

Lab 4: Deploy Opendatacam (powered by darknet YOLO NN) via Kubernetes

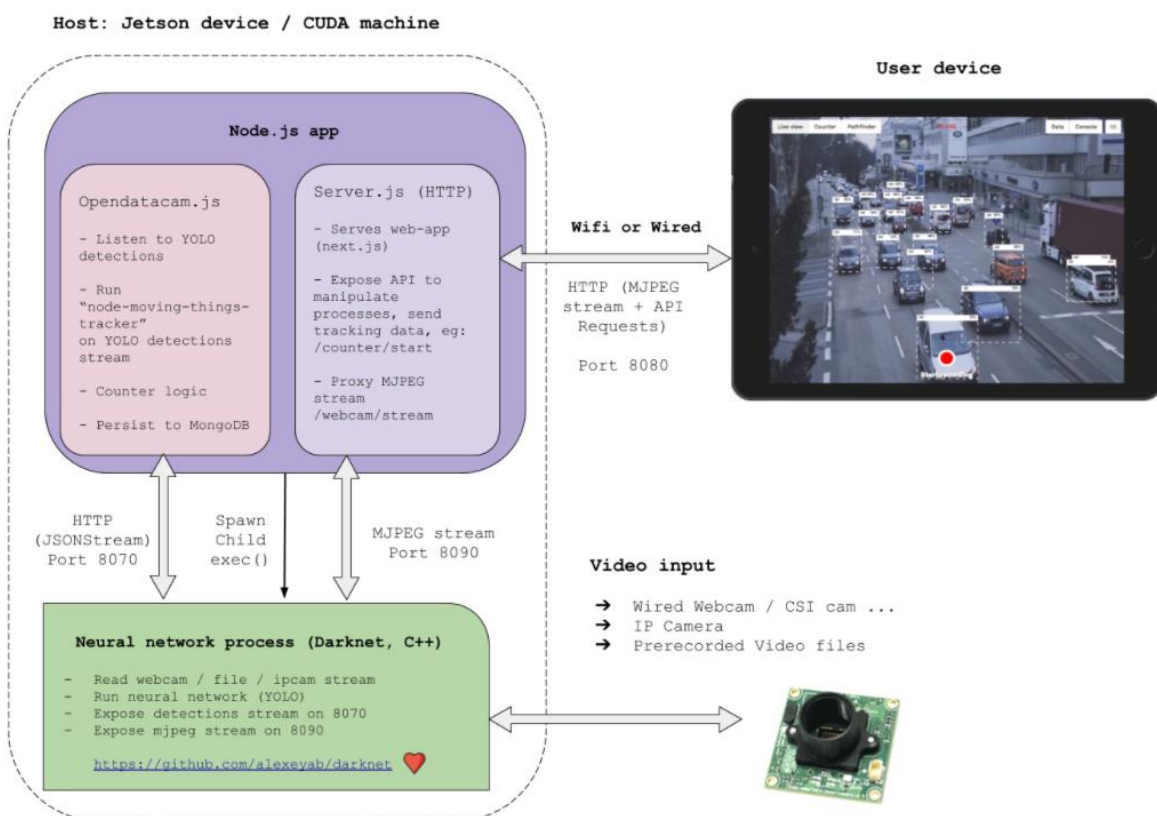
Objectives

- Use Kubernetes CLI to create different types of Kubernetes resources.
- Deploy Opendatacam via kubectl CLI.
- Access Opendatacam Web UI and explore its features.

Deliverables

Compulsory (10%)

- K8s YAML files for Opendatacam and Opendatacam-mongodb deployment and service.
- Screenshot of [Counter] and [Data] from Opendatacam web UI.



