DGX-2 / Kubernetes Refresher Bootcamp

Nicholas Schenone

Feb 4th, 2020

Presentation Overview

Part I: Kubernetes

- What is Kubernetes (K8s)?
- Basic K8s concepts
  - Pod
  - Replica Set
  - Deployment
  - Job
  - Storage
    - Persistent Volume
    - Persistent Volume Claim
    - Storage Class
  - Service
  - Secret
- K8s Best Practices
- K8s TensorFlow Deployment / Job Example

Part II: DeepOps

- What is DeepOps?
- DeepOps Architecture
- Deploying a K8s cluster via DeepOps
  - Clone repo
  - Set up provisioning machine
  - Copy SSH keys (optional)
  - Configure Inventory files
  - Deploy cluster
- Modifying DeepOps Cluster
  - Scale cluster (add/remove nodes)
  - Delete cluster
- Deploying DeepOps Services
  - Kubernetes Dashboard
  - Persistent storage via Rook Ceph
  - Monitoring via Grafana and Prometheus
  - Kubeflow (Pipelines & JupyterLab)
    - Kubeflow Pipeline Demo
- Troubleshooting: GPU's Not Connecting


Part I: Kubernetes (K8s)

What is Kubernetes?

- **Kubernetes (K8s) is a container deployment and orchestration tool:**
  - *Docker documentation: "A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another."*
- K8s will create, manage, and delete these containers to run applications flexibly and at scale

kubernetes

Basic Kubernetes Concepts - Pod

- A Pod (as in a pod of whales or pea pod) is a group of one or more containers (such as Docker containers), with shared storage/network, and a specification for how to run the containers.
- Pods aren't intended to be treated as durable entities. They won't survive scheduling failures, node failures, or other evictions, such as due to lack of resources, or in the case of node maintenance.
- In general, users shouldn't need to create Pods directly. They should almost always use controllers even for singletons, for example, Deployments. Controllers provide self-healing with a cluster scope, as well as replication and rollout management.

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

Basic Kubernetes Concepts - Replica Set

A ReplicaSet fulfills its purpose by creating and deleting Pods as needed to reach the desired number of pods. When a ReplicaSet needs to create new Pods, it uses its Pod template.

A ReplicaSet ensures that a specified number of pod replicas are running at any given time

However, a Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features

This actually means that you may never need to manipulate ReplicaSet objects: use a Deployment instead, and define your application in the spec section.

Within deployment:

```
...
spec:
  replicas: 3
...
```

Basic Kubernetes Concepts - Deployment

- A Deployment controller provides declarative updates for Pods and ReplicaSets.
- You describe a desired state in a Deployment, and the Deployment controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.
- Create a Deployment to rollout a ReplicaSet. The ReplicaSet creates Pods in the background.

- We will be going over an example of a deployment later using a TensorFlow container

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Basic Kubernetes Concepts – Job

- 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. When a specified number of successful completions is reached, the task (ie, Job) is complete. Deleting a Job will clean up the Pods it created.
- A simple case is to create one Job object in order to reliably run one Pod to completion. The Job object will start a new Pod if the first Pod fails or is deleted (for example due to a node hardware failure or a node reboot).
- You can also use a Job to run multiple Pods in parallel.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

- We will be going over an example of a job later using aTensorFlow container

GROUPWARE TECHNOLOGY

Basic Kubernetes Concepts – Storage

Basic Kubernetes Concepts – Persistent Volume

- A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes.
- It is a resource in the cluster just like a node is a cluster resource.
- PVs are volume plugins like Volumes, but have a lifecycle independent of any individual pod that uses the PV.
- This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

Basic Kubernetes Concepts – Persistent Volume Claim

- A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a pod. Pods consume node resources and PVCs consume PV resources.
- While PersistentVolumeClaims allow a user to consume abstract storage resources, it is common that users need PersistentVolumes with varying properties, such as performance, for different problems.
- Cluster administrators need to be able to offer a variety of PersistentVolumes that differ in more ways than just size and access modes, without exposing users to the details of how those volumes are implemented. For these needs there is the StorageClass resource.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Basic Kubernetes Concepts – Storage Class

