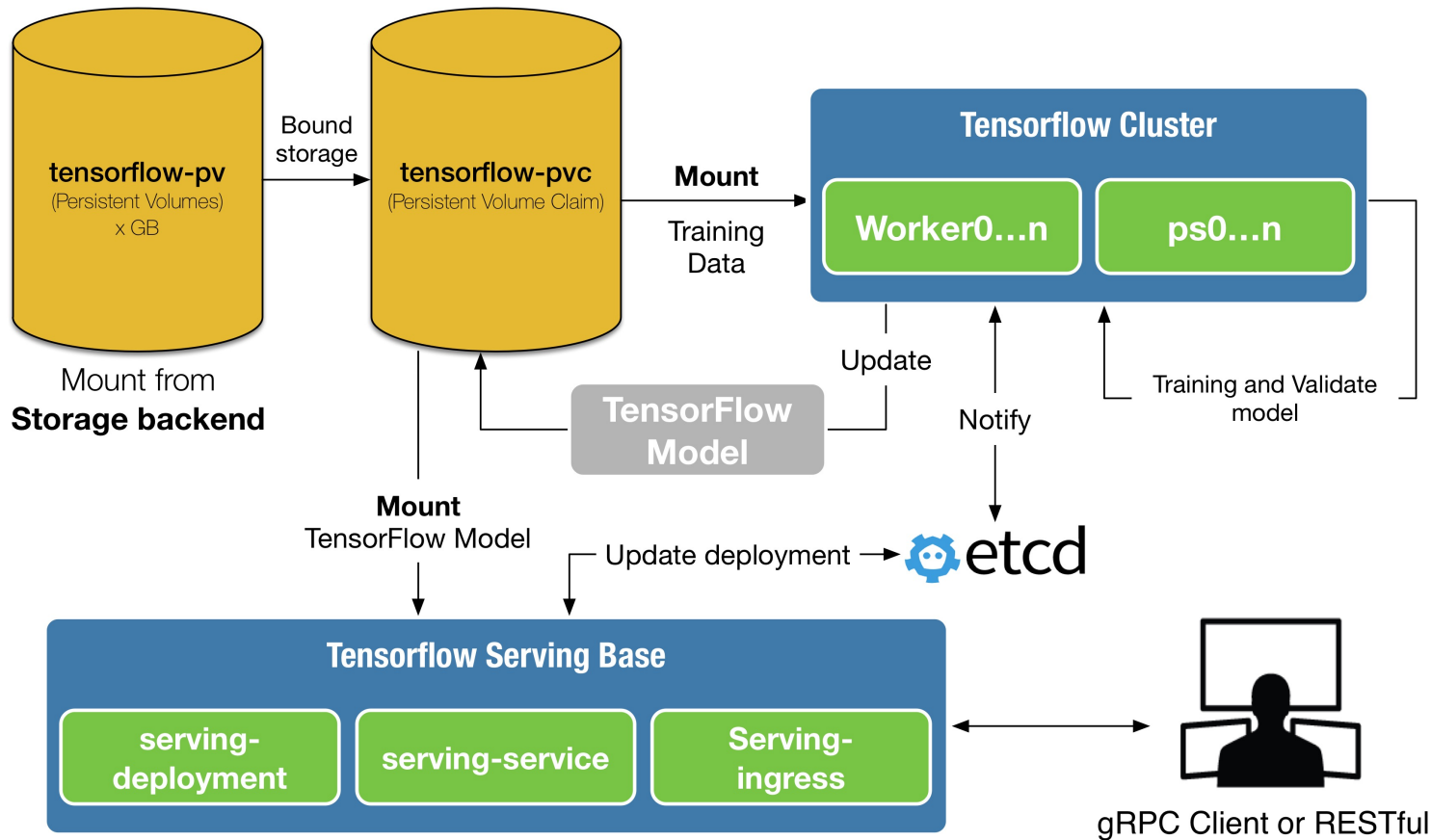
Tensorboard service

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: tf-dashboard
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: tf-dashboard
    spec:
      containers:
        - name: tf-dashboard
          image: example/tensorflow:1.2.1
          ports:
            - containerPort: 6006
          command: ["/bin/sh", "-c"]
          args:
            - "tensorboard --logdir=run1:/tmp/train --
port=6006"
          volumeMounts:
            - name: tmp
              mountPath: /tmp/train
          volumes:
            - name: tmp
              persistentVolumeClaim:
                claimName: tensorflow-pvc
```