Task 1: Install lightweight Kubernetes k3s to Xavier NX**Step 1: Install k3s to NX. NX will become both the master and worker node.**

1. Login to your Xavier NX machine and install k3s. This will turn the NX into both Master and Worker Node.

```
sudo swapoff -a  
curl -sfL https://get.k3s.io | sh -
```

2. Check that k3s is running.

```
kubectl get nodes -o wide
```

If kubectl failed to connect, copy the cluster admin credentials to your home folder/.kube.

```
mkdir ~/.kube  
sudo cp /etc/rancher/k3s/k3s.yaml ~/.kube
```

Task 2: Getting to know 'kubectl' CLI**Step 1. Get used to kubectl syntax**

```
kubectl [command] [TYPE] [NAME] [flags]
command: create, get, describe, delete
TYPE: resource type
NAME: the name of the resource
flags: Specifies optional flags.
```

1. Check cluster info. Double check the port is 6443 which runs K8s API server.

```
kubectl cluster-info
```

2. Check node info.

```
kubectl get nodes
```

Optional: You could get full node info such as IP address and Linux kernel version by appending -o wide.

3. Get more info about the nodes.

```
kubectl describe node <NODE_NAME>
```

4. Get all pods in namespace

```
kubectl get pods --all-namespaces
```

Notice that some of control plane components such as kube-api-server, kube-controller, kube-scheduler, and etcd is missing here. This is because k3s embed them into k3s process itself instead of container. You will see those components running in kube-system namespace if you install K8s instead.

5. Get all resources in all namespace

```
kubectl get all -A
```

6. Log in into container in a Pod

```
kubectl exec -it <pod_id> -- <command>
```

7. View the container log file in a Pod

```
kubectl logs -f <pod_id>
```

Task 3: Deploy Opendatacam Pod

Opendatacam runs darknet YOLO internally.

Step 1. Create K8s ConfigMap to store Opendatacam program configuration settings.

1. Create a folder named [opendatacam] to store the Kubernetes YAML file and Opendatacam configuration file.
2. Create a config.json file with the following content. The settings uses a local video file that is pre-built into the container image, runs YOLOv4 model.

```
{
  "OPENDATACAM_VERSION": "3.0.2",
  "PATH_TO_YOLO_DARKNET" : "/var/local/darknet",
  "VIDEO_INPUT": "file",
  "NEURAL_NETWORK": "yolov4",
  "VIDEO_INPUTS_PARAMS": {
    "file": "opendatacam_videos/demo.mp4",
    "usbcam": "v4l2src device=/dev/video0 ! video/x-raw, framerate=30/1,
width=640, height=360 ! videoconvert ! appsink",
    "raspberrycam": "nvarguscamerasrc ! video/x-
raw(memory:NVMM),width=1280, height=720, framerate=30/1, format=NV12 !
nvvidconv ! video/x-raw, format=BGRx, width=640, height=360 !
videoconvert ! video/x-raw, format=BGR ! appsink",
    "remote_cam": "YOUR IP CAM STREAM (can be .m3u8, MJPEG ...), anything
supported by opencv",
    "remote_hls_gstreamer": "souphttpsrc
location=http://YOUR_HLSSTREAM_URL_HERE.m3u8 ! hlsdemux ! decodebin !
videoconvert ! videoscale ! appsink"
  },
  "TRACKER_SETTINGS": {
    "objectMaxAreaInPercentageOfFrame": 80,
    "confidence_threshold": 0.2,
    "iouLimit": 0.05,
    "unMatchedFrameTolerance": 5
  },
  "COUNTER_SETTINGS": {
    "minAngleWithCountingLineThreshold": 5,
    "computeTrajectoryBasedOnNbOfPastFrame": 5
  },
  "VALID_CLASSES": ["*"],
  "DISPLAY_CLASSES": [
    { "class": "bicycle", "hexcode": "1F6B2"},
    { "class": "person", "hexcode": "1F6B6"},
    { "class": "truck", "hexcode": "1F69B"},
    { "class": "motorbike", "hexcode": "1F6F5"},
    { "class": "car", "hexcode": "1F697"},
    { "class": "bus", "hexcode": "1F68C"}
  ],
  "PATHFINDER_COLORS": [
    "#1f77b4",
    "#ff7f0e",
    "#2ca02c",
```

```

    "#d62728",
    "#9467bd",
    "#8c564b",
    "#e377c2",
    "#7f7f7f",
    "#bcbd22",
    "#17becf"
  ],
  "COUNTER_COLORS": {
    "yellow": "#FFE700",
    "turquoise": "#A3FFF4",
    "green": "#a0f17f",
    "purple": "#d070f0",
    "red": "#AB4435"
  },
  "NEURAL_NETWORK_PARAMS": {
    "yolov4": {
      "data": "cfg/coco.data",
      "cfg": "cfg/yolov4-416x416.cfg",
      "weights": "yolov4.weights"
    },
    "yolov4-tiny": {
      "data": "cfg/coco.data",
      "cfg": "cfg/yolov4-tiny.cfg",
      "weights": "yolov4-tiny.weights"
    }
  },
  "TRACKER_ACCURACY_DISPLAY": {
    "nbFrameBuffer": 300,
    "settings": {
      "radius": 3.1,
      "blur": 6.2,
      "step": 0.1,
      "gradient": {
        "0.4": "orange",
        "1": "red"
      }
    },
    "canvasResolutionFactor": 0.1
  },
  "MONGODB_URL": "mongodb://opendatacam-mongo:27017",
  "PORTS": {
    "app": 8080,
    "darknet_json_stream": 8070,
    "darknet_mjpeg_stream": 8090
  }
}

