How To Build Efficient ML Pipelines

From the Startup Perspective

Jaeman An <jaeman@aitrics.com>

GPU Technology Conference, 2019

AITRICS

Machine Learning Pipelines

- Challenges that many fast-growing startups face
- Solutions we came up with
- Several tools and tips that may be useful for you : kubernetes, polyaxon, kubeflow, terraform, ...
- Way to build your own training farm by step by step
- How to deploy & manage trained model by step by step

01 Why we built a ML pipeline

02 Brief introduction to kubernetes

03 Model building & training phase

- Building training farm from zero (step by step)
- Terraform, Polyaxon

04 Model deployment & production phase

- Building inference farm from zero (step by step)
- Several ways to make microservices
- Kubeflow

05 Conclusion

06 What's next?

AITRICS

Why we built a ML pipeline

Very simple way to start machine learning startup

- Buy GPU machines
- Build (Explore) your own models
- Train models
- Freeze and deploy as as service
- Conduct fitting and re-training
- **Earn money and exit**

Data refining

Model building

Training

Deploying

Fitting, re-training

Very simple way to start machine learning startup

- Buy GPU machines
- Build (Explore) your own models
- Train models
- Freeze and deploy as as service
- Conduct fitting and re-training
- Earn money and exit

Data refining

Model building

Training

Deploying

Fitting, re-training

What's going on in data refining phase

- Mostly time-consuming job
- Sometimes we need to do large-scale data processing
  - Use Apache Spark!
    (This won't be covered in this talk)
- We've not handle real-time data *yet*
  - Kafka Streams is feasible solution
    (This won't be covered in this talk)
- Have to manage several data versions
  - due to sampling policies and operational definitions (labeling)
  - Can use Git-like solutions
- It'll be great to import data easily in the training phase like
  - `./train --data=images_v1`
- Permission Control

Data refining

Model building

Training

Deploying

Fitting, re-training

What's going on in model building phase

- Referring tons of precedent research
- Pick a simple model for baseline with small set of data
  - Check minimal accuracy and debug our model
  - (if data matters) refining data more precisely
  - (if model matters) iteratively improve our model
- Mostly only need GPU instance or notebook and small datasets; don't want to care about other stuffs!
  - `./run-notebook tf-v12-gpu --gpu=4 --data=images_v1`
  - `./ssh tf-v12-gpu --gpu=2 --data=images_v1`

Data refining

Model building

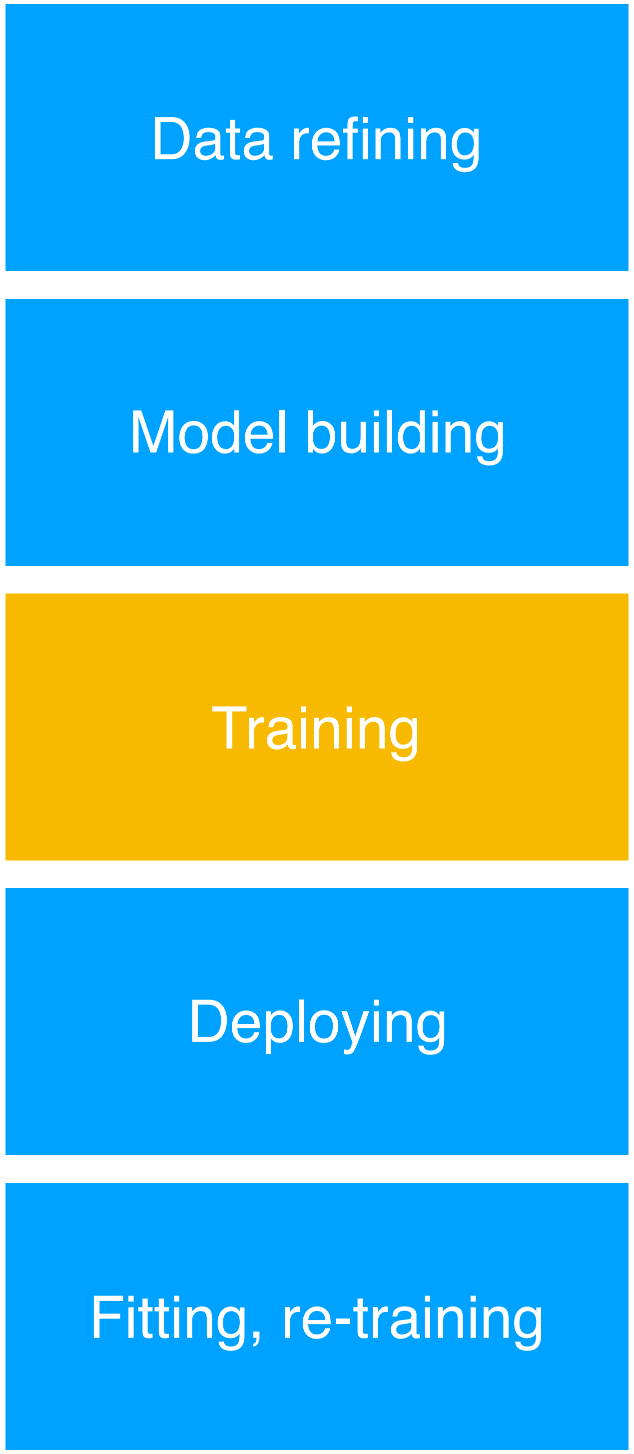
Training

Deploying

Fitting, re-training

What's going on in training phase

- Training on large datasets
- Researchers have to "hunt" idle GPU resources by accessing 10+ servers <u>one by one</u>
  - **Scalability**: Sometimes there’s no idle GPU resources (depends on product timeline / paper deadline)
  - **Access Control**: Sometimes all resources are occupied by outside collaborators
  - **Data accessibility**: Fetching / moving training data servers to servers is very painful!
  - **Monitoring**: Want to know how our experiments are going and what's going on our resources



What's going on in deploying phase

- In the middle of machine learning engineering and software engineering
- Want to manage model independently for the product
- Build micro-services that inference test data synchronously / asynchronously
- Have to consider high availability on production usage

Data refining

Model building

Training

Deploying

Fitting, re-training

What's going on to us in fitting phase

- Data distribution always changes; therefore, have to keep fitting the model with the real data
- Want to easily change the model code interactively
- Try to build online-learning model or re-training model in certain schedule
- Sometimes need to create real time data flow with Kafka
- Have to manage several model versions
  - As new models are developed
  - As the usage varies

Data refining

Model building

Training

Deploying

Fitting, re-training

Problems and requirements

- Model building & training phase:
  - We need to know the status of resources without access to our physical servers one by one.
  - We want to use easily idle GPU with proper training datasets
  - We have to control permissions of our resources and datasets
  - We only want to mainly focus on our research: developing innovative models, conducting experiments and such, ... not infrastructures

Problems and requirements

- Model deploying & updating phase:
  - It's hard to control because it is in the middle of machine learning engineering and software engineering
  - We want to create simple micro-services that don't need much management
  - There are many models with different purposes;
    - some models need real-time inference
    - some models do not require real-time, but they need inference in the certain time range
  - We have to consider high availability configuration
  - Models must be fitted and re-trained easily
  - We have to manage several versions of models

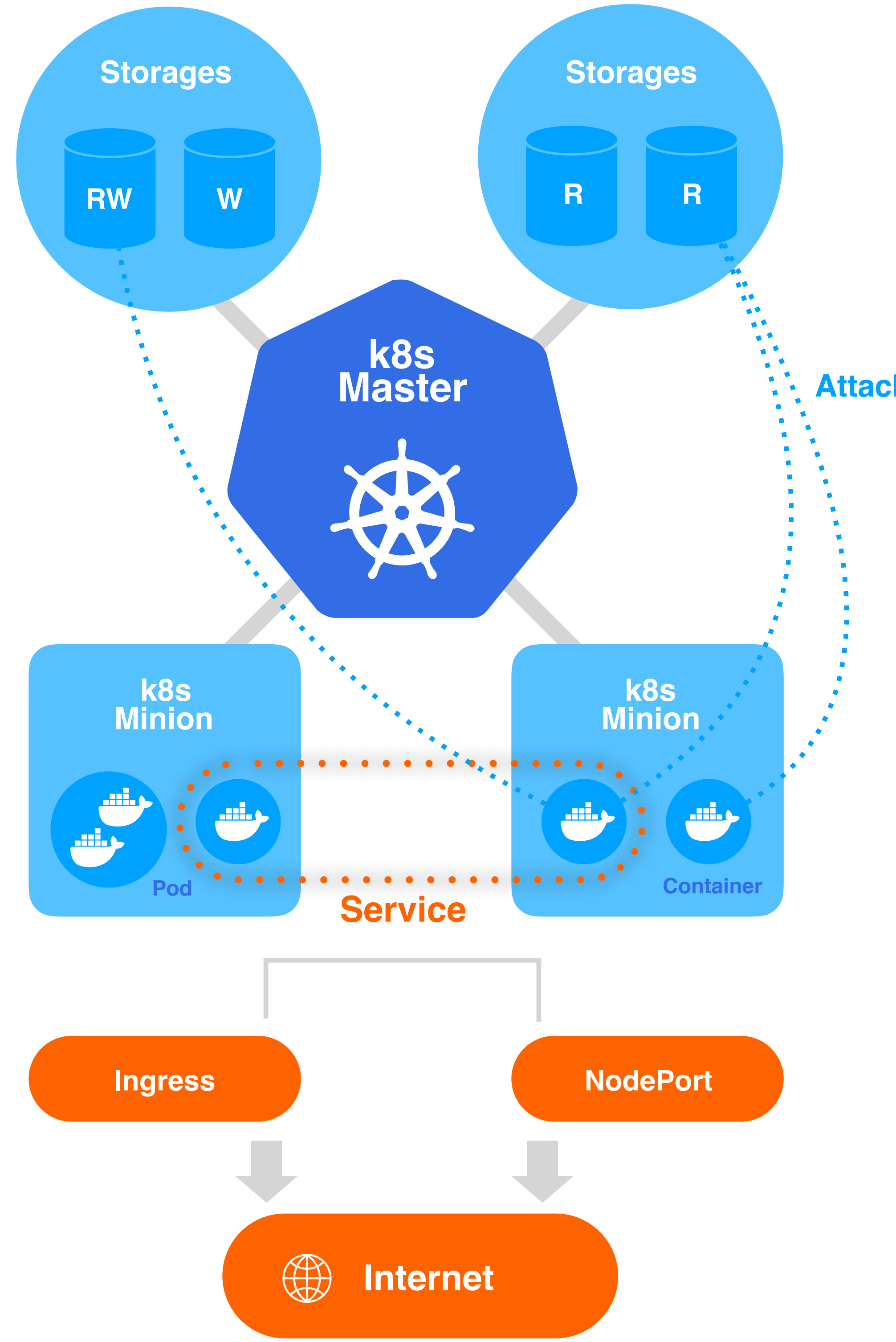
How to solve

- Managing resources over multiple servers, deploying microservices, permission controls, ...
- These can be solved with orchestration solutions.
- We are going to build training farm using kubernetes.
- Before that, what is kubernetes?

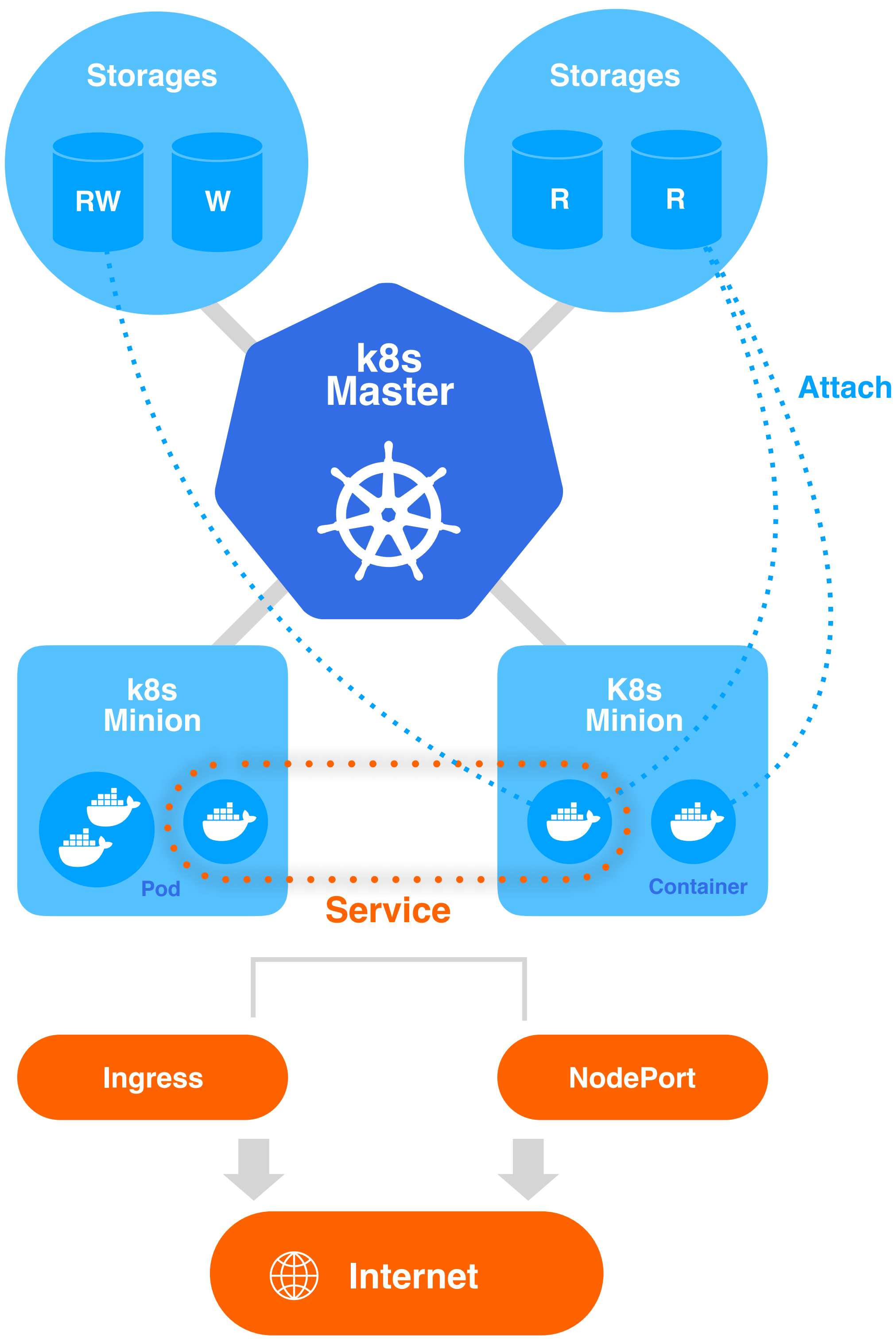
Kubernetes in 5 minutes

Kubernetes

- Kubernetes (k8s) is an open-source system for automating deployment, scaling, and management of containerized applications.
- It orchestrates computing, networking, and storage infrastructure on behalf of user workloads.
- NVIDIA GPU also can be orchestrated through NVIDIA's k8s device plugin