```
apiVersion: v1
kind: Service
metadata:
  name: tf-dashboard
spec:
  selector:
    app: tf-dashboard
  type: LoadBalancer
  externalIPs:
    - <external ip>
  ports:
    - port: 6006
      targetPort: 6006
```



Simple TensorFlow Serving on K8s



Serving service

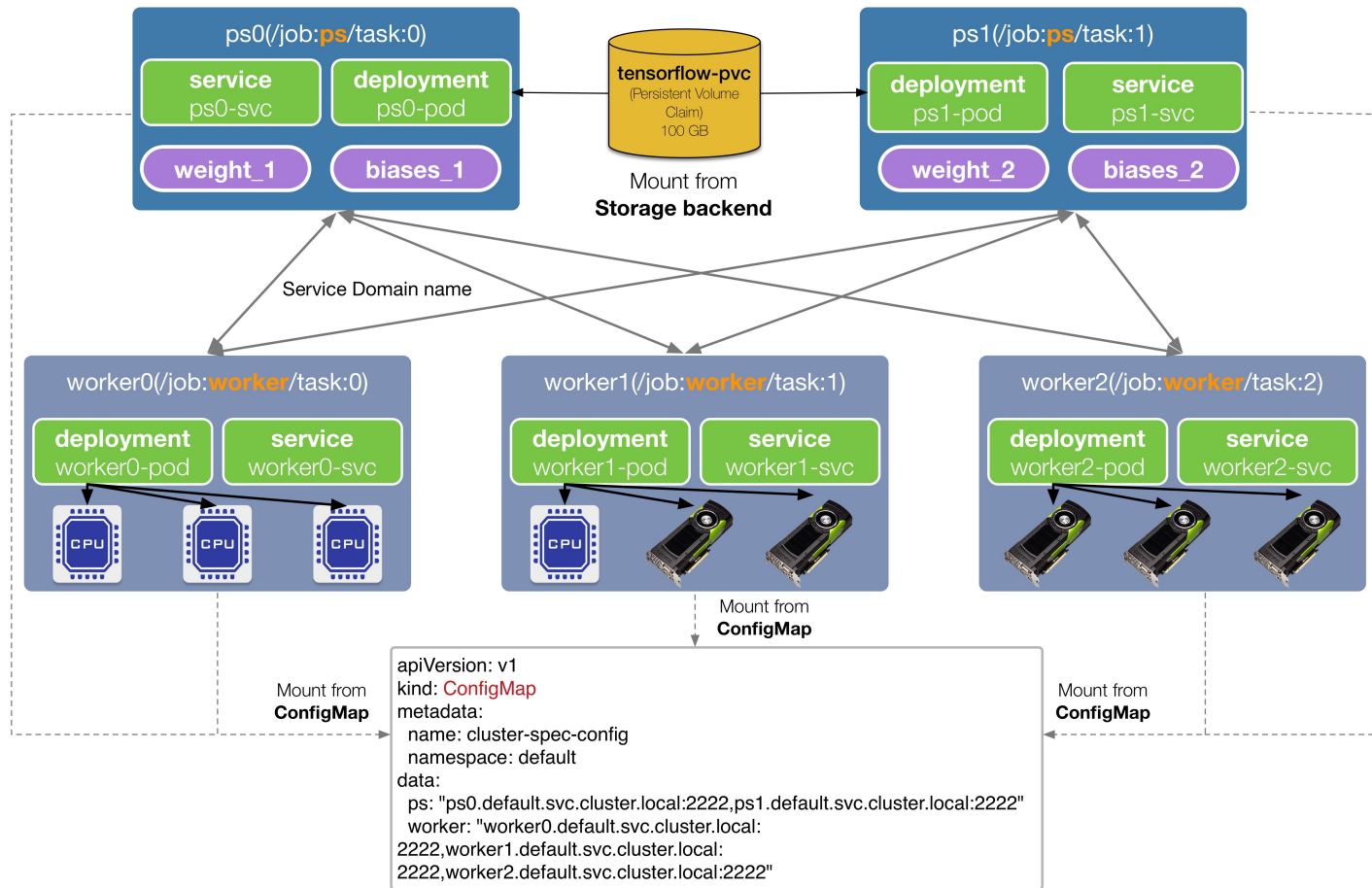
```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: serving
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: serving
    spec:
      containers:
      - name: serving
        image: example/serving-base:0.5.1
        ports:
        - containerPort: 9000
        resources:
          requests:
            memory: "128Mi"
            cpu: "500m"
          limits:
            memory: "512Mi"
            cpu: "2000m"
        command: ["/bin/sh", "-c"]