```

3. Create a K8s ConfigMap resource.

```
kubectl create configmap opendatacam --from-file=config.json --dry-run -o yaml | kubectl apply -f -
```

4. Check that the resource is created.

```
kubectl get configmap
```

5. View the ConfigMap resource content. Notice that config.json is now embedded as a configmap.

```
kubectl describe configmap/opendatacam
```

Step 2: Create a K8s Deployment for Opendatacam

1. Create a file called opendatacam-deployment.yaml.
2. Copy and paste the skeleton Deployment YAML from deployment-template.
3. Create a label with two keys/value pairs: [app: opendatacam] and [tier: frontend]. We will later create a K8s Service Resource that will select Pods created by this Deployment.
4. Use the container image [opendatacam/opendatacam:v3.0.2-xavier].
5. Override the container launch command with the command ["/bin/bash"].
6. Fill in the argument with ["-c", "/var/local/opendatacam/launch.sh"]. This is a shell script.
7. Create a volume named [opendatacam-config] under spec.template.spec. Configure it as type [configMap] and use the same ConfigMap name you created in step 1.
8. Create a volumeMounts under spec.template.spec.containers.volumes and mount it from the volume name you created from 4. This container is then able to read the configuration file from this mountPath.
9. Expose 3 ports, 8080 (Opendatacam Web UI), 8070 (YOLO JSON stream), and 8090 (YOLO MJPEG images)
10. Assign privilege mode so that the container can use NX GPU.
11. Apply the deployment to K8s API server via kubectl.

Note: The opendatacam container image size is 3.2GB and might take around 5 minutes to complete.

Step 3: Create K8s Service

Objective: Create a LoadBalancer service that will bind port 8080, 8070, and 8090 from client to the pods with the same ports.

1. Create a file called opendatacam-service.yaml.
2. Copy and paste the skeleton Service YAML from service-template.
3. Create a label with key [app] and value [opendatacam].
4. Assign the type as [LoadBalancer].
5. Use the key [app: opendatacam] and [tier: frontend] under spec.selector. This will tell the service to only select pods with the matching key value pairs.

6. Create 3 ports for 8080, 8070, and 8090. Use the same number for name, port, and targetPort to make things easier.
7. Apply the service to K8s API server via kubectl.

Task 4: Deploy MongoDB Pod

MongoDB will be used to store the recordings from Opendatacam

Step 1: Create a K8s PersistentVolumeClaim (PVC) resource for MongoDB

We will allocate 10 GB of storage, then later bind this to MongoDB Pod.