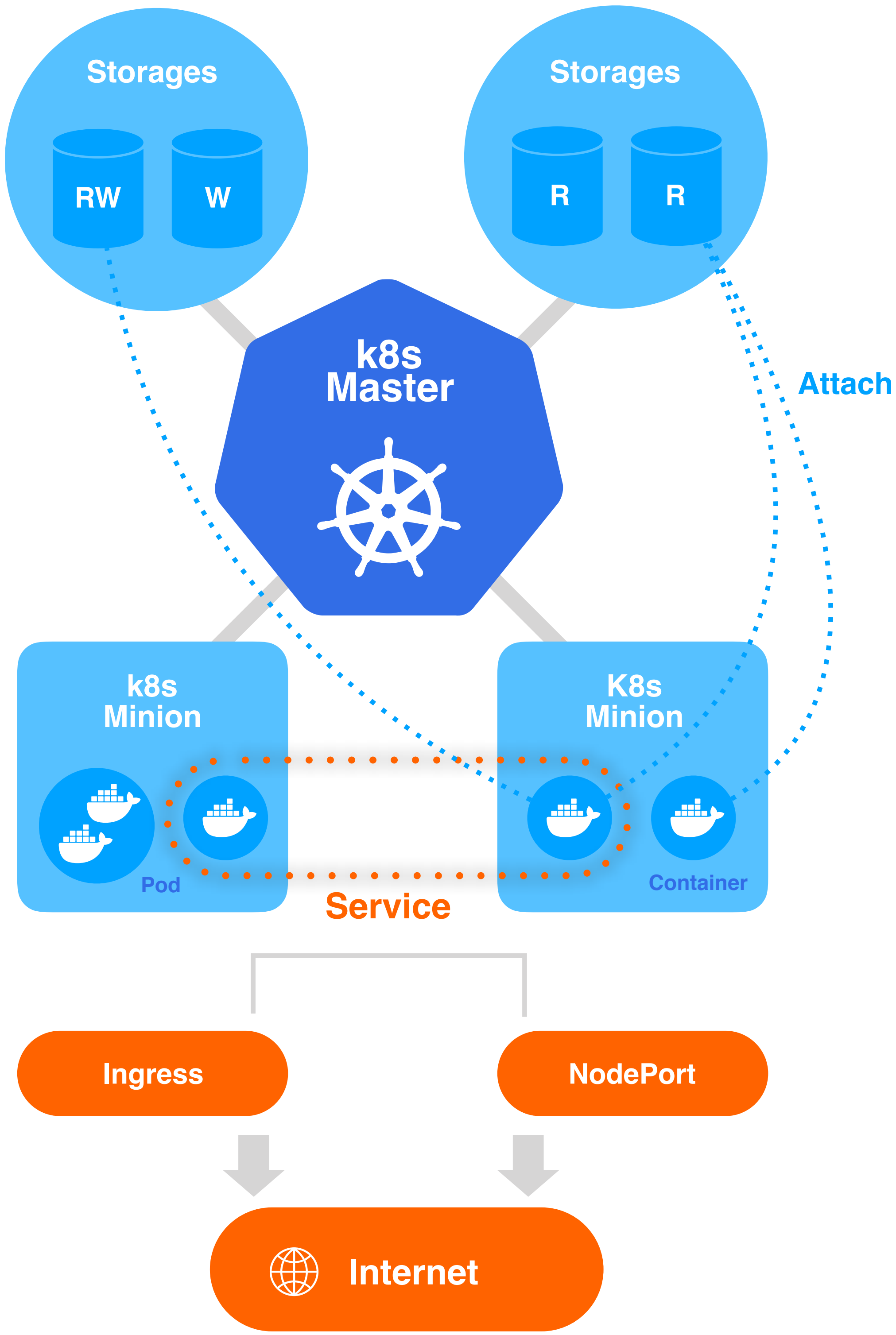
Kubernetes



- Give me 4 CPU, 1 Memory, 1 GPU
- I'm Jaeman An, and I'm in team A namespace
- With 4 External Port
- With abcd.aitrics.com hostname
- With latest gpu tensorflow image
- With 100GB writable volumes and data from readable source

- OK, Here you are
- No, you have no permission
- No, you've already use resources that you can
- No, there's no idle resources, please wait

Kubernetes



Workload & Services
Pod
Service
Ingress
Deployment
Replication Controller
...

Storage Class
PersistentVolume
PersistentVolumeClaim
...

Workload Controllers
Job
CronJob
ReplicaSet
RepliactionController
DaemonSet
...

<Objects>

Namespace
Role & Authorization
Resource Quota
...

<Meta & Policies>

AI TRICS

Kubernetes

Workload & Services
Pod
Service
Ingress
Deployment
Replication Controller
...

Storage Class
PersistentVolume
PersistentVolumeClaim
...

Workload Controllers
Job
CronJob
ReplicaSet
RepliactionController
DaemonSet
...

A Pod is the basic building block of Kubernetes – the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents a running process on your cluster.

```
kind: Pod
metadata:
  name: gpu-pod
spec:
  containers:
  - name: cuda-container
    image: nvidia/cuda:9.0-base
    resources:
      limits:
        nvidia.com/gpu: 1 # requesting 1 GPU
    command: ["nvidia-smi"]
```

Ref: https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/

Kubernetes

Workload & Services
Pod
Service
Ingress
Deployment
Replication Controller
...

Storage Class
PersistentVolume
PersistentVolumeClaim
...

Workload Controllers
Job
CronJob
ReplicaSet
RepliactionController
DaemonSet
...

A Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service.

```yaml
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

Ref: https://kubernetes.io/docs/concepts/services-networking/service/

Kubernetes

Workload & Services
Pod
Service
Ingress
Deployment
Replication Controller
...

Storage Class
PersistentVolume
PersistentVolumeClaim
...

Workload Controllers
Job
CronJob
ReplicaSet
RepliactionController
DaemonSet
...

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

```yaml
kind: Ingress
metadata:
  name: test-ingress
spec:
  rules:
  - host: foo.bar.com
  - http:
      paths:
      - backend:
          serviceName: MyService
          servicePort: 80
```

Ref: https://kubernetes.io/docs/concepts/services-networking/ingress/

Kubernetes

Workload & Services
Pod
Service
Ingress
Deployment
Replication Controller
...

Storage Class
PersistentVolume
PersistentVolumeClaim
...

Workload Controllers
Job
CronJob
ReplicaSet
RepliactionController
DaemonSet
...

A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator. It is a resource in the cluster just like a node is a cluster resource.

```yaml
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  nfs:
    path: /tmp
    server: 172.17.0.2
```

Ref: https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Kubernetes

Workload & Services
Pod
Service
Ingress
Deployment
Replication Controller
...

Storage Class
PersistentVolume
PersistentVolumeClaim
...

Workload Controllers
Job
CronJob
ReplicaSet
RepliactionController
DaemonSet
...

A PersistentVolumeClaim (PVC) is a request for storage by a user. Claims can request specific size and access modes (e.g., can be mounted once read/write or many times read-only).

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
```

Ref: https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims

Workload & Services
Pod
Service
Ingress
Deployment
Replication Controller
...

Storage Class
PersistentVolume
PersistentVolumeClaim
...

Workload Controllers
Job
CronJob
ReplicaSet
RepliactionController
DaemonSet
...

A Job creates one or more Pods and ensures that a specified number of them successfully terminate. As pods successfully complete, the Job tracks the successful completions.

```
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-
wle", "print bpi(2000)"]
```

Ref: https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims

Policies & Others

Namespace

Resource Quota

Role & Authorization

...

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces. Those are intended for use in environments with many users spread across multiple teams, or projects.

```
$ kubectl get namespaces

NAME          STATUS    AGE
default       Active    1d
kube-system   Active    1d
kube-public   Active    1d
```

Ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

Kubernetes

Policies & Others

Namespace

Resource Quota

Role & Authorization

...

A *resource quota*, defined by a ResourceQuota object, provides constraints that limit aggregate resource consumption per namespace.

```yaml
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    requests.nvidia.com/gpu: 1
```

Ref: https://kubernetes.io/docs/concepts/policy/resource-quotas/

Policies & Others
Namespace
Resource Quota
Role & Authorization
...

In Kubernetes, you must be authenticated (logged in) before your request can be authorized (granted permission to access).

Kubernetes uses client certificates, bearer tokens, an authenticating proxy, or HTTP basic auth to authenticate API requests through authentication plugins.

Ref: https://kubernetes.io/docs/reference/access-authn-authz/authentication/

Policies & Others

Namespace

Resource Quota

Role & Authorization

...