        args:
        - "cd /opt/serving;"
        - "bin/example/inception_saved_model --checkpoint_dir=/tmp/model-data --output_dir=inception-export;"
        - "bin/model_servers/tensorflow_model_server --port=9000 --model_name=inception --model_base_path=inception-export;"
      volumeMounts:
      - name: model-data
        mountPath: /tmp/model-data
    volumes:
    - name: model-data
      persistentVolumeClaim:
        claimName: model-pvc
```

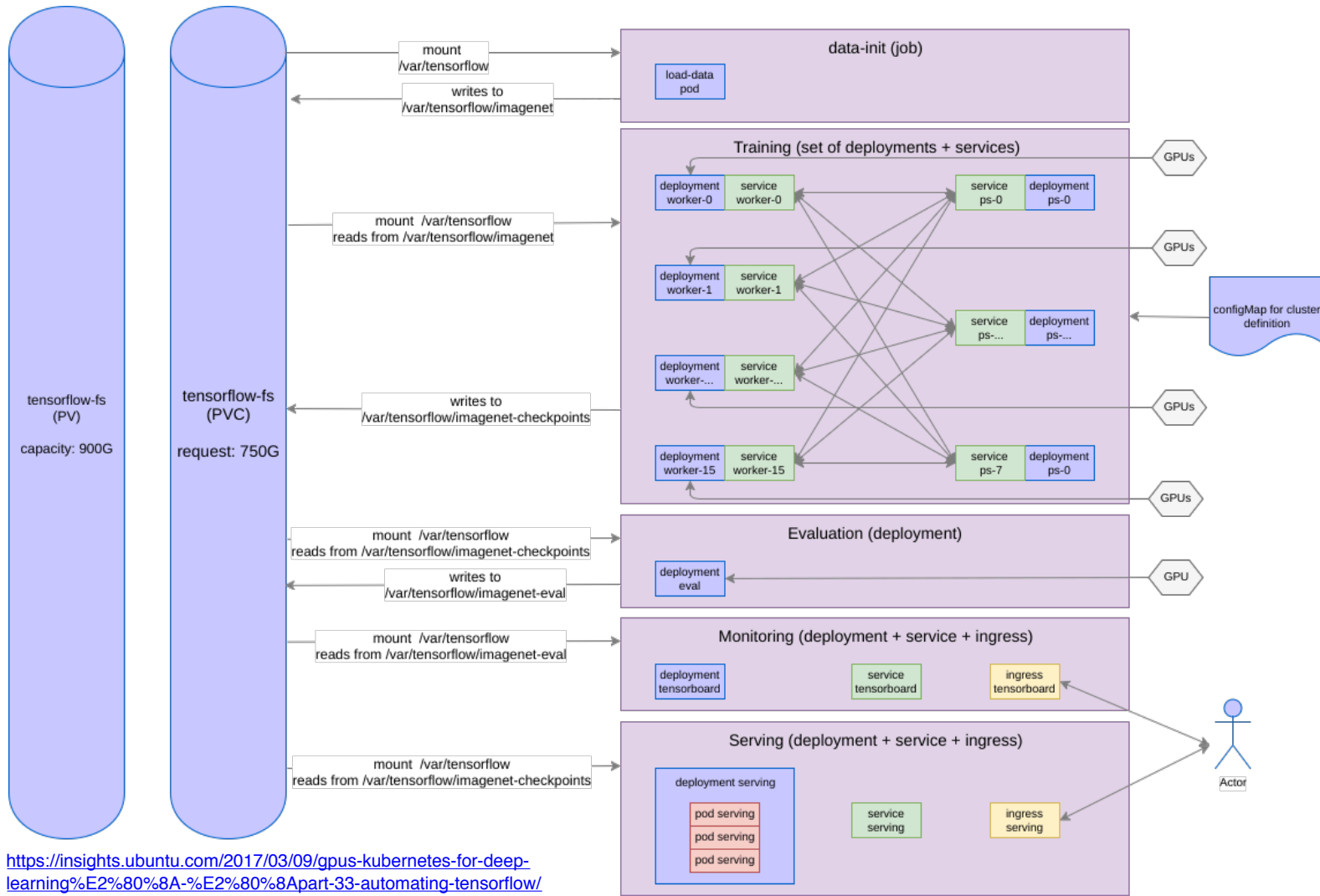
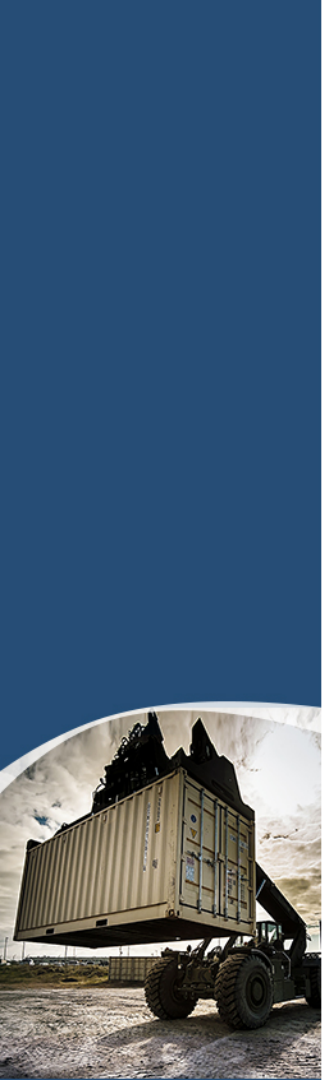
```
apiVersion: v1
kind: Service
metadata:
  labels:
    name: serving-service
    name: serving
spec:
  selector:
    app: serving
  type: LoadBalancer
  externalIPs:
  - <external ip>
  ports:
  - port: 9000
    targetPort: 9000
```