1. Create a file called `opendatacam-mongo-pvc.yaml`
2. Paste the following content:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pv-claim
  labels:
    app: opendatacam
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Step 2: Create a K8s Deployment for MongoDB

1. Create a file called `opendatacam-mongo-deployment.yaml`.
2. Copy and paste the skeleton Deployment YAML from `deployment-template`.
3. Create a label with two keys/value pairs: `[app: opendatacam]` and `[tier: mongo]`. We will later create a K8s Service Resource that will select Pods created by this Deployment.
4. Use the container image `[mongo]`.
5. Create a volume named `[mongodb-persistent-storage]` under `spec.template.spec.volumes`. Configure it as type `[persistentVolumeClaim]` and use the same `[claimName]` you created in step 1.
6. Create a `volumeMounts` under `spec.template.spec.containers.volumeMounts` and mount it from the volume name you created from 5. Set the `mountPath` to `[/data/db]`.
7. Expose 1 port, 27017.
8. Apply the deployment to K8s API server via `kubectl`.

Notice that you do not need to specify the command and argument. we will simply use the default command specified by the developer when building the mongo container image.

Step 3: Create K8s Service

Objective: Create a ClusterIP service that will bind port 27017 from client network request to the MongoDB port listening at the same ports.

1. Create a file called `opendatacam-mongo-service.yaml`.
2. Copy and paste the skeleton Service YAML from `service-template`.
3. Set the service name to `[opendatacam-mongo]`.
4. Create a label with key `[app]` and value `[opendatacam]`.
5. Assign the type as `[ClusterIP]`.
6. Use the two labels created on Deployment `[app: opendatacam]` and `[tier: mongo]` as `spec.selector`.
This will tell the service to only select pods with the matching key value pairs.
7. Create port 27017. Use the same number for name, port, and targetPort to make things easier.
8. Apply the service to K8s API server via `kubectl`.

Task 5: Access the Opendatacam Web UI

1. To access the web UI, first get the port opened by the load balancer to the service [opendatacam] on port 8080.

```
kubectl get svc
```

Output

```
root@user-desktop:/home/user# kubectl get svc
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
kubernetes                          ClusterIP           10.43.0.1        <none>            443/TCP
72m
opendatacam-mongo                   ClusterIP           10.43.92.214     <none>            27017/TCP
70m
opendatacam                         LoadBalancer        10.43.217.54     192.168.85.55    8080:30921/TCP,8070:30769/TCP,8090:30326/TCP
70m
```

In this case, the port is 30921. Your case might varies.

2. Open a web browser and go to the following URL: <http://<NX-IP>:30921>. You should see a initializing UI.

Task 6: Use Opendatacam Features

- Explore [Live View]

Live view shows you the current real live feed.