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise.

```
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
group:
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

Ref: https://kubernetes.io/docs/reference/access-authn-authz/rbac/

Policies & Others
Namespace
Resource Quota
Role & Authorization
…

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise.

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Ref: https://kubernetes.io/docs/reference/access-authn-authz/rbac/

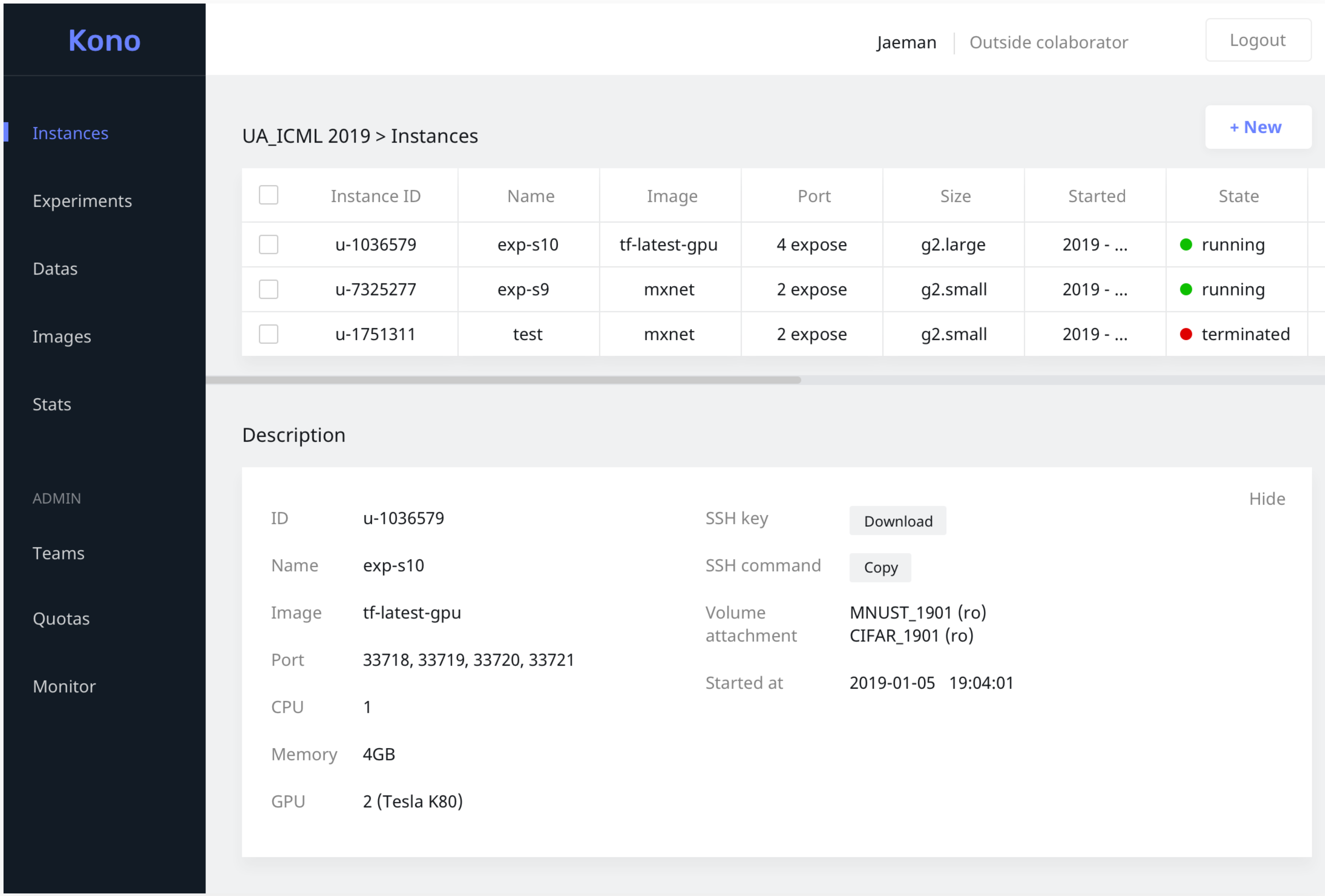
Model building & training phase

- Building training farm from zero (step by step)
- Polyaxon
- Terraform

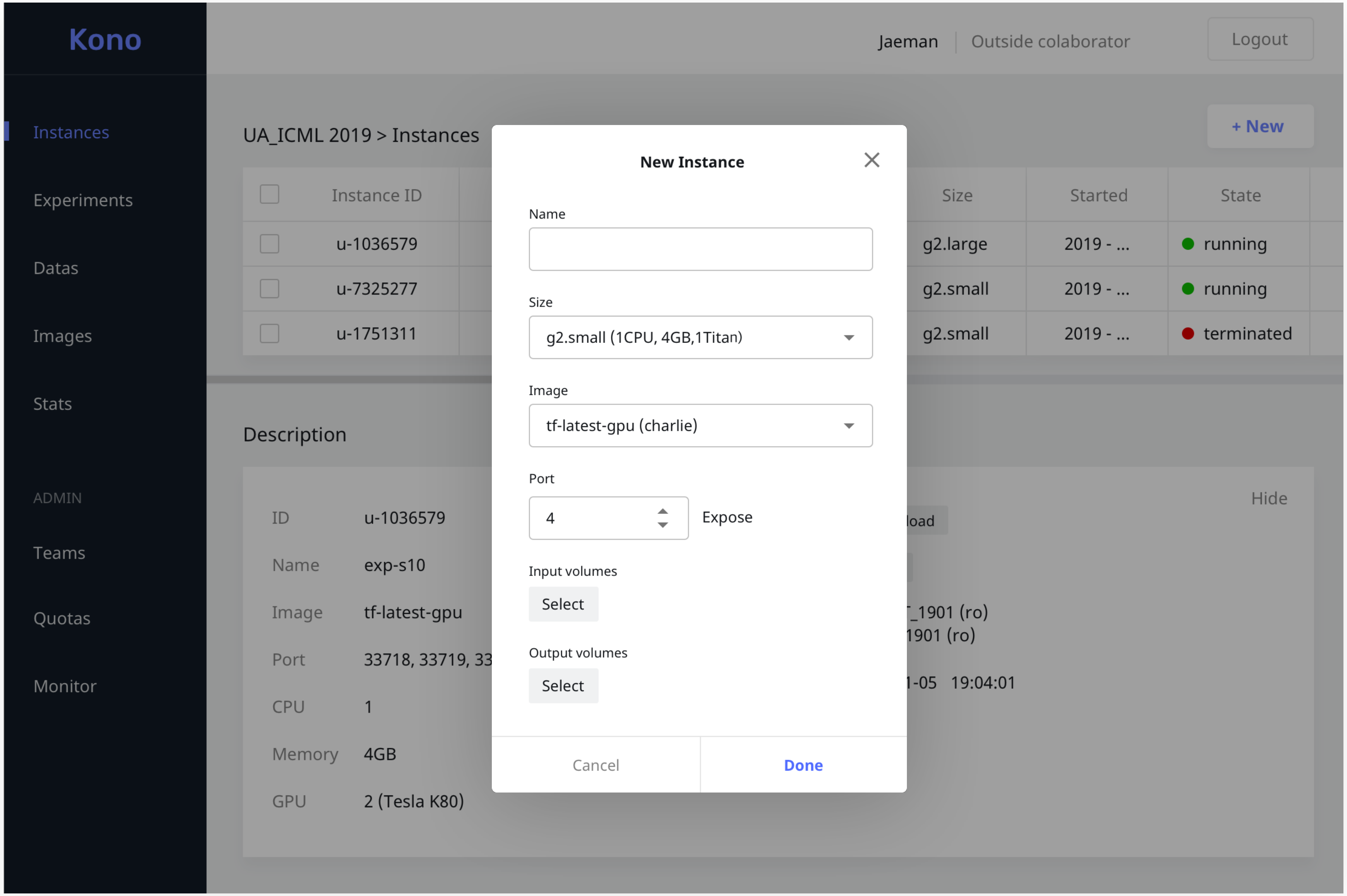
RECAP: Our requirements

- We need to know GPU resource status without accessing our physical servers one by one.
- We want to easily use idle GPU with proper training datasets
- We have to control permissions of our resources and datasets
- We only want to focus on our research: building models, doing the experiments, ... not infrastructures!
  - `./run-notebook tf-v12-gpu --gpu=4 --data=images_v1`
  - `./train tf-v12-gpu model.py --gpu=4 --data=images_v1`
  - `./ssh tf-v12-gpu --gpu=4 --data=images_v1 --exposes-port=4`

Blueprint


Kono
Jaeman
Outside colaborator
Logout
Instances
Experiments
Datas
Images
Stats
ADMIN
Teams
Quotas
Monitor
UA_ICML 2019 > Instances
+ New
Instance ID
Name
Image
Port
Size
Started
State
u-1036579 exp-s10 tf-latest-gpu 4 expose g2.large 2019 - ... running
u-7325277 exp-s9 mxnet 2 expose g2.small 2019 - ... running
u-1751311 test mxnet 2 expose g2.small 2019 - ... terminated
Description
Hide
ID u-1036579
Name exp-s10
Image tf-latest-gpu
Port 33718, 33719, 33720, 33721
CPU 1
Memory 4GB
GPU 2 (Tesla K80)
SSH key Download
SSH command Copy
Volume attachment MNUST_1901 (ro) CIFAR_1901 (ro)
Started at 2019-01-05 19:04:01

Blueprint



Kono
Jaeman
Outside colaborator
Logout
Instances
Experiments
Datas
Images
Stats
ADMIN
Teams
Quotas
Monitor
UA_ICML 2019 > Instances
+ New
Instance ID
Size
Started
State
u-1036579
g2.large
2019 - ...
running
u-7325277
g2.small
2019 - ...
running
u-1751311
g2.small
2019 - ...
terminated
Description
Hide
ID
u-1036579
Name
exp-s10
Image
tf-latest-gpu
Port
33718, 33719, 33
CPU
1
Memory
4GB
GPU
2 (Tesla K80)
New Instance
Name
Size
g2.small (1CPU, 4GB,1Titan)
Image
tf-latest-gpu (charlie)
Port
4
Expose
Input volumes
Select
Output volumes
Select
Cancel
Done

Instructions

- Step 1. Install Kubernetes master on AWS
- Step 2. Install Kubernetes as nodes in physical servers
- Step 3. Run hello world training containers
- Step 4. RBAC Authorization & resource quota
- Step 5. Expand GPU servers on demand with AWS
- Step 6. Attach training data
- Step 7. Web dashboard or cli tools to run training container
- Step 8. With other tools (Polyaxon)

Step 1. Install Kubernetes master on AWS

- There are several ways to install kubernetes
- Use kubeadm in this session.
  - Other options: conjure-up, kops
- Network option: flannel (https://github.com/coreos/flannel)
- Server configuration that I've used in k8s master:
  - AWS t3.large: 2 vCPUs, 8GB Memory
  - Ubuntu 18.04, docker version 18.09

Ref: https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/

Step 1. Install Kubernetes master on AWS

```
# Install kubeadm
# https://kubernetes.io/docs/setup/independent/install-kubeadm/

$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg \
  | apt-key add -


$ cat <<EOF > /etc/apt/sources.list.d/kubernetes.list
  deb https://apt.kubernetes.io/ kubernetes-xenial main
  EOF


$ apt-get install -y kubelet kubeadm kubectl
```

Ref: https://kubernetes.io/docs/setup/independent/install-kubeadm/

Step 1. Install Kubernetes master on AWS

```
# Initialize with Flannel (https://github.com/coreos/flannel)

$ kubeadm init --pod-network-cidr=10.244.0.0/16
```

Ref: https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/

Step 1. Install Kubernetes master on AWS

```
# Initialize with Flannel (https://github.com/coreos/flannel)

$ kubeadm init --pod-network-cidr=10.244.0.0/16

Your kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

You can now join any number of machines by running the following on each node
as root:

kubeadm join 172.31.30.194:6443 --token *** --discovery-token-ca-cert-hash ***
```

Ref: https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/

Step 1. Install Kubernetes master on AWS

```
# Initialize with Flannel (https://github.com/coreos/flannel)

$ kubectl -n kube-system apply -f https://raw.githubusercontent.com/
  coreos/flannel/62e44c867a2846fefb68bd5f178daf4da3095ccb/
  Documentation/kube-flannel.yml
```

Ref: https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/

Step 1. Install Kubernetes master on AWS

```
# Install NVIDIA k8s-device-plugin
# https://github.com/NVIDIA/k8s-device-plugin

$ kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-
  device-plugin/v1.11/nvidia-device-plugin.yml
```

Ref: https://github.com/NVIDIA/k8s-device-plugin

Step 2. Install kubernetes as nodes in physical servers

- In this step,
  - install nvidia-docker
  - join to kubernetes master
    - use kubeadm join command
  - install NVIDIA's k8s-device-plugin
  - create kubernetes dashboard to check resources
- Server configuration that I've used in k8s node:
  - 32 CPU core, 128GB Memory
  - 4 GPU (Titan Xp), Driver version: 396.44
  - Ubuntu 16.04, docker version 18.09

Step 2. Install kubernetes as nodes in physical servers

```
# Install nvidia-docker (https://github.com/NVIDIA/nvidia-docker)

$ curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | apt-key
  add -
$ curl -s -L https://nvidia.github.io/nvidia-docker/ubuntu18.04/nvidia-
  docker.list | tee /etc/apt/sources.list.d/nvidia-docker.list

$ apt-get update
$ apt-get install -y nvidia-docker2
```

Ref: https://github.com/NVIDIA/nvidia-docker

Step 2. Install kubernetes as nodes in physical servers

```
# change docker default runtime to nvidia-docker

$ vi /etc/docker/daemon.json
{
    "default-runtime": "nvidia",
    "runtimes": {
        "nvidia": {
            "path": “nvidia-container-runtime",
            "runtimeArgs": []
        }
    }
}

$ systemctl restart docker
```

Ref: https://github.com/NVIDIA/nvidia-docker

AI TRICS

Step 2. Install kubernetes as nodes in physical servers

```
# test nvidia-docker is successfully installed

$ docker run --rm -it nvidia/cuda nvidia-smi
```

Ref: https://github.com/NVIDIA/nvidia-docker

AITRICS

Step 2. Install kubernetes as nodes in physical servers

```
# test nvidia-docker is successfully installed

$ docker run --rm -it nvidia/cuda nvidia-smi

+-----------------------------------------------------------------------------+
| NVIDIA-SMI 396.44    Driver Version: 396.44    CUDA Version: 10.0         |
|-----------------------------------------------------------------------------|
| GPU Name        Persistence-M| Bus-Id    Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf   Pwr:Usage/Cap|     Memory-Usage | GPU-Util  Compute M. |
|=============================+==================+======================|
| 0  Titan Xp            On    | 00 :00:1E.0 Off |                    0 |
+-----------------------------+------------------+-------- -------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Ref: https://github.com/NVIDIA/nvidia-docker

Step 2. Install kubernetes as nodes in physical servers

```
# join to kubernetes master with kubeadm

$ kubeadm join 172.31.30.194:6443 --token *** --discovery-token-ca-
  cert-hash ***
```

Step 2. Install kubernetes as nodes in physical servers

```
# join to kubernetes master with kubeadm

$ kubeadm join 172.31.30.194:6443 --token *** --discovery-token-ca-
  cert-hash ***

...

This node has joined the cluster.
* Certificate signing request was sent to apiserver and a response was
received
* The Kubelet was informed of the new secure connection details

Run 'kubectl get nodes' on the master to see this node join the
cluster.
```

Step 2. Install kubernetes as nodes in physical servers

```
# check the node join the cluster
# run this on the master

$ kubectl get nodes
```

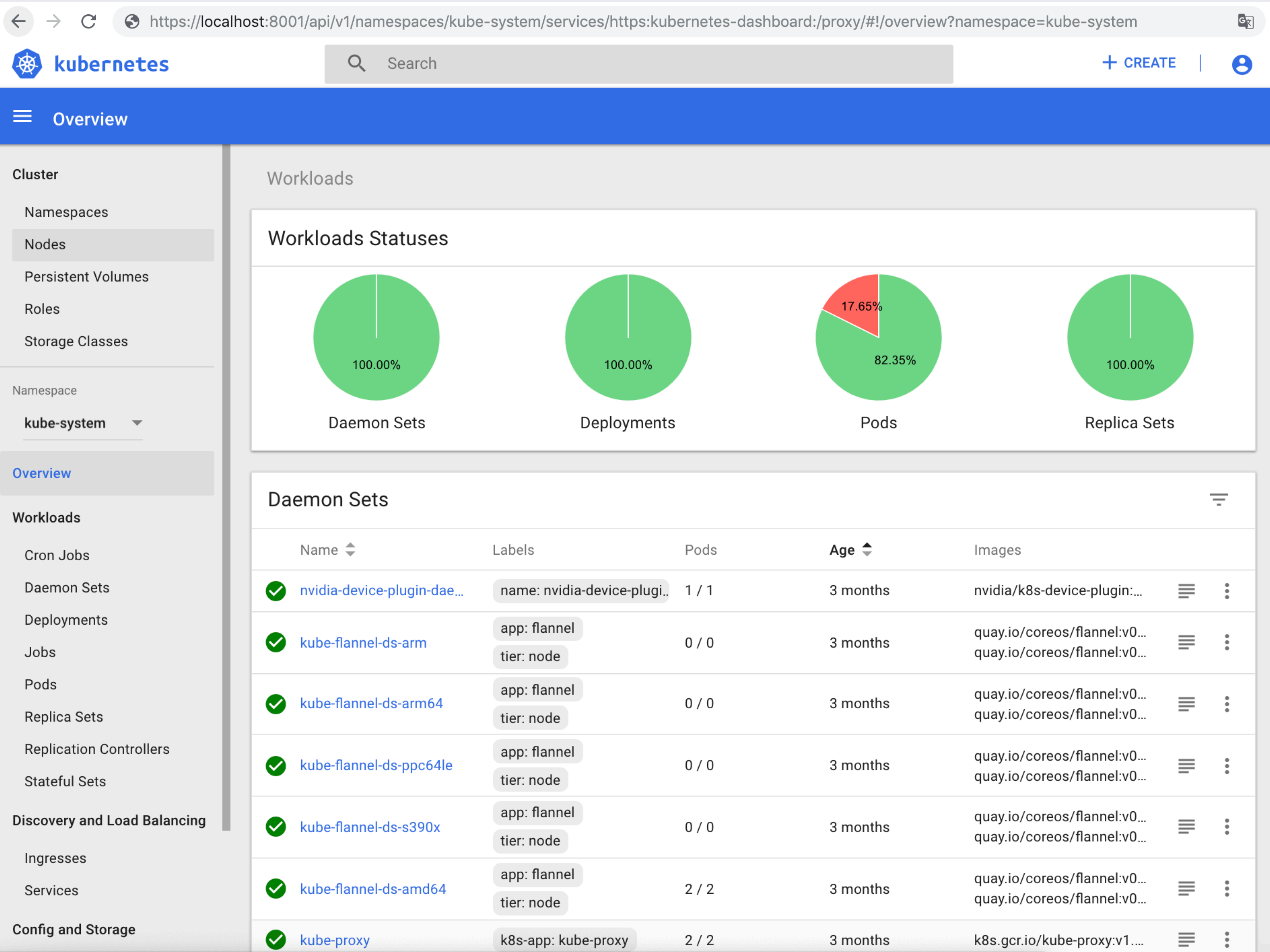
Step 2. Install kubernetes as nodes in physical servers

```
# check if the node (named as 'stark') join the cluster
# run this command on the master

$ kubectl get nodes

NAME              STATUS   ROLES    AGE   VERSION
ip-172-31-99-9    Ready    master   99d   v1.12.2
stark             Ready    <none>   99d   v1.12.2
```