```
apiVersion: extensions/
v1beta1
kind: Ingress
metadata:
  name: serving-ingress
spec:
  rules:
  - http:
      paths:
      - path: /inception
        backend:
          serviceName:
serving-service
          servicePort: 9000
```



Full Distributed on K8s





Kubernetes with GPU enabled

在 Kubernetes 中使用 GPU 需要預先完成以下幾點：

- 在所有 Node 上安裝 Nvidia 驅動程式，包含 CUDA Toolkit 與 cuDNN 等。
- 在 Master 的 apiserver 和 Node 的 kubelet 上開啟 **--feature-gates="Accelerators=true"**。
- 使用 Docker 作為容器引擎，目前其他引擎不支援 GPU 特性。
- 接著在 Pod 掛載 nvidia-driver、libcuda-so、libcuda-so.1 等。

volumes:

```
- name: bin
  hostPath:
    path: /usr/lib/nvidia-375/bin
- name: lib
  hostPath:
    path: /usr/lib/nvidia-375
- name: libcuda-so
  hostPath:
    path: /usr/lib/x86_64-linux-gnu/libcuda.so
- name: libcuda-so-1
  hostPath:
    path: /usr/lib/x86_64-linux-gnu/libcuda.so.1
- name: libcuda-so-375-66
  hostPath:
    path: /usr/lib/x86_64-linux-gnu/libcuda.so.375.66
```

volumeMounts:

```
- mountPath: /usr/local/nvidia/bin
  name: bin
- mountPath: /usr/lib/nvidia
  name: lib
- mountPath: /usr/lib/x86_64-linux-gnu/libcuda.so
  name: libcuda-so
- mountPath: /usr/lib/x86_64-linux-gnu/libcuda.so.1
  name: libcuda-so-1
- mountPath: /usr/lib/x86_64-linux-gnu/libcuda.so.375.66
  name: libcuda-so-375-66
```



GPU Pod Examples

```
apiVersion: v1
kind: Pod
metadata:
  name: tensorflow-gpu
spec:
  restartPolicy: Never
  containers:
  - image: gcr.io/tensorflow/tensorflow:latest-gpu
    name: gpu-container-1
    command: ["python"]
    env:
    - name: LD_LIBRARY_PATH
      value: /usr/lib/nvidia
    args:
    - -u
    - -c
    - from tensorflow.python.client import device_lib; print device_lib.list_local_devices()
  resources:
    limits:
      alpha.kubernetes.io/nvidia-gpu: 1 # requests one GPU
  volumeMounts:
  ...
volumes:
  ...
```



GPU enabled limit

- GPU 資源必須在 `resources.limits` 設定，在 `resources.requests` 中會無效。
- 一個容器可以使用一個或多個 GPU，但不能只使用一部分。
- 多容器之間無法共享使用同一張 GPU。
- 預設假設環境的所有 Node 都安裝相同型號 GPU。



Different GPU card(1)

假設叢集 Node 安裝了不同型號的 GPU，則可以透過 Kubernetes **Node affinity** 或 **Node selector** 來排成 Pod 指定到對應 GPU 型號的 Node 上執行。

首先在所有 Kubernetes Node 中，加入 nvidia-gpu-name 標籤，該標籤定義使用的型號：