- Each StorageClass contains the fields provisioner, parameters, and reclaimPolicy, which are used when a PersistentVolume belonging to the class needs to be dynamically provisioned.
- The name of a StorageClass object is significant, and is how users can request a particular class. Administrators set the name and other parameters of a class when first creating StorageClass objects, and the objects cannot be updated once they are created.
- Administrators can specify a default StorageClass just for PVCs that don't request any particular class to bind to.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
  - debug
volumeBindingMode: Immediate
```

Basic Kubernetes Concepts – Service

- In Kubernetes, a Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service). The set of Pods targeted by a Service is usually determined by a selector.
- For example, consider a stateless image-processing backend which is running with 3 replicas. Those replicas are fungible—frontends do not care which backend they use. While the actual Pods that compose the backend set may change, the frontend clients should not need to be aware of that, nor should they need to keep track of the set of backends themselves.
- The Service abstraction enables this decoupling.

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

Basic Kubernetes Concepts – Secret

- Kubernetes secret objects let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys. Putting this information in a secret is safer and more flexible than putting it verbatim in a Pod definition or in a container image
- Users can create secrets, and the system also creates some secrets.
- To use a secret, a pod needs to reference the secret. A secret can be used with a pod in two ways: as files in a volume mounted on one or more of its containers, or used by kubelet when pulling images for the pod.

```
echo -n 'admin' | base64
YWRtaW4=
echo -n '1f2d1e2e67df' | base64
MWYyZDFlMmU2N2Rm
```

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

Kubernetes Best Practices

General Configuration

- When defining configurations, specify the latest stable API version.
- Configuration files should be stored in version control before being pushed to the cluster. This allows you to quickly roll back a configuration change if necessary. It also aids cluster re-creation and restoration.
- Write your configuration files using YAML rather than JSON. Though these formats can be used interchangeably in almost all scenarios, YAML tends to be more user-friendly.
- Group related objects into a single file whenever it makes sense. One file is often easier to manage than several. See the guestbook-all-in-one.yaml file as an example of this syntax.
- Note also that many kubectl commands can be called on a directory. For example, you can call kubectl apply on a directory of config files.
- Don't specify default values unnecessarily: simple, minimal configuration will make errors less likely.
- Put object descriptions in annotations, to allow better introspection.

Kubernetes Best Practices (cont'd)

"Naked" Pods vs ReplicaSets, Deployments, and Jobs

- Don't use naked Pods (that is, Pods not bound to a ReplicaSet or Deployment) if you can avoid it. Naked Pods will not be rescheduled in the event of a node failure.
- A Deployment, which both creates a ReplicaSet to ensure that the desired number of Pods is always available, and specifies a strategy to replace Pods (such as RollingUpdate), is almost always preferable to creating Pods directly, except for some explicit restartPolicy: Never scenarios. A Job may also be appropriate.

Kubernetes Best Practices (cont'd)

Services

- Create a Service before its corresponding backend workloads (Deployments or ReplicaSets), and before any workloads that need to access it. When Kubernetes starts a container, it provides environment variables pointing to all the Services which were running when the container was started.
- *This does imply an ordering requirement* - any Service that a Pod wants to access must be created before the Pod itself, or else the environment variables will not be populated. DNS does not have this restriction.

Kubernetes Best Practices (cont'd)

Storage

- Always include Persistent Volume Claims in the config (Deployment, Job, etc).
- Never include PVs in the config.
- Always create a default storage class.
- Give the user the option of providing a storage class name.

Example TensorFlow Deployment

- Pulls From NGC (NVIDIA GPU-Accelerated Containers) using K8s secret
- Creates 2 TF containers requesting 8 GPU's each
- Launched with command to sleep infinitely – will run until terminated
- Great for development within the container

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tensorflow-deployment
  namespace: gw-demo
  labels:
    app: tf-demo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: tf-demo
  template:
    metadata:
      labels:
        app: tf-demo
    spec:
      imagePullSecrets:
        - name: nvcr.dgxkey
      containers:
        - name: tensorflow-container
          image: nvcr.io/nvidia/tensorflow:19.07-py3
          command: ["/bin/sh", "-c"]
          args: ["sleep infinity"]
          resources:
            limits:
              nvidia.com/gpu: 8
```