Step 2. Install kubernetes as nodes in physical servers

```
# create kubernetes dashboard

$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/
  dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml

$ kubectl proxy
```

Ref: https://github.com/kubernetes/dashboard


https://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/overview?namespace=kube-system
kubernetes
Search
+ CREATE
Overview
Cluster
Namespaces
Nodes
Persistent Volumes
Roles
Storage Classes
Namespace
kube-system
Overview
Workloads
Cron Jobs
Daemon Sets
Deployments
Jobs
Pods
Replica Sets
Replication Controllers
Stateful Sets
Discovery and Load Balancing
Ingresses
Services
Config and Storage
Workloads
Workloads Statuses
100.00%
Daemon Sets
100.00%
Deployments
17.65%
82.35%
Pods
100.00%
Replica Sets
Daemon Sets
Name
Labels
Pods
Age
Images
nvidia-device-plugin-dae...
name: nvidia-device-plugi..
1 / 1
3 months
nvidia/k8s-device-plugin:...
kube-flannel-ds-arm
app: flannel
tier: node
0 / 0
3 months
quay.io/coreos/flannel:v0...
quay.io/coreos/flannel:v0...
kube-flannel-ds-arm64
app: flannel
tier: node
0 / 0
3 months
quay.io/coreos/flannel:v0...
quay.io/coreos/flannel:v0...
kube-flannel-ds-ppc64le
app: flannel
tier: node
0 / 0
3 months
quay.io/coreos/flannel:v0...
quay.io/coreos/flannel:v0...
kube-flannel-ds-s390x
app: flannel
tier: node
0 / 0
3 months
quay.io/coreos/flannel:v0...
quay.io/coreos/flannel:v0...
kube-flannel-ds-amd64
app: flannel
tier: node
2 / 2
3 months
quay.io/coreos/flannel:v0...
quay.io/coreos/flannel:v0...
kube-proxy
k8s-app: kube-proxy
2 / 2
3 months
k8s.gcr.io/kube-proxy:v1....

Step 3. Run hello-world container

- Write pod definition
  - Run nvidia-smi with cuda image
  - Train MNIST with tensorflow and save model in S3

Example: nvidia-smi

```yaml
# run nvidia-smi in container
# pod.yml

apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  containers:
    - name: cuda-container
      image: nvidia/cuda:9.0-devel
      resources:
        limits:
          nvidia.com/gpu: 1 # requesting 1 GPU
  command: ["nvidia-smi"]
```

Example: nvidia-smi

```
# create pod from definition

$ kubectl create -f pod.yml
```

Example: nvidia-smi

```
# create pod from definition

$ kubectl create -f pod.yml

pod/gpu-pod created
```

Example: nvidia-smi

```
# create pod from definition

$ kubectl logs gpu-pod

+-----------------------------------------------------------------------------+
| NVIDIA-SMI 396.44    Driver Version: 396.44    CUDA Version: 10.0          |
|-------------------------------------------------------------------------------|
| GPU Name        Persistence-M| Bus-Id    Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf   Pwr:Usage/Cap|     Memory-Usage | GPU-Util  Compute M. |
|=============================+==================+======================|
| 0  Titan Xp            On    | 00 :00:1E.0 Off |                    0 |
+-----------------------------+------------------+-------- -------------+


+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Example: MNIST

```
# train_mnist.py


import tensorflow as tf

def main(args):
    mnist = tf.keras.datasets.mnist

    (x_train, y_train),(x_test, y_test) = mnist.load_data()
    x_train, x_test = x_train / 255.0, x_test / 255.0

    model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])
    model.compile(optimizer='adam',
          loss='sparse_categorical_crossentropy',
          metrics=['accuracy'])

    model.fit(x_train, y_train, epochs=args.epoch)
    model.evaluate(x_test, y_test)

    saved_model_path = tf.contrib.saved_model.save_keras_model(model, args.save_dir)
```

Example: MNIST

```
# Dockerfile

FROM tensorflow/tensorflow:latest-gpu-py3

WORKDIR /train_demo/
COPY . /train_demo/

RUN pip --no-cache-dir install --upgrade awscli

ENTRYPOINT ["/train_demo/run.sh"]


# run.sh

python train_mnist.py --epoch 1
aws s3 sync saved_models/ $MODEL_S3_PATH
```

Example: MNIST

```yaml
# pod definition

apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  containers:
    - name: cuda-container
      image: aitrics/train-mnist:1.0
      resources:
        limits:
          nvidia.com/gpu: 1 # requesting 1 GPU
      env:
      - name: MODEL_S3_PATH
        value: "s3://aitrics-model-bucket/saved_model"
```

Example: MNIST

```
# create pod from definition

$ kubectl create -f pod.yml

pod/gpu-pod created
```

Example: MNIST

It works!

Summary

- Now we have,
  - Minimally working proof of concept
  - Researchers can train on kubernetes with kubectl
- We have to do,
  - RBAC (Role based access control) between researchers, engineers, and outside collaborators.
  - Training data & output volume attachment
  - Researchers don't want to know what kubernetes is. They only need
    - a instance which are accessible via SSH (with frameworks and training data)
    - or nice webview and jupyter notebook
    - or automatic hyperparameter searching...

Step 4. Role Based Access Control & Resource Quota

- Instructions:
  - Create user (team) namespace
  - Create user credentials with cluster CA key
    - default CA key location: /etc/kubernetes/pki
  - Create role and role binding with proper permissions
  - Create resource quota per namespace

- References:
  - https://docs.bitnami.com/kubernetes/how-to/configure-rbac-in-your-kubernetes-cluster/
  - https://kubernetes.io/docs/reference/access-authn-authz/rbac/

Step 4. Role Based Access Control & Resource Quota

```
# create user (team) namespace


$ kubectl create namespace team-a
```

AITRICS

Step 4. Role Based Access Control & Resource Quota

```
 # create user (team) namespace


 $ kubectl get namespaces

NAME           STATUS   AGE
default        Active   99d
team-a         Active   4s
kube-public    Active   99d
kube-system    Active   99d
```

Step 4. Role Based Access Control & Resource Quota

```
# create user credentials


$ openssl genrsa -out jaeman.key 2048

$ openssl req -new -key jaeman.key -out user.csr -subj "/CN=jaeman/
  O=aitrics"

$ openssl x509 -req -in jaeman.csr -CA CA_LOCATION/ca.crt -CAkey
  CA_LOCATION/ca.key -CAcreateserial -out jaeman.crt -days 500
```

Ref: https://kubernetes.io/docs/reference/access-authn-authz/authentication/

Step 4. Role Based Access Control & Resource Quota

```
# create Role definition

kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: team-a
  name: software-engineer-role
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["deployments", "replicasets", "pods", "configmaps"]
  verbs: ["get", "list", "watch", "create", "update", "patch",
"delete"] # You can also use ["*"]
```

Ref: https://kubernetes.io/docs/reference/access-authn-authz/authentication/

AI TRICS

Step 4. Role Based Access Control & Resource Quota

```
# create ClusterRoleBinding definition

kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: team-a
  name: jaeman-software-engineer-role-binding
subjects:
- kind: User
  name: jaeman
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: software-engineer-role
  apiGroup: rbac.authorization.k8s.io
```

Ref: https://kubernetes.io/docs/reference/access-authn-authz/authentication/

AI TRICS

Step 4. Role Based Access Control & Resource Quota

```
# create resource quota

apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    requests.nvidia.com/gpu: 1
```

Step 5. Expand GPU servers on AWS

- Store kubeadm join script in S3
- Write userdata (instance bootstrap script)
    - install kubeadm, nvidia-docker
    - join
- Add AutoScaling Group

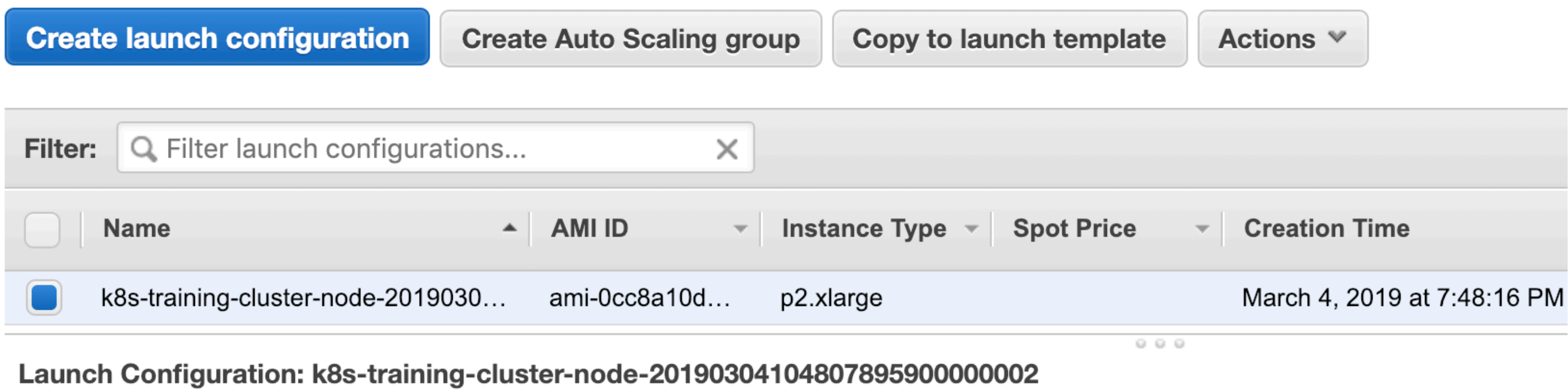
Step 5. Expand GPU servers on AWS

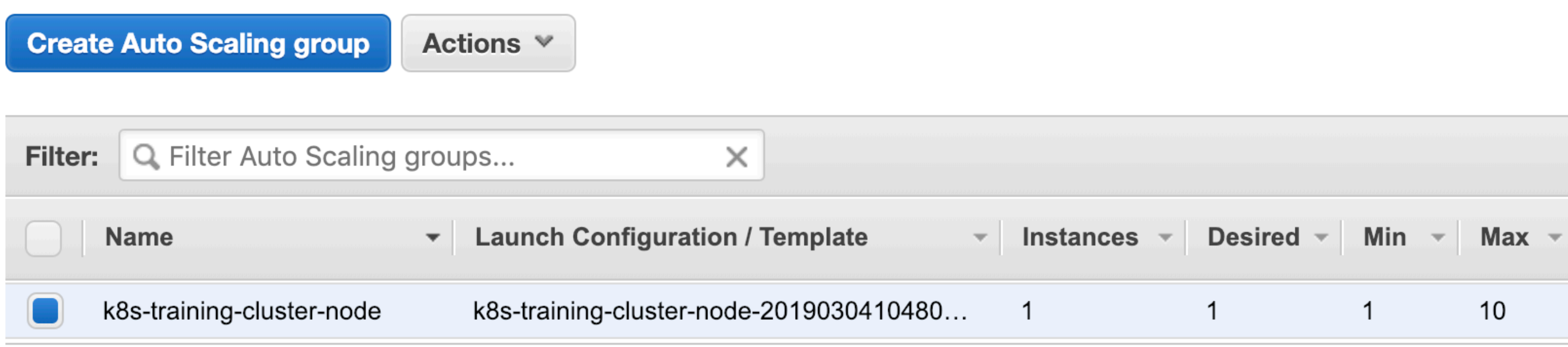
```
# save master join command in AWS S3
# s3://k8s-training-cluster/join.sh

kubeadm join 172.31.75.62:6443 --token *** --discovery-token-ca-cert-hash ***
```

Step 5. Expand GPU servers on AWS

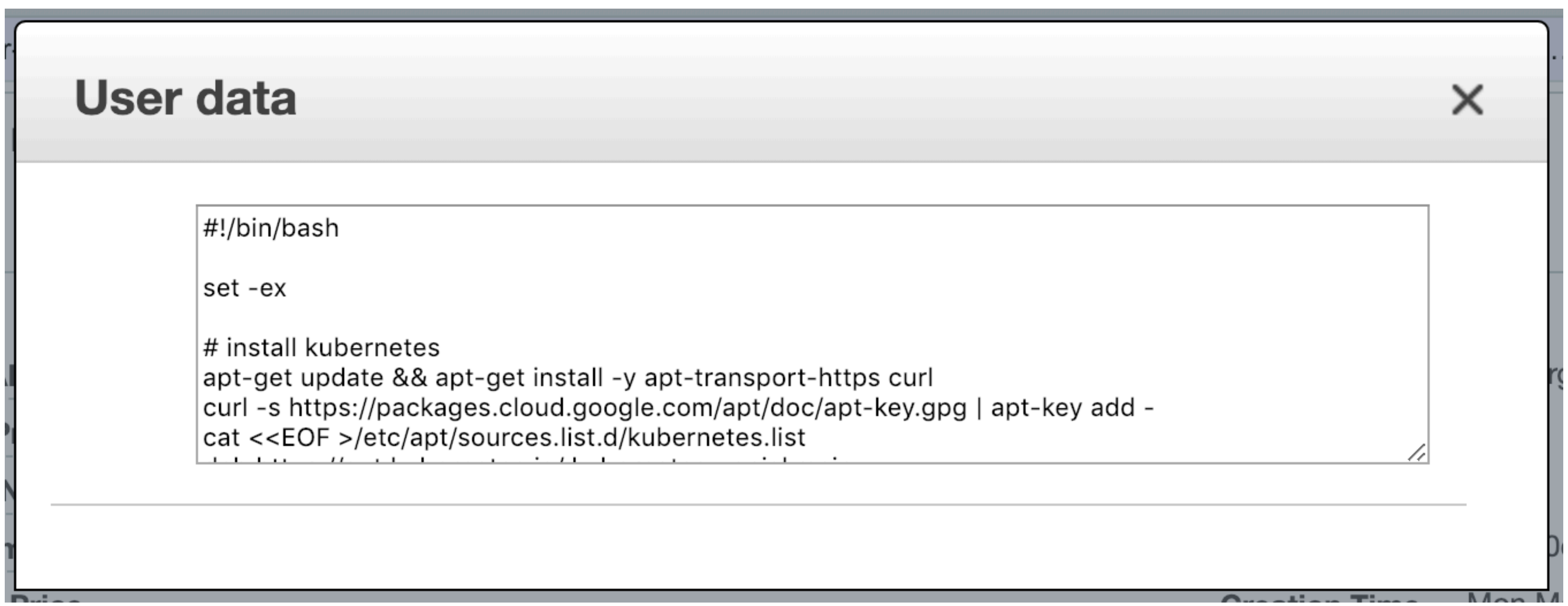
```
# userdata script file
# RECAP: install kubernetes as a node to join master (step 2)

