JTC14 Ansible Operators Labs

Lab0 - Lab information

The Operator Framework is an open source toolkit to manage Kubernetes native applications, called Operators, in an effective, automated, and scalable way.

In this Lab you will learn about Kubernetes Operator basics and create your first Ansible based Operator.

Lab sources

All the source code for the lab is available here:

https://github.com/niklaushirt/training

Lab overview

Lab 1: Provides a hands-on for creating a Ansible based Operator.

- Creating the Operator Project
- Creating the Operator API
- Creating the Operator Controller
- Build and deploy the Operator
- Create and deploy the Custom Resource
- Update the Custom Resource

Lab0 - Lab semantics

Nomenclatures

Shell Commands

The commands that you are going to execute to progress the Labs will look like this:

THIS IS AN EXAMPLE - DO NOT EXECUTE THIS!

```Bash
kubectl create -f redis-slave-service.yaml

 > Output Line 1
 > Output Line 2
 > Output Line 3
 ...
```

> **IMPORTANT NOTE:** The example output of a command is prefixed by ">" in order to make it more distinguishable.
>
> So in the above example you would only enter/copy-paste `kubectl create -f redis-slave-service.yaml` and the output from the command is "Output Line 1" to "Output Line 3"

Code Examples

Code examples are presented like this:

```YAML
apiVersion: lab.ibm.com/v1beta1
kind: MyResource
metadata:
  name: example
spec:
  size: 3
  image: busybox
```

This is only for illustration and is not being actively used in the Labs.

Lab 0 - Prepare the Lab environment

Before starting the Labs, let's make sure that we have the latest source code from the GitHub repository:

https://github.com/niklaushirt/training

Termnial icon in the bottom dock

1. Open a Terminal window by clicking on the Termnial icon in the left sidebar - we will use this extensively later as well

2. Execute the following commands to initialize your Training Environment

   ```Bash
   ./welcome.sh
   ```

   This will

   - pull the latest example code from my GitHub repository
   - start minikube if not already running
   - installs the registry
   - installs the Network Plugin (Cilium)
   - starts the Personal Training Environment

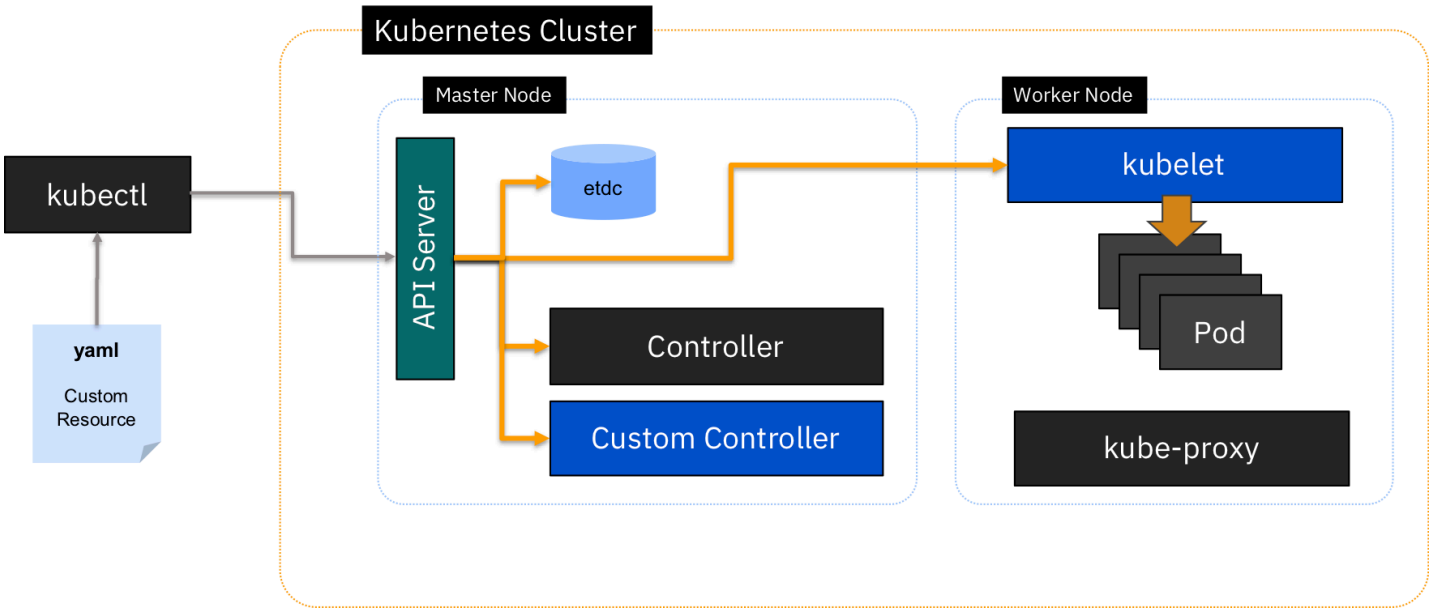
   > During this you will have to provide a name (your name) that will be used to show your progress in the Instructor Dashboard in order to better assist you.

Kubernetes