```
$ NVIDIA_GPU_NAME=$(nvidia-smi --query-gpu=gpu_name --  
format=csv,noheader -id=0)
```

```
$ source /etc/default/kubelet  
$ KUBELET_OPTS="$KUBELET_OPTS --node-  
labels='alpha.kubernetes.io/nvidia-gpu-  
name=$NVIDIA_GPU_NAME'"
```

```
$ echo "KUBELET_OPTS=$KUBELET_OPTS" > /etc/default/kubelet
```

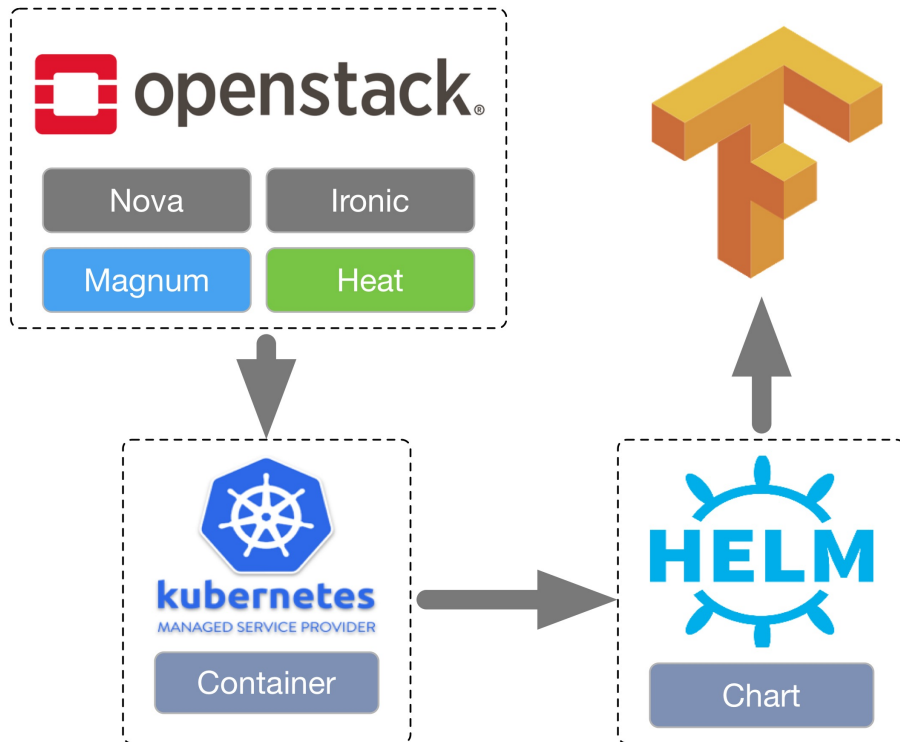


Different GPU card(2)

```
kind: pod
apiVersion: v1
metadata:
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
    {
      "nodeAffinity": {
        "requiredDuringSchedulingIgnoredDuringExecution": {
          "nodeSelectorTerms": [
            {
              "matchExpressions": [
                {
                  "key": "alpha.kubernetes.io/nvidia-gpu-name",
                  "operator": "In",
                  "values": ["Tesla K80", "Tesla P100"]
                }
              ]
            }
          ]
        }
      }
    }
spec:
  containers:
    - image: gcr.io/tensorflow/tensorflow:latest-gpu
      name: gpu-container-1
      command: ["python"]
      args: ["-u", "-c", "import tensorflow"]
      resources:
        limits:
          alpha.kubernetes.io/nvidia-gpu: 2
```



Integration Options



- Bare metal or Virtual machine.
- Auto-scale the Kubernetes Cluster.
- Helm-based Application deployment on Kubernetes.
- Multi-tenant environment.
- Manage multi Kubernetes cluster.

**WE'RE
HIRING**



迎棧科技股份有限公司



www.inwinstack.com

Thank You!!



迎棧科技股份有限公司



www.inwinstack.com