# install kubernetes
apt-get install -y kubelet kubeadm kubectl

# install nvidia-docker
apt-get install -y nvidia-docker2

...

$(aws s3 cp s3://k8s-training-cluster/join.sh -)
```

Step 5. Expand GPU servers on AWS





Step 5. Expand GPU servers on AWS



Step 5. Expand GPU servers on AWS

```
# check bootstrapping log

$ tail -f /var/log/cloud-init-output.log
```

Step 5. Expand GPU servers on AWS

```
# check bootstrapping log

$ tail -f /var/log/cloud-init-output.log

...
++ aws s3 cp s3://k8s-training-cluster/join.sh -
+ kubeadm join 172.31.75.62:6443 --token *** --discovery-token-ca-cert-hash ***
[preflight] Running pre-flight checks
[discovery] Trying to connect to API Server "172.31.75.62:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://172.31.75.62:6443"
[discovery] Requesting info from "https://172.31.75.62:6443" again to validate TLS against the pinned public key
...
```

Step 6. Training data attachment

- Initially store training data in S3 (with encryption)
- Option 1: Download training data when pod starts
  - training data is usually big
  - same training data are often used, so it would be very inefficient
  - caching to host machine volumes --> occupied easily
  - use storage server and mount volumes that!
- Option 2: Create NFS on AWS EC2 or storage server (e.g. NAS)
  - Sync all data with S3
  - Mount as Persistent Volume with ReadOnlyMany / ReadWriteMany
- Option 3: shared storage with s3fs
  - https://icicimov.github.io/blog/virtualization/Kubernetes-shared-storage-with-S3-backend/

Step 6. Training data attachment

```
 # make nfs server on EC2 (or physical storage server)
 # https://www.digitalocean.com/community/tutorials/how-to-set-up-an-
nfs-mount-on-ubuntu-16-04

 $ apt-get update
 $ apt-get install nfs-kernel-server

 $ mkdir /var/nfs -p

 $ cat <<EOF > /etc/exports
   /var/nfs    172.31.75.62(rw,sync,no_subtree_check)
   EOF

 $ systemctl restart nfs-kernel-server
```

Step 6. Training data attachment

```yaml
# define persistent volume

apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs
spec:
  capacity:
    storage: 3Gi
  accessModes:
    - ReadWriteMany
  nfs:
    server: <server ip>
    path: "/var/nfs"
```

AI TRICS

Step 6. Training data attachment

```yaml
# define persistent volume claim

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 3Gi
```

Step 6. Training data attachment

```yaml
# mount volume in pod

apiVersion: v1
kind: Pod
metadata:
  name: pvpod
spec:
  volumes:
    - name: testpv
      persistentVolumeClaim:
        claimName: nfs-pvc
  containers:
  - name: test
    image: python:3.7.2
    volumeMounts:
    - name: testpv
      mountPath: /data/test
```

Step 7. Web dashboard or cli tools to run training container

- Make script like
  - `./kono ssh --image tensorflow/tensorflow --expose-ports 4`
  - `./kono train --image tensorflow/tensorflow --entrypoint main.py .`
- Create web dashboard

Step 7. Web dashboard or cli tools to run training container

```
# cli tool to use our cluster

$ kono login
```

Step 7. Web dashboard or cli tools to run training container

```
# cli tool to use our cluster

$ kono login

  Username: jaeman
  Password: [hidden]
```

Step 7. Web dashboard or cli tools to run training container

```
# cli tool to use our cluster

$ kono train \
      --image tensorflow/tensorflow:latest-gpu \
      --gpu 1 \
      --script train.py \
      --input-data /var/project-a-data/:/opt/project-a-data/ \
      --output-dir /opt/outputs/:./outputs/ \
      -- \
      --epoch=1 --checkpoint=/opt/outputs/ckpts/
```

Step 7. Web dashboard or cli tools to run training container

```
# cli tool to use our cluster

$ kono train \
      --image tensorflow/tensorflow:latest-gpu \
      --gpu 1 \
      --script train.py \
      --input-data /var/project-a-data/:/opt/project-a-data/ \
      --output-dir /opt/outputs/:./outputs/ \
      -- \
      --epoch=1 --checkpoint=/opt/outputs/ckpts/


  ...
  ...
  training completed!
  Sending output directory to s3... [>>>>>>>>>>>>>>>>>>>>>>>] 100%
  Pulling output directory to local... [>>>>>>>>>>>>>>>>>>>>>>>] 100%
  Check your directory ./outputs/
```

Step 7. Web dashboard or cli tools to run training container

```
# cli tool to use our cluster

$ kono ssh \
      --image tensorflow/tensorflow:latest-gpu \
      --gpu 1 \
      --expose-ports 4 \
      --input-data /var/project-a-data/:/opt/project-a-data/
```

Step 7. Web dashboard or cli tools to run training container

```
# cli tool to use our cluster

$ kono ssh \
      --image tensorflow/tensorflow:latest-gpu \
      --gpu 1 \
      --expose-ports 4 \
      --input-data /var/project-a-data/:/opt/project-a-data/

  ...
  ...
  ...

  Your container is ready!
  ssh ubuntu@k8s.aitrics.com -p 31546
```

Step 7. Web dashboard or cli tools to run training container

```
# cli tool to use our cluster

$ kono terminate-all --force
```

AITRICS

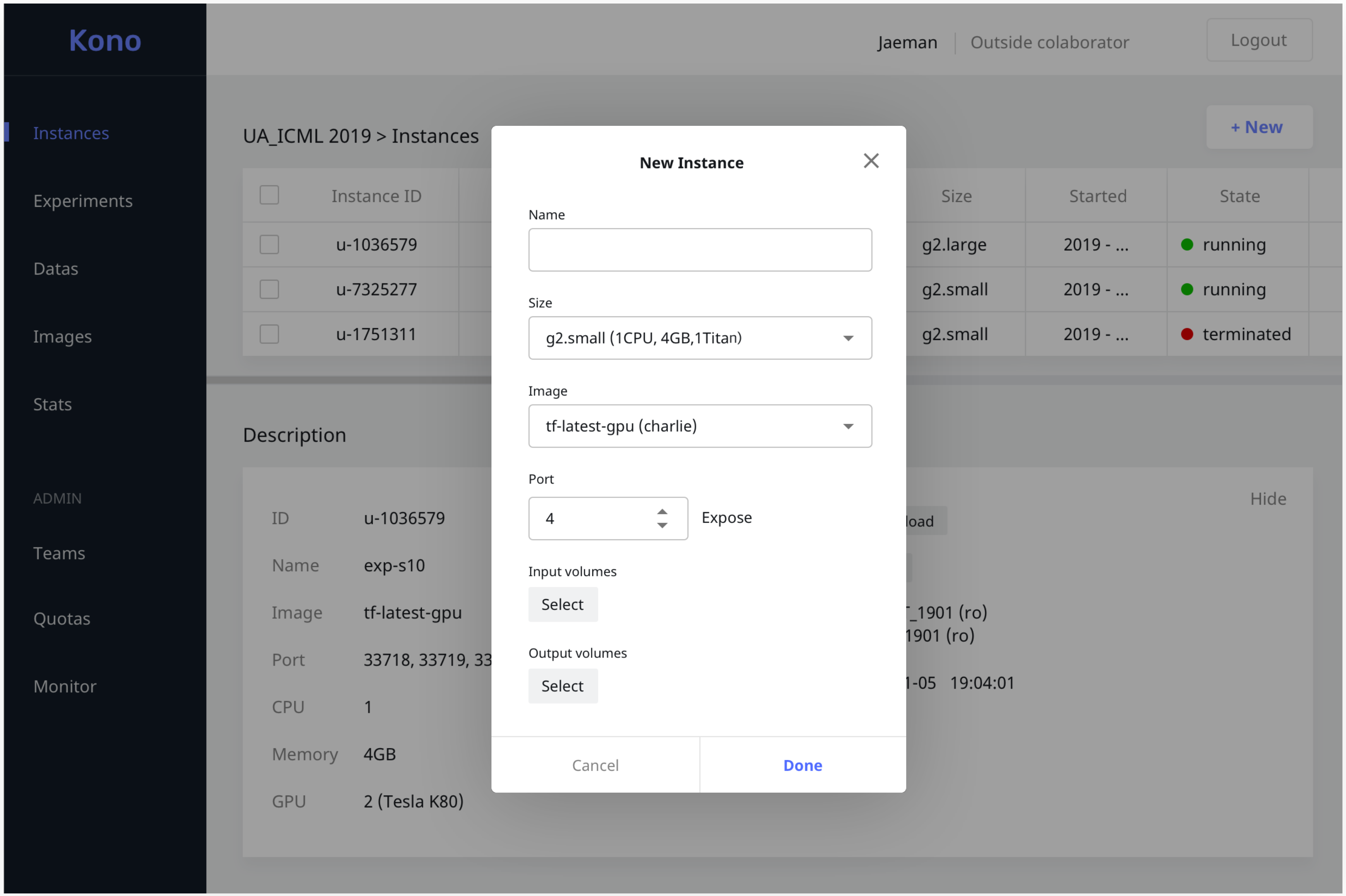
Step 7. Web dashboard or cli tools to run training container

```
# cli tool to use our cluster

$ kono terminate-all --force

  terminate all your containers? [Y/n]: Y
```

Step 7. Web dashboard or cli tools to run training container