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: gw-demo
```

Note:

It is generally best practice to include related objects into a single file. However we will be reusing this namespace in the next example so we've separated it into a different file.

GROUPWARE TECHNOLOGY

Example TensorFlow Job

Note:

It is generally best practice to include related objects into a single file. However we will be reusing this namespace in the next example so we've separated it into a different file.

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: gw-demo
```

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: tensorflow-example
  namespace: gw-demo
spec:
  backoffLimit: 5
  template:
    spec:
      imagePullSecrets:
        - name: nvcr.dgxkey
      restartPolicy: Never
      containers:
        - name: tensorflow-container
          image: nvcr.io/nvidia/tensorflow:19.07-py3
          command: ["/bin/sh", "-c"]
          args:
            - mpiexec --allow-run-as-root --bind-to socket -np 16 python /workspace/nvidia-examples/cnn/resnet.py --layers=50 --precision=fp16 --batch_size=512
          resources:
            limits:
              nvidia.com/gpu: 16

```


Part II: DeepOps

Presentation Overview

Part I: Kubernetes

- What is Kubernetes (K8s)?
- Basic K8s concepts
  - Pod
  - Replica Set
  - Deployment
  - Job
  - Storage
    - Persistent Volume
    - Persistent Volume Claim
    - Storage Class
  - Service
  - Secret
- K8s Best Practices
- K8s TensorFlow Deployment / Job Example

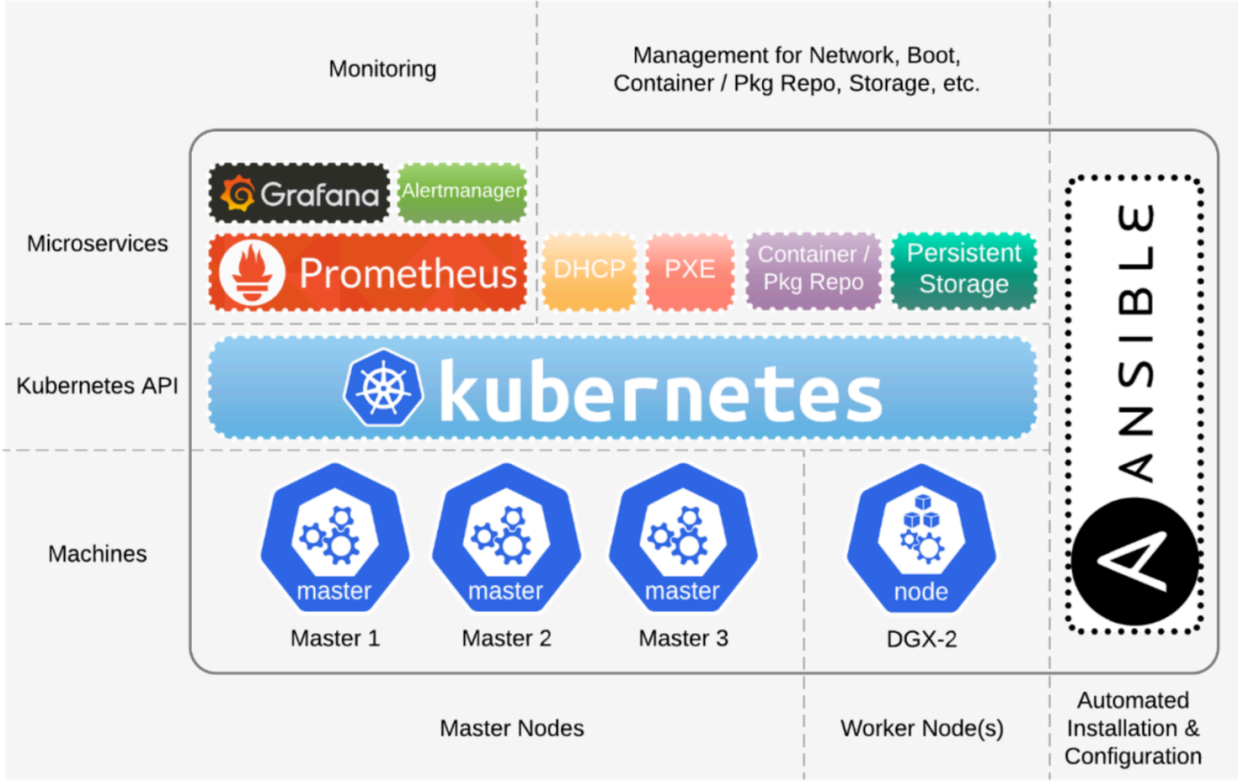
Part II: DeepOps

- What is DeepOps?
- DeepOps Architecture
- Deploying a K8s cluster via DeepOps
  - Clone repo
  - Set up provisioning machine
  - Copy SSH keys (optional)
  - Configure Inventory files
  - Deploy cluster