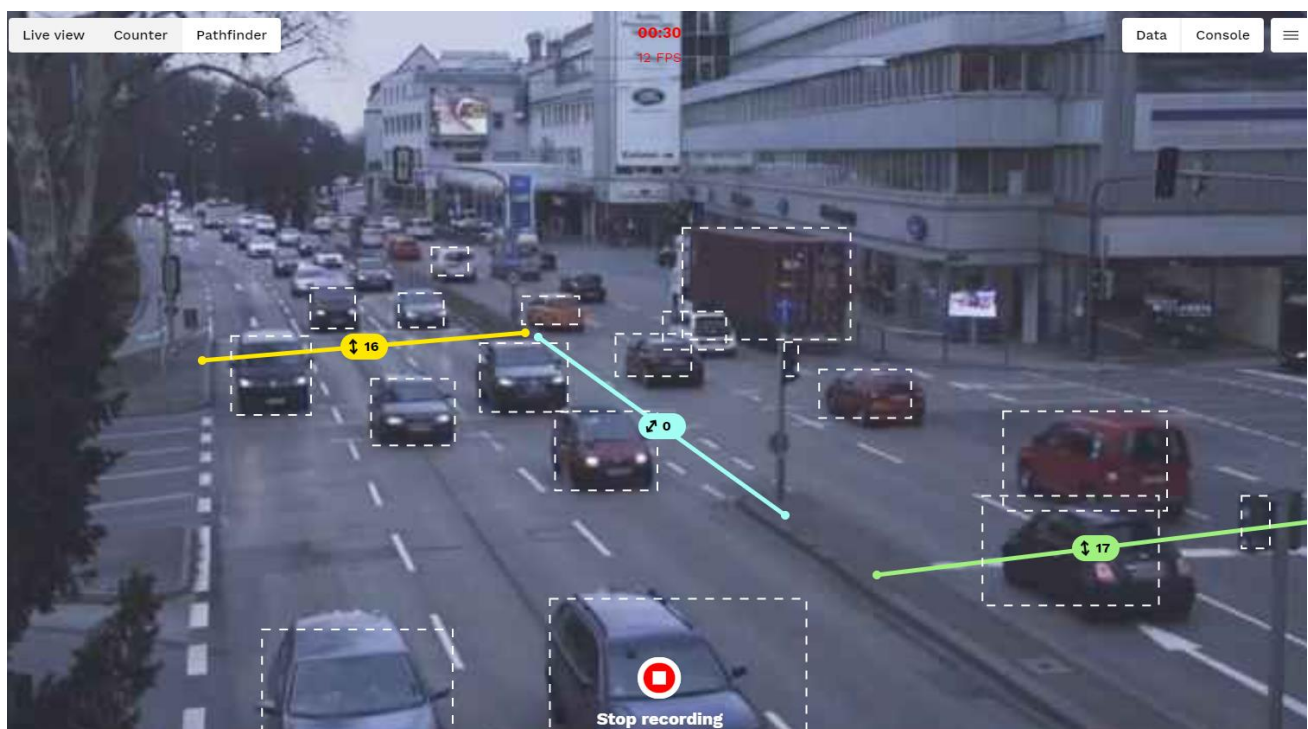
- Explore [Pathfinder]

Pathfinder will draw traces of line for each moving object. Each object is assigned with a color so that you can tell its moving path.

- Explore [Counter]

Opendatacam allow you to draw a line that will count any object that go pass it. To do so, select [Counter], draw a line. Then click [Start Recording]. A counter will be shown next to the direction for each line. You can also click [Data] at the top right corner to see the live stats.

-



Click [Data] button to see the live real.

Live view

Counter

Pathfinder

00:49
12 FPS

Data

Console




☰




Apr 02, 2022 02:58 am - Ongoing demo.mp4

Counter




Download: JSON CSV




lane-left ● ↓

0  0  0 




0  23  0 




lane-middle ● ↗

0  0  0 

0  0  0 

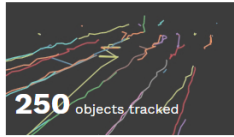
lane-right ● ↓

0  0  0 

0  13  0 

Tracker

Download: JSON



250 objects tracked

Skeleton Template K8s YAML Files

deployment-template

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    ...
  name: ...
spec:
  replicas: 1
  selector:
    matchLabels:
      ...
  template:
    metadata:
      labels:
        ...
    spec:
      containers:
      - image: ...
        command: ...
        args: ...
        name: opendatacam
        ports:
        - containerPort: ...
          ...
        resources: {}
        securityContext:
          privileged: ...
        volumeMounts:
        - mountPath: ...
          name: ...
          subPath: "config.json"
      restartPolicy: Always
      volumes:
      - name: ...
        configMap:
          name: ...
          items:
          - key: config.json
            path: config.json
```

service-template

```
apiVersion: v1
kind: Service
metadata:
  labels:
    ...
  name: ...
spec:
```

```
ports:
- name: "..."  
  port: ...  
  targetPort: ...  
...  
selector:  
  ...  
type: ...
```

References

<https://github.com/opendatacam/opendatacam>