```
# cli tool to use our cluster

$ kono terminate-all --force

  terminate all your containers? [Y/n]: Y

  ...
  ...
  ...
  Success!
```

Step 7. Web dashboard or cli tools to run training container

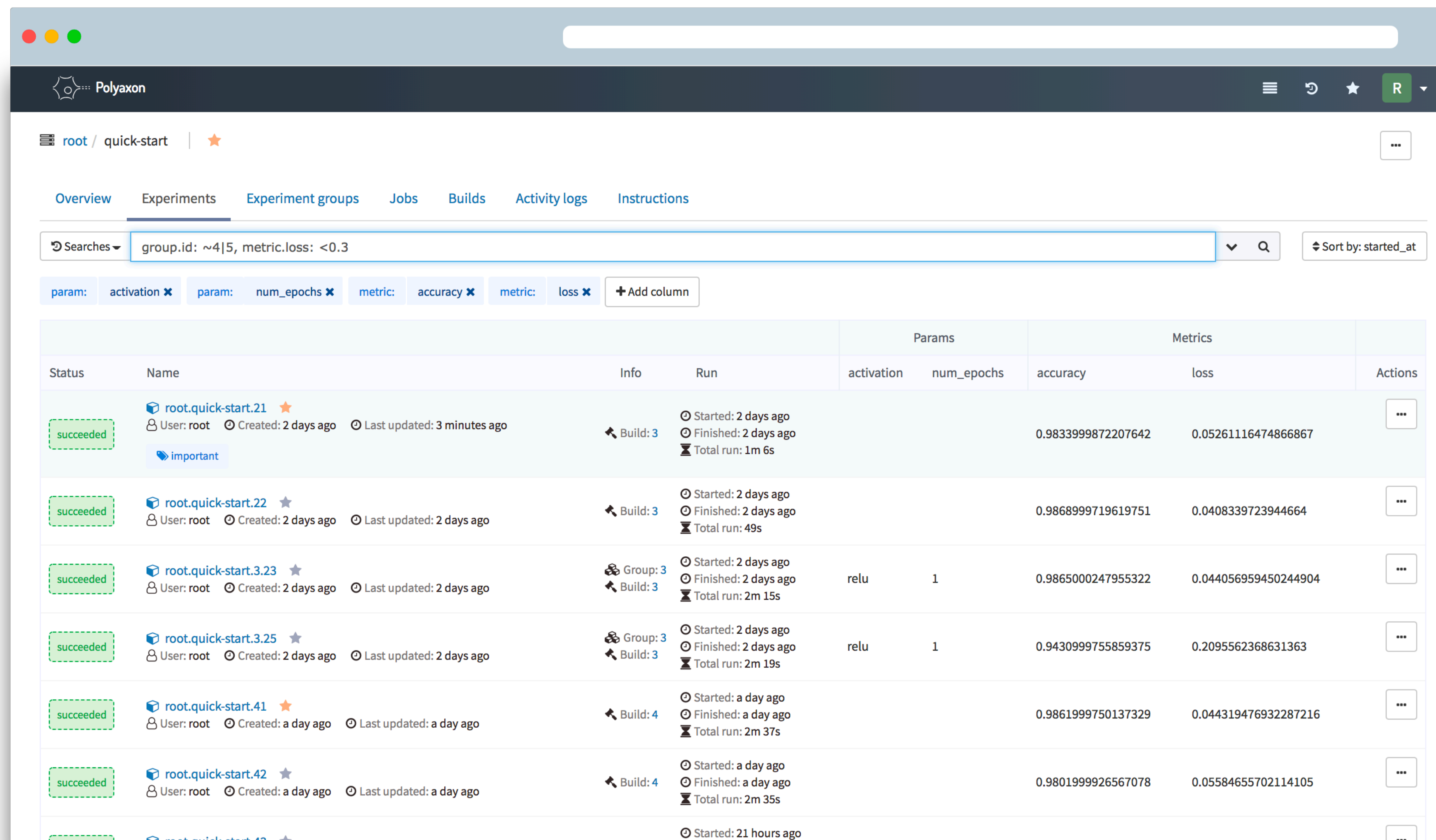


Step 7. Web dashboard or cli tools to run training container

- We are still working on it
  - Check our improvements or contribute to us
  - https://github.com/AITRICS/kono

Step 8. Use other tools (polyaxon)

- A platform for reproducing and managing the whole life cycle of machine learning and deep learning applications.
- https://polyaxon.com/
- Most feasible tools to our training cluster
- Can be installed on kubernetes easily



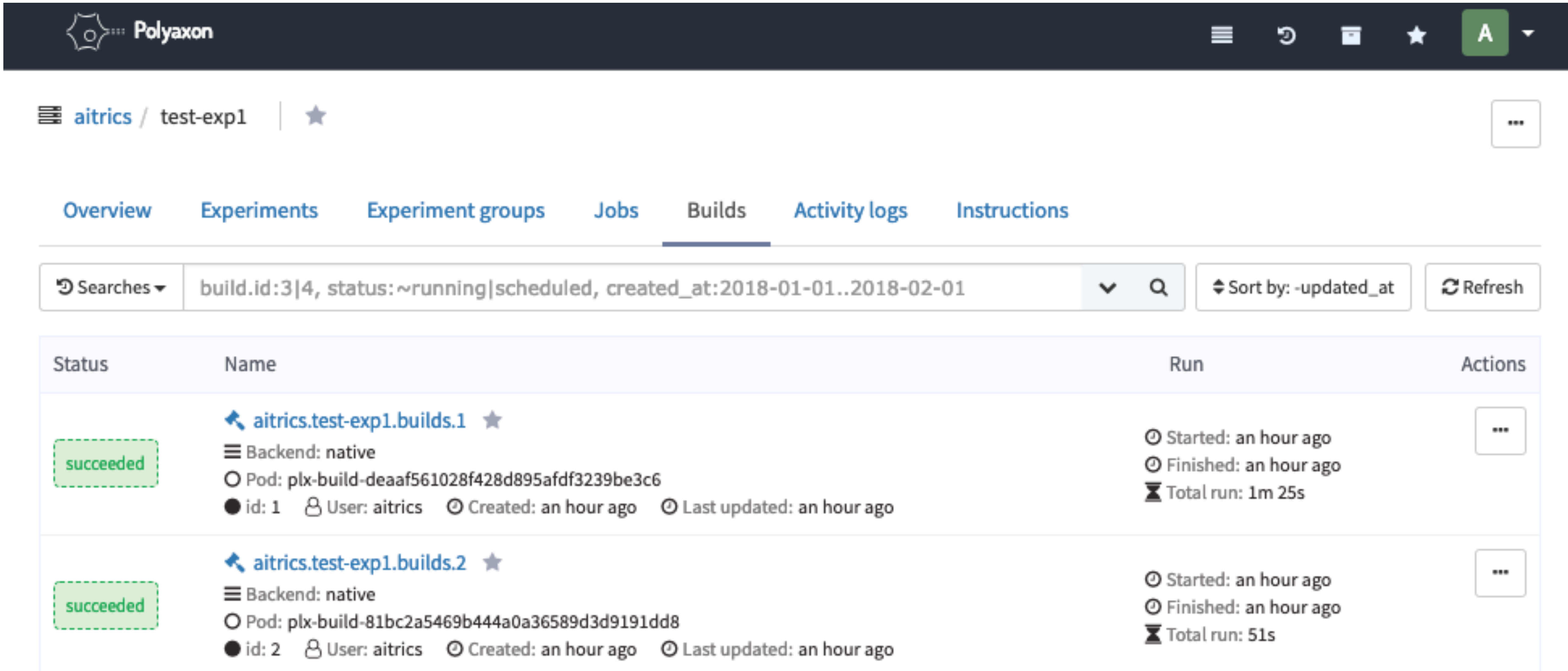
Ref: https://www.polyaxon.com/

Polyaxon usage

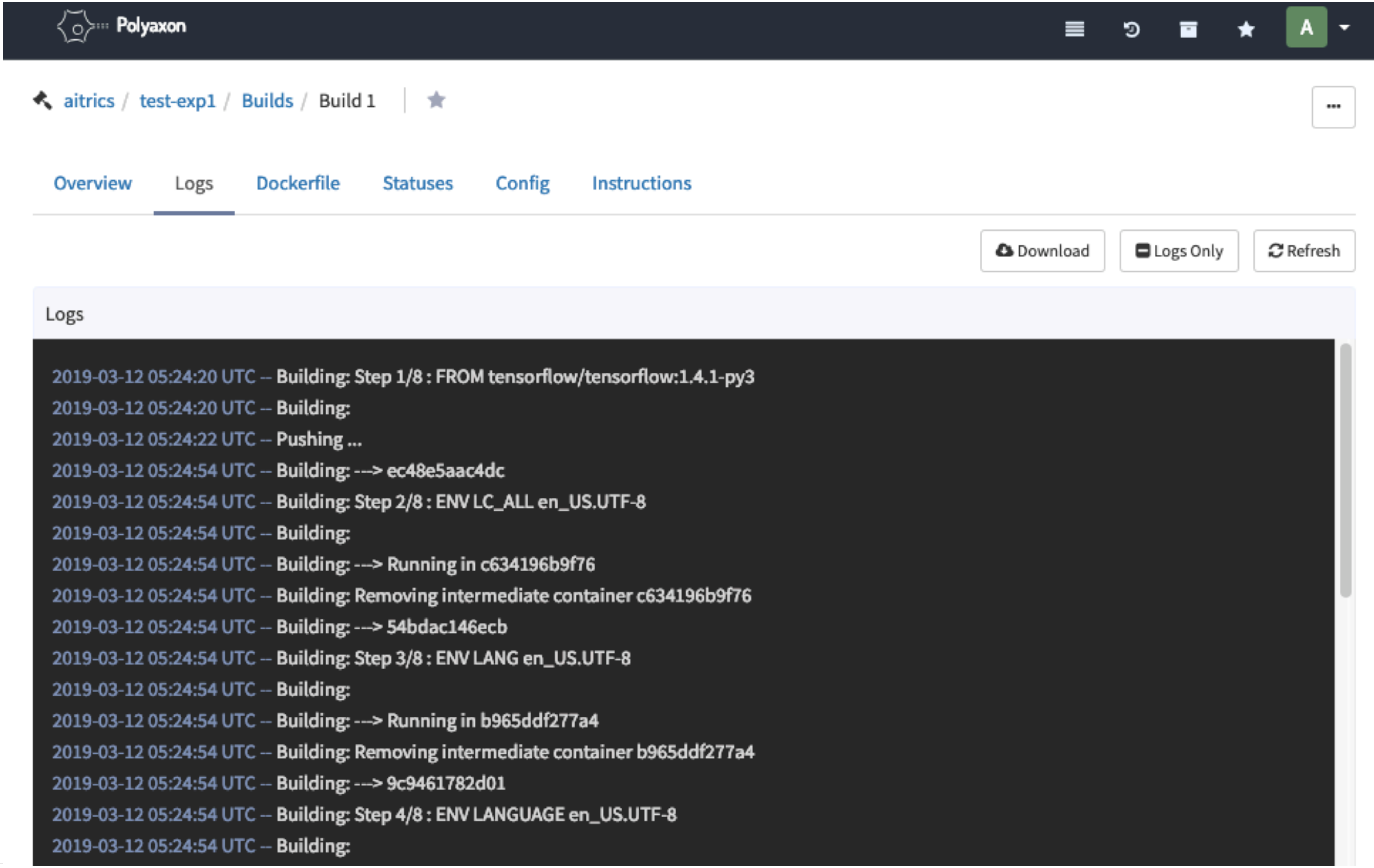
```
# Polyaxon usage

# Create a project
$ polyaxon project create --name=quick-start --description='Polyaxon
  quick start.’

# Initialize
$ polyaxon init quick-start

# Upload code and start experiments
$ polyaxon run -u
```

Ref: https://github.com/polyaxon/polyaxon

Polyaxon usage



Polyaxon usage

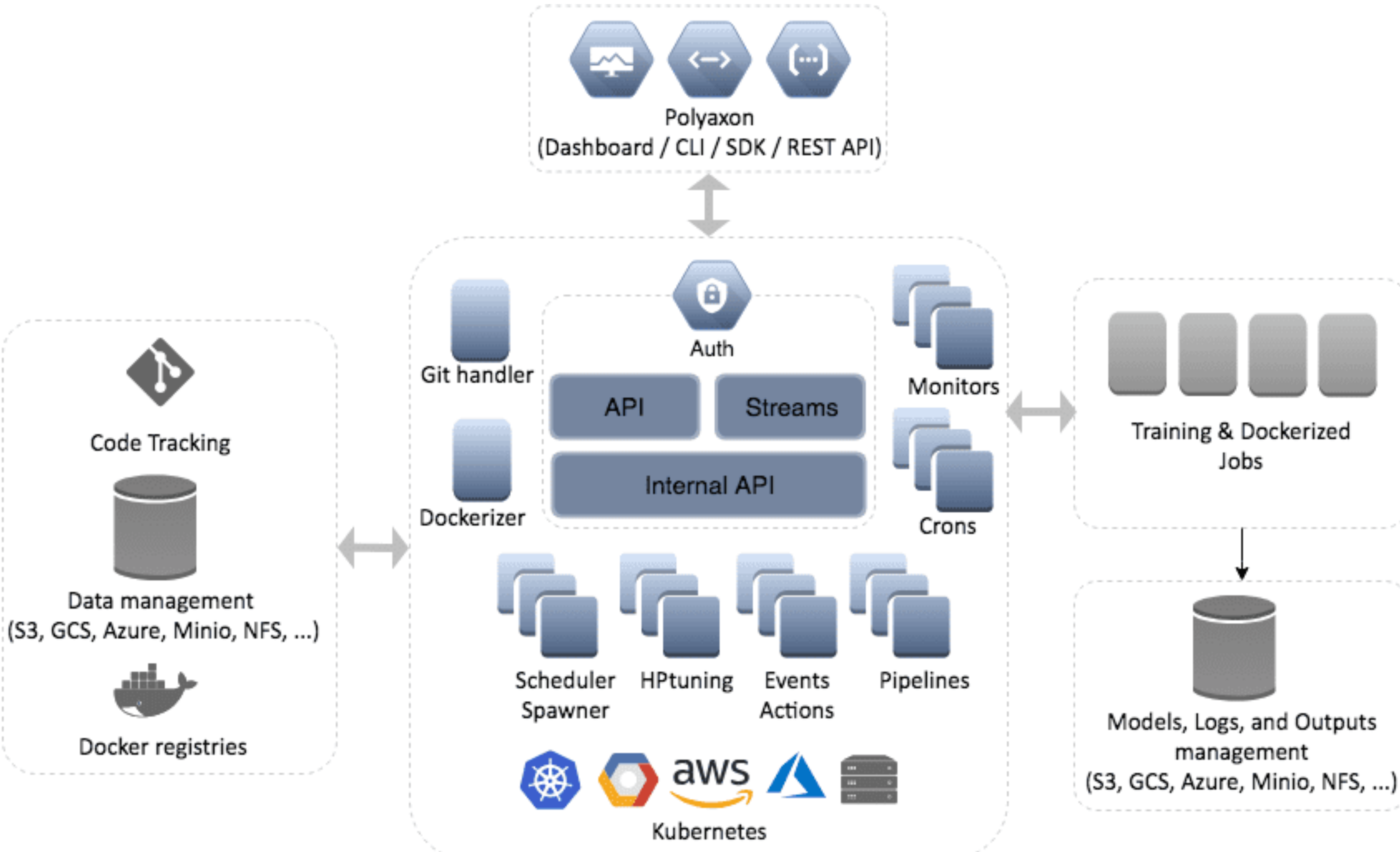


Polyaxon

- Polyaxon is a platform for managing the whole lifecycle of large scale deep learning and machine learning applications, and it supports all the major deep learning frameworks such as Tensorflow, MXNet, Caffe, Torch, etc.
- Features
  - Powerful workspace
  - Reproducible results
  - Developer-friendly API
  - Built-in Optimization engine
  - Plugins & integrations
  - Roles & permissions

Ref: https://docs.polyaxon.com/concepts/features/

Polyaxon architecture



Ref: https://docs.polyaxon.com/concepts/architecture/

How to run my experiment on polyaxon?

- 1. Create project on polyaxon
  - `polyaxon project create --name=quick-start`
- 2. Initialize the project
  - `polyaxon init quick-start`
- 3. Create polyaxonfile.yml
  - See next slide
- 4. Upload your code and start an experiment with it

Polyaxon usage