- Modifying DeepOps Cluster
  - Scale cluster (add/remove nodes)
  - Delete cluster
- Deploying DeepOps Services
  - Kubernetes Dashboard
  - Persistent storage via Rook Ceph
  - Monitoring via Grafana and Prometheus
  - Kubeflow (Pipelines & JupyterLab)
    - Kubeflow Pipeline Demo
- Troubleshooting: GPU's Not Connecting

What is DeepOps?

- The DeepOps project encapsulates best practices in the deployment of GPU server clusters and sharing single powerful nodes (such as NVIDIA DGX Systems). DeepOps can also be adapted or used in a modular fashion to match site-specific cluster needs. For example:
- An on-prem, air-gapped data center of NVIDIA DGX servers where DeepOps provides end-to-end capabilities to set up the entire cluster management stack
- An existing cluster running Kubernetes where DeepOps scripts are used to deploy Kubeflow and connect NFS storage
- An existing cluster that needs a resource manager / batch scheduler, where DeepOps is used to install Slurm, Kubernetes, or a hybrid of both
- A single machine where no scheduler is desired, only NVIDIA drivers, Docker, and the NVIDIA Container Runtime

DeepOps Architecture



Deploying a K8s Cluster via DeepOps

- Clone repo
  - `git clone --recurse-submodules https://github.com/NVIDIA/deepops.git`
- Set up provisioning machine (from deepops directory)
  - `./scripts/setup.sh`
- Copy SSH keys (optional)
  - `ssh-copy-id user@hostname.example.com`
- Configure inventory file
  - See next slide for details
- Deploy cluster
  - `ansible-playbook -i config/inventory playbooks/k8s-cluster.yml`
    - If SSH requires a password, add: `-k`
    - If sudo on remote machine requires a password, add: `-K`
    - If SSH user is different than current user, add: `-u ubuntu`

DeepOps Inventory File

- deeops/config/inventory