```yaml
# polyaxonfile.yml

version: 1

kind: experiment

build:
  image: tensorflow/tensorflow:1.4.1-py3
  build_steps:
    - pip3 install polyaxon-client

run:
  cmd: python model.py
```

Ref: https://docs.polyaxon.com/concepts/quick-start-internal-repo/

AITRICS

Polyaxon usage

```python
# model.py
# https://github.com/polyaxon/polyaxon-quick-start/blob/master/model.py

from polyaxon_client.tracking import Experiment, get_data_paths, get_outputs_path

data_paths = list(get_data_paths().values())[0]
mnist = input_data.read_data_sets(data_paths, one_hot=False)

experiment = Experiment()

...

estimator = tf.estimator.Estimator(
  get_model_fn(learning_rate=learning_rate, dropout=dropout, activation=activation),
  model_dir=get_outputs_path())

estimator.train(input_fn, steps=num_steps)

...

experiment.log_metrics(loss=metrics['loss'],
                       accuracy=metrics['accuracy'],
                       precision=metrics['precision'])
```

Ref: https://github.com/polyaxon/polyaxon-quick-start/blob/master/model.py

Polyaxon usage

```
# Integrations in polyaxon

# Notebook
$ polyaxon notebook start -f polyaxon_notebook.yml

# Tensorboard
$ polyaxon tensorboard -xp 23 start
```

Ref: https://github.com/polyaxon/polyaxon

Experiment Groups - Hyperparameter Optimization

- How to?
  - Make single file train.py that accepts 2 parameters
    - learning rate - `lr`
    - batch size - `batch_size`
  - Update the polyaxonfile.yml with matrix
  - Make experiment group

- Experiment group search algorithm
  - grid search / random search / Hyperband / Bayesian Optimization
  - https://docs.polyaxon.com/references/polyaxon-optimization-engine/

Ref: https://docs.polyaxon.com/concepts/experiment-groups-hyperparameters-optimization/

Experiment Groups - Hyperparameter Optimization

```yaml
# polyaxonfile.yml

version: 1
kind: group
declarations:
  batch_size: 128
hptuning:
  matrix:
    lr:
      logspace: 0.01:0.1:5
build:
  image: tensorflow/tensorflow:1.4.1-py3
  build_steps:
    - pip install scikit-learn
run:
  cmd: python3 train.py --batch-size={{ batch_size }} --lr={{ lr }}
```

Ref: https://docs.polyaxon.com/concepts/experiment-groups-hyperparameters-optimization/

Experiment Groups - Hyperparameter Optimization

```yaml
# polyaxonfile_override.yml

version: 1
hptuning:
  concurrency: 2
  random_search:
    n_experiments: 4
  early_stopping:
    - metric: accuracy
      value: 0.9
      optimization: maximize
    - metric: loss
      value: 0.05
      optimization: minimize
```

Ref: https://docs.polyaxon.com/concepts/experiment-groups-hyperparameters-optimization/

AITRICS

How to install polyaxon?

- Instructions
  - Install helm - kubernetes application manager
  - Create polyaxon namespace
  - Write your own config for polyaxon
  - Run polyaxon with helm

How to install polyaxon?

```
# install helm (kubernetes package manager)

$ snap install helm --classic

$ helm init
```

Ref: https://github.com/polyaxon/polyaxon

How to install polyaxon?

```
# install polyaxon with helm

$ kubectl create namespace polyaxon

$ helm repo add polyaxon https://charts.polyaxon.com

$ helm repo update
```

Ref: https://github.com/polyaxon/polyaxon

How to install polyaxon?

```yaml
# config.yaml

rbac:
  enabled: true
ingress:
  enabled: true
serviceType: LoadBalancer
persistent:
  data:
    training-data-a-s3:
      store: s3
      bucket: s3://aitrics-training-data
    data-pvc1:
      mountPath: "/data-pvc/1"
      existingClaim: "data-pvc-1"
  outputs:
    devtest-s3:
      store: s3
      bucket: s3://aitrics-dev-test
integrations:
  slack:
    - url: https://hooks.slack.com/services/***/***
      channel: research-feed
```

Ref: https://github.com/polyaxon/polyaxon

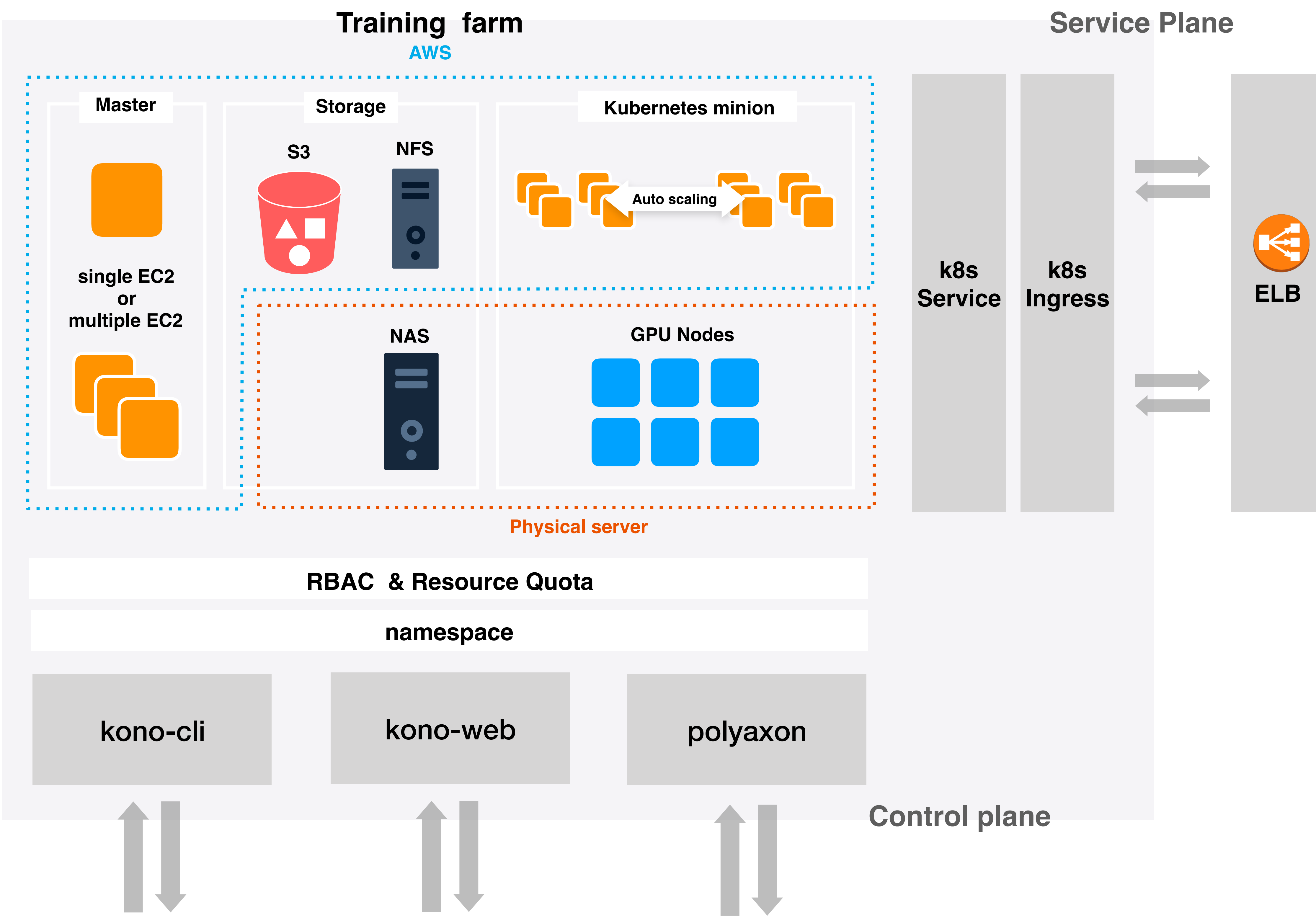
How to install polyaxon?

```
# install polyaxon with helm

$ helm install polyaxon/polyaxon \
      --name=polyaxon \
      --namespace=polyaxon \
      -f config.yml
```

How to install polyaxon?

```
# install polyaxon with helm

$ helm install polyaxon/polyaxon \
    --name=polyaxon \
    --namespace=polyaxon \
    -f config.yml


1. Get the application URL by running these commands:
   export POLYAXON_IP=$(kubectl get svc --namespace polyaxon polyaxon-polyaxon-
ingress -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
   export POLYAXON_HTTP_PORT=80
   export POLYAXON_WS_PORT=80

   echo http://$POLYAXON_IP:$POLYAXON_HTTP_PORT

2. Setup your cli by running theses commands:
  polyaxon config set --host=$POLYAXON_IP --http_port=$POLYAXON_HTTP_PORT  –
ws_port=$POLYAXON_WS_PORT
```

Summary



AI TRICS

RECAP: Our requirements

- Need to know GPU resource status without accessing our physical servers one by one.
  - **Use web dashboard or other monitoring tools like Prometheus + cAdvisor**
- Want to easily use idle GPU with proper training datasets
  - **Use kubernetes objects to get resources and to mount volumes**
- Have to control permissions of our resources and datasets
  - **RBAC / Resource quota in kubernetes**
- Want to focus on our research: building models, doing the experiments, ... not infrastructures!
  - **Use kono / polyaxon**

Too many steps to build my own cluster!

- Make it as reusable component
- Use Terraform

Terraform

- Infrastructure as a code

Terraform

- Infrastructure as a code

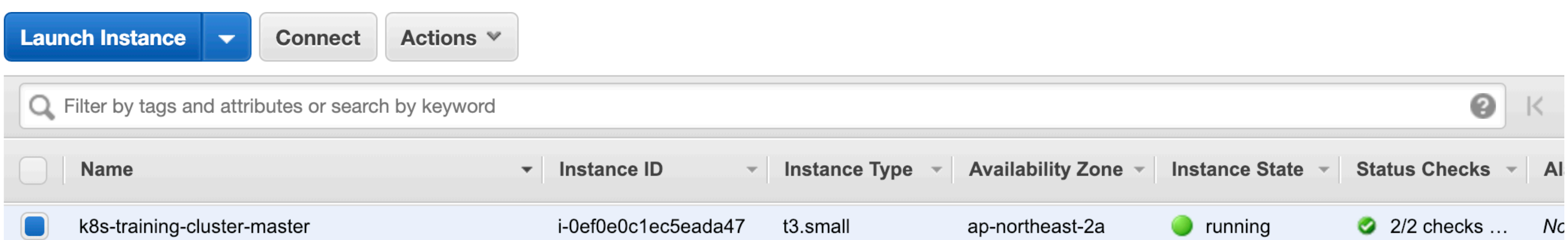
```
resource "aws_instance" "master" {
  ami                    = "ami-593801f1"
  instance_type          = "t3.small"
  key_name               = "aitrics-secret-master-key"
  iam_instance_profile = "kubernetes-master-iam-role"
  user_data                = "${data.template_file.master.rendered}"

  root_block_device = {
    volume_size = "15"
  }
}
```

```
$ terraform apply
```

Terraform

- Infrastructure as a code

```
resource "aws_instance" "master" {
  ami                  = "ami-593801f1"
  instance_type        = "t3.small"
  key_name             = "aitrics-secret-master-key"
  iam_instance_profile = "kubernetes-master-iam-role"
  user_data              = "${data.template_file.master.rendered}"

  root_block_device = {
    volume_size = "15"
  }
}
```



Terraform

- We publish our infrastructure as a code
  - https://github.com/AITRICS/kono
  - Configure your settings and just type `terraform apply` to get your own training cluster!

Model deployment & production phase

- Building inference farm from zero (step by step)
- Several ways to make microservices
- Kubeflow

RECAP: Our requirements

- It's hard to control because it is in the middle of machine learning engineering and software engineering
- We want to create simple micro-services that don't need much management
- There are many models with different purposes;
  - some models need real-time inference
  - some models do not require real-time, but they need inference in the certain time range
- We have to consider high availability configuration
- Models must be fitted and re-trained easily
- We have to manage several versions of models

Instructions

- Step 1. Build another kubernetes cluster for production
- Step 2. Make simple web-based micro services for trained models
  - 2-1. HTTP API Server Example
  - 2-2. Asynchronous inference farm example
- Step 3. Deploy
  - 3-1. on the kubernetes with ingress
  - 3-2. standalone server with docker and auto scaling group
- Step 4. Using TensorRT Inference Server
- Step 5. Terraform
- Case Study. Kubeflow

Step 1. Build production kubernetes cluster

- Launch again like training cluster!

Step 2. Make simple web-based microservices for trained models

- 2-1. For real time inference (synchronous)
  - Use simple web framework to build HTTP-based microservice!
  - We use bottle (or flask)
- 2-2. For asynchronous (inference farm)
  - with kubernetes job - has overheads to be executed
  - with celery - which I prefer

Example. Using bottle for HTTP based microservices

```
from bottle import run, get, post, request, response
from bottle import app as bottle_app
from aws import aws_client

@post('/v1/<location>/<prediction_type>/')
def inference(location, prediction_type):
  model = select_model(location, prediction_type)
  input_array = deserialize(request.json)
  output_array = inference(input_array)
  return serialize(output_array)

if __name__ == '__main__':
  args = parse_args()
  aws_client.download_model(args.model_path, args.model_version)
  app = bottle_app()
  run(app=app, host=args.host, port=args.port)
```

Example. Using kubernetes job for inference

```yaml
# job.yml

apiVersion: batch/v1
kind: Job
metadata:
  name: inference-job
spec:
  template:
    spec:
      containers:
      - name: inference
        image: inference
        command: ["python", "main.py", "s3://ps-images/images.png"]
      restartPolicy: Never
  backoffLimit: 4
```

Ref: https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/

Celery

- Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well.

- Celery is used in production systems to process millions of tasks a day.