```
######
# ALL NODES
######
[all]
mgmt-node01     ansible_host=127.0.0.1 ip=127.0.0.1
worker-node01   ansible_host=127.0.0.2 ip=127.0.0.2
worker-node02   ansible_host=127.0.0.3 ip=127.0.0.3
######
# KUBERNETES
######
[kube-master]
mgmt-node01     ansible_host=127.0.0.1 ip=127.0.0.1

[etcd]
mgmt-node01     ansible_host=127.0.0.1 ip=127.0.0.1

[kube-node]
worker-node01   ansible_host=127.0.0.2 ip=127.0.0.2
worker-node02   ansible_host=127.0.0.3 ip=127.0.0.3

[k8s-cluster:children]
kube-master
kube-node
```

Cluster Deployment

- https://drive.google.com/file/d/1fHfutI2T5HTo2up_IVVgR8ZRaCVwfpa7/view?usp=sharing

- `ansible-playbook -i config/inventory playbooks/k8s-cluster.yml`

Kubernetes Dashboard Deployment

- https://drive.google.com/file/d/1UgBVUXC0F_4Wi-PZo1u81IONGKi1zSdJ/view?usp=sharing

- Install:

```
./scripts/k8s_deploy_dashboard_user.sh
```

- Remove:

```
kubectl delete -f services/k8s-dashboard-admin.yml
```

Modifying DeepOps Cluster

- Add node (from kubespray/ directory)
  - Add new node to inventory/mycluster/hosts.yml (as well as ../config/inventory)
    - Make sure inventory files match
    - Example on next slide
  - Run script
    - `ansible-playbook -i inventory/mycluster/hosts.yml --become --become-user=root scale.yml`
      - If SSH requires a password, add: `-k`
      - If sudo on remote machine requires a password, add: `-K`
      - If SSH user is different than current user, add: `-u ubuntu`
- Remove node (from kubespray/ directory)
  - Add all nodes to inventory/mycluster/hosts.yml (as well as ../config/inventory)
  - Run script
    - `ansible-playbook -i inventory/mycluster/hosts.yml --become --become-user=root remove-node.yml --extra-vars "node=node-to-remove"`

DeepOps Hosts File

- deepops/kubespray/inventory/mycluster/hosts.yml

```
[all]
mgmt-node01      ansible_host=127.0.0.1 ip=127.0.0.1
worker-node01    ansible_host=127.0.0.2 ip=127.0.0.2
worker-node02    ansible_host=127.0.0.3 ip=127.0.0.3

[kube-master]
mgmt-node01

[kube-node]
worker-node01
worker-node02

[etcd]
mgmt-node01

[k8s-cluster:children]
kube-node
kube-master
```

Modifying DeepOps Cluster (cont'd)

- Delete cluster
  - Add all nodes to inventory/mycluster/hosts.yml (as well as ../config/inventory)
  - Run script (Method 1)
    - `ansible-playbook -i inventory/mycluster/hosts.yml --become --become-user=root reset.yml`
      - If SSH requires a password, add: `-k`
      - If sudo on remote machine requires a password, add: `-K`
      - If SSH user is different than current user, add: `-u ubuntu`
    - *NOTE: This does not do a complete tear-down of the cluster*
  - Run script (Method 2)
    - `ansible-playbook -i inventory/mycluster/hosts.yml --become --become-user=root remove-node.yml --extra-vars "node=node2,node3,node4,..."`
    - `ansible-playbook -i inventory/mycluster/hosts.yml --become --become-user=root remove-node.yml --extra-vars "node=node1"`

Deploying DeepOps Services

- Rook Ceph (Persistent Storage)
  - Pre-requisite for Kubeflow (creates a default storage class)
  - Has object storage, block storage, and shared file systems
  - May take up to 10 minutes for storage to be ready

- Monitoring (Grafana and Prometheus)
  - Grafana: http://mgmt:30200
  - Prometheus: http://mgmt:30500
  - Alertmanager: http://mgmt:30400

- Kubeflow (Pipelines and JupyterLab)
  - Machine Learning Toolkit for Kubernetes
  - Authentication details here

Install:
```
./scripts/k8s_deploy_rook.sh
```

Remove:
```
./scripts/rmrook.sh
```

Install:
```
./scripts/k8s_deploy_monitoring.sh
```
Remove:
```
./scripts/k8s_deploy_monitoring.sh  delete
```

Install:
```
./scripts/k8s_deploy_kubeflow.sh
```
Remove:
```
./scripts/k8s_deploy_kubeflow.sh  -D
```

Deploying DeepOps Services

- Monitoring (Grafana and Prometheus)
  - Grafana: http://10.168.128.83:30200/
    - Admin user: admin
    - Admin pass: deepops
  - Prometheus: http://10.168.128.83:30500/
  - Alertmanager: http://10.168.128.83:30400/

- Kubeflow (Pipelines and JupyterLab)
  - Machine Learning Toolkit for Kubernetes
  - Authentication details here
  - http://10.168.128.83:31380
  - Default credentials:
    - Email: admin@kubeflow.org
    - Pass: 12341234

Kubeflow Pipeline Demo

Troubleshooting: GPU's Not Connecting

- If you cannot schedule pods with GPU resources, check the log of the nvidia-device-plugin-daemonset pod on the GPU node to find the error.
- If the log says "Failed to initialize NVML: could not load NVML library":
  - Check to make sure `/etc/docker/daemon.json` on your GPU node is configured to use the nvidia-container-runtime.
    - Add '`"default-runtime": "nvidia",`'
  - If the docker runtime is configured correctly and you still cannot schedule GPU resources, try the following commands on your GPU node:
    - `systemctl daemon-reload`
    - `systemctl restart docker`

THANK YOU

GROUPWARE TECHNOLOGY

We Answer the Call