```
from celery import Celery

app = Celery('hello', broker='amqp://guest@localhost//')

@app.task
def hello():
    return 'hello world'
```

Ref: http://www.celeryproject.org/

Example. Using celery for asynchronous inference farm

```python
from celery import task
from aws import aws_client
from db import IdentifyResult
from aitrics.models import FasterRCNN

model = FasterRCNN(model_path=settings.MODEL_PATH)

@task
def task_identify_image_color_shape(id, s3_path):
  image = aws_client.download_image(s3_path)
  color, shape = model.inference(image)
  IdentifyResult.objects.create(id, s3_path, color, shape)
```

Step 3. Deploy

- on the kubernetes cluster
  - service & ingress to expose
  - use workload controller like deployments, replica set, replication controller, don't use pod itself to get high availability.
- on the AWS instance directly
  - simple docker run example
  - use auto scaling group and load balancers with userdata

Step 3-1. Deploy on kubernetes cluster (ingress)

```yaml
kind: Ingress
metadata:
  name: inference-ingress
spec:
  rules:
  - host: inference.aitrics.com
  - http:
      paths:
      - backend:
          serviceName: MyInferenceService
          servicePort: 80
```

Ref: https://kubernetes.io/docs/concepts/services-networking/ingress/

Step 3-1. Deploy on kubernetes cluster (deployment)

```yaml
kind: Deployment
metadata:
  name: inference-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: inference
  template:
    metadata:
      labels:
        app: inference
    spec:
      containers:
      - name: ps-inference
        image: ps-inference:latest
        ports:
        - containerPort: 80
```

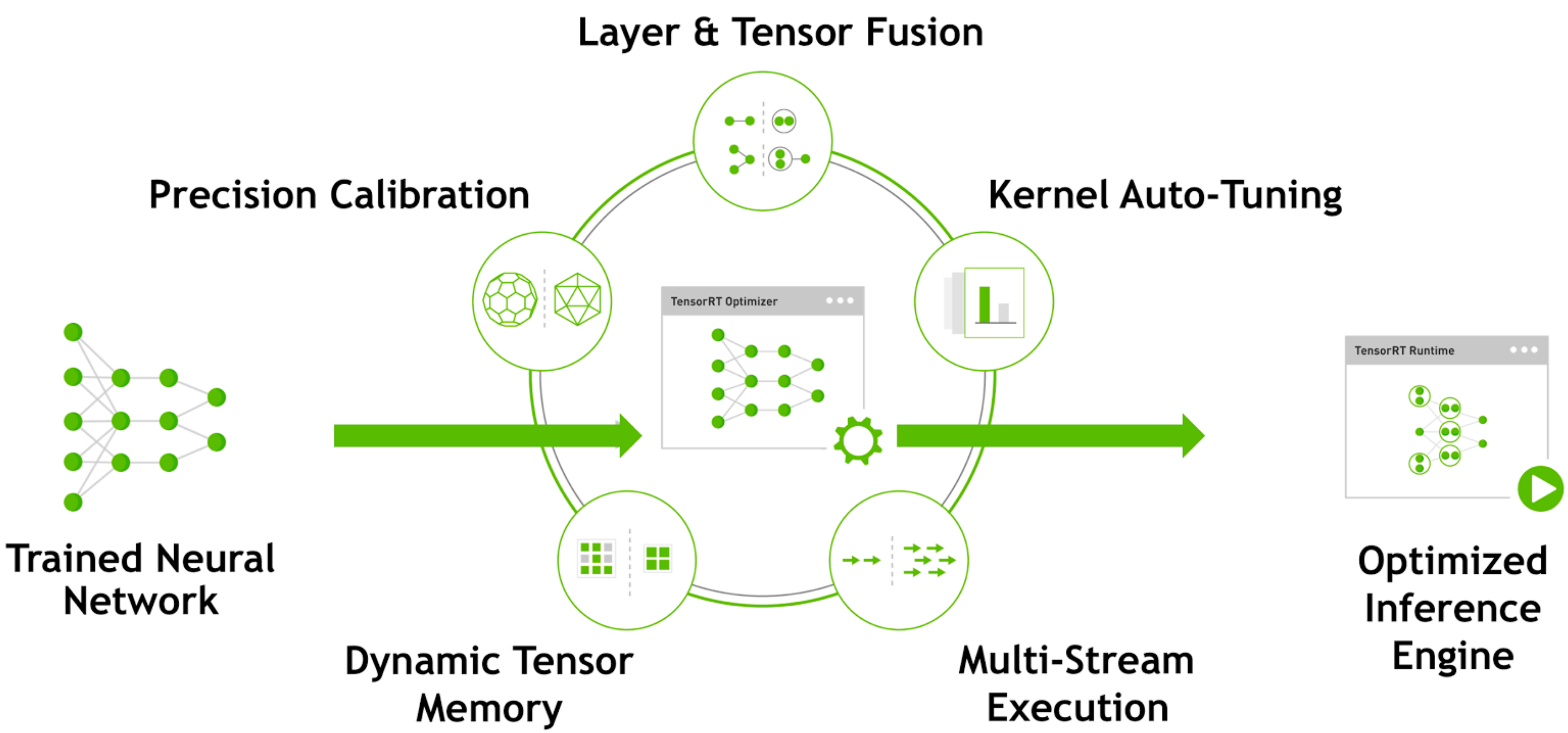
Ref: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Step 3-2. Deploy on EC2 directly

```bash
#!/bin/bash

docker kill ps-inference || true
docker rm ps-inference || true
docker run -d -p 35000:8000 \
       --runtime=nvidia \
       -e NVIDIA_VISIBLE_DEVICES=0 \
       docker-registry.aitrics.com/ps-inference:gpu \
       --host=0.0.0.0 \
       --port=8000 \
       --sentry-dsn=http://somesecretstring@sentry.aitricsdev.com/13 \
       --gpus=0 \
       --character-model=best_model.params/faster_rcnn_renet101_v1b \
       --shape-model=scnet_shape.params/ResNet50_v2 \
       --color-model=scnet_color.params/ResNet50_v2 \
       --s3-bucket=aitrics-research \
       --s3-path=faster_rcnn/result/181109 \
       --model-path=.data/models \
       --aws-access-key=*** \
       --aws-secret-key=***
```

Step 4. Using TensorRT Inference Server

- TensorRT is a high-performance deep learning inference optimizer and runtime engine for production deployment of deep learning applications.



Ref: https://developer.nvidia.com/tensorrt

Step 4. Using TensorRT Inference Server

- Use Tensorflow or Caffe to apply TensorRT easily
  - Consider TensorRT when you build model
  - Some operations might not be supported

- Add some TensorRT related code in Python script
  - Use TensorRT docker image to run inference server.

Step 4. Using TensorRT Inference Server

```
# TensorRT From ONNX with Python Example

import tensorrt as trt

with builder = trt.Builder(TRT_LOGGER) as builder, \
               builder.create_network() as network, \
               trt.OnnxParser(network, TRT_LOGGER) as parser:
  with open(model_path, 'rb') as model:
    parser.parse(model.read())



...
```

Ref: https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html#import_onnx_python

Step 4. Using TensorRT Inference Server

```
# Dockerfile
# https://github.com/NVIDIA/tensorrt-inference-server/blob/master/Dockerfile

FROM aitrics/tensorrt-inference-server:cuda9-cudnn7-onnx

ADD . /ps-inference/

ENTRYPOINT ["/ps-inference/run.sh"]
```

Ref: https://github.com/onnx/onnx-tensorrt/blob/master/Dockerfile

Step 5. Terraform

- You can also find our inference cluster as a code!
  - https://github.com/AITRICS/kono
  - Configure your settings and test example microservices and inference farm with terraform!

Case Study. Kubeflow

- The Kubeflow project is dedicated to making deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable.
- https://www.kubeflow.org/
- When to use
  - You want to train/serve TensorFlow models in different environments (e.g. local, on prem, and cloud)
  - You want to use Jupyter notebooks to manage TensorFlow training jobs
  - You want to launch training jobs that use resources – such as additional CPUs or GPUs – that aren't available on your personal computer
  - You want to combine TensorFlow with other processes
  - For example, you may want to use tensorflow/agents to run simulations to generate data for training reinforcement learning models.

Ref: https://www.kubeflow.org/

Case Study. Kubeflow

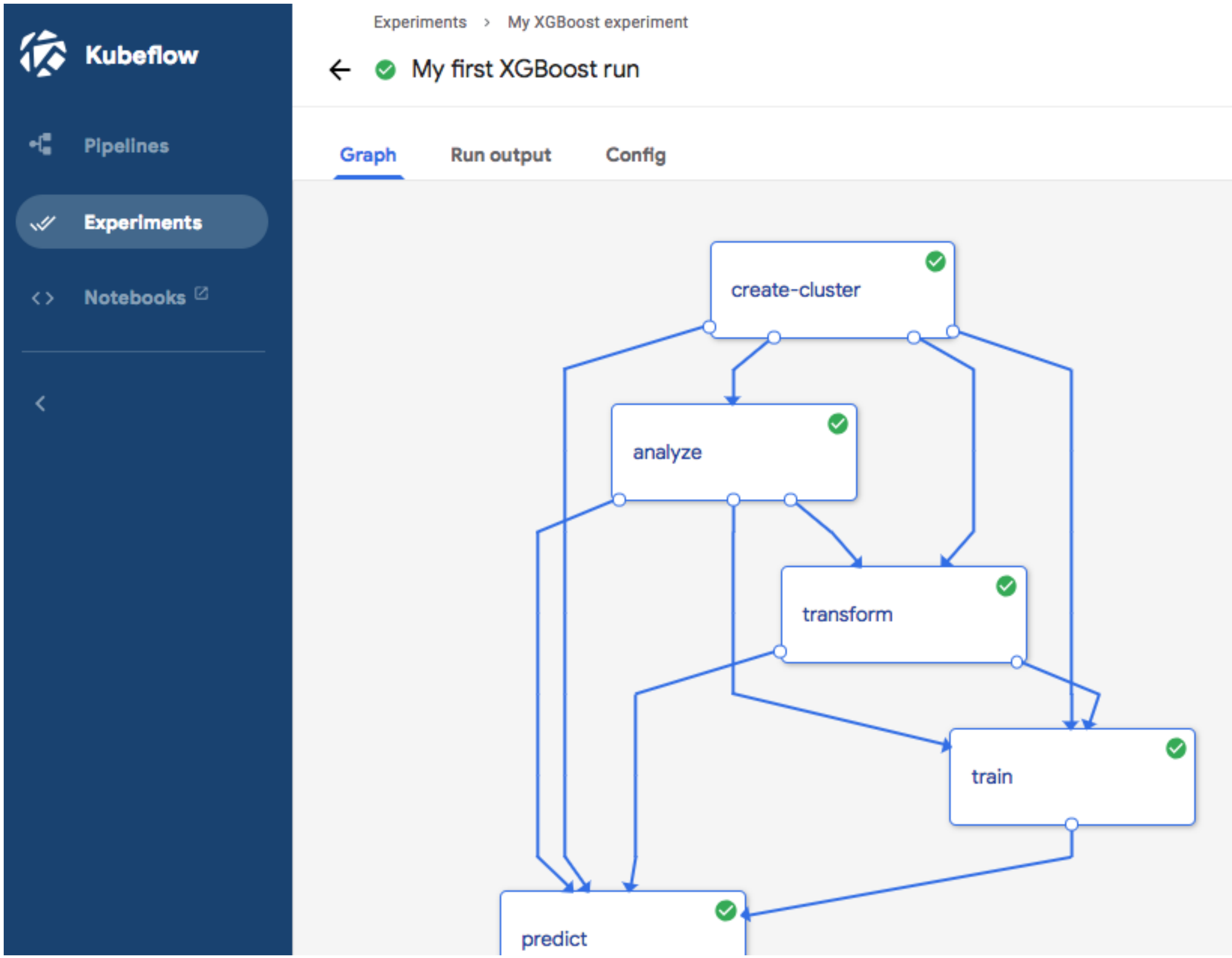
- **Re-define a machine learning workflow object with kubernetes object**
- **Run training, inferencing, serving, and other things on kubernetes**
- Need ksonnet, configuration management tools for kubernets manifests
  - https://www.kubeflow.org/docs/components/ksonnet/
- Only works well with tensorflow (support for PyTorch, MPI, MXNet is on alpha/beta stage)
- Some functions only works on GKE cluster
- Very early stage product (less than 1 year)

TF Job

```yaml
# TF Job
# https://www.kubeflow.org/docs/components/tftraining/

apiVersion: kubeflow.org/v1beta1
kind: TFJob
metadata:
  labels:
    experiment: experiment10
  name: tfjob
  namespace: kubeflow
spec:
  tfReplicaSpecs:
    Ps:
      replicas: 1
      template:
        metadata:
          creationTimestamp: null
        spec:
          containers:
          - args:
            - python
            - tf_cnn_benchmarks.py
...
```

Ref: https://www.kubeflow.org/docs/components/tftraining/

Pipelines



Ref: https://www.kubeflow.org/docs/components/tftraining/

AITRICS

Conclusion

Summary

- You can build your own training cluster!
- You also can build your own inference cluster!

- If you do not want to get your hands dirty, you can use our terraform code and cli.
    - https://github.com/AITRICS/kono

What's next?

What's next topic (which is not covered)?

- Monitoring resources
  - Prometheus + cAdvisor
  - https://devopscube.com/setup-prometheus-monitoring-on-kubernetes/
- Training models from real-time data streaming
  - Real-time one Kafka Stream (+ Spark Streaming) + Online learning
  - https://github.com/kaiwaehner/kafka-streams-machine-learning-examples
- Large-scale data preprocessing
  - Apache Spark

What's next topic (which is not covered)?

- Distributed training
  - Polyaxon supports: https://github.com/polyaxon/polyaxon-examples/blob/master/in_cluster/tensorflow/cifar10/polyaxonfile_distributed.yml
  - Use horovod: https://github.com/horovod/horovod
- Model & Data Versioning
  - https://github.com/iterative/dvc

Thank you!

Jaeman An <jaeman@aitrics.com>

Contact:
Jaeman An <jaeman@aitrics.com>
Yongseon Lee <yongseon@aitrics.com>
Tony Kim <tonykim@aitrics.com>

AITRICS

www.aitrics.com contact@aitrics.com

Tel. +82 2 569 5507 Fax. +82 2 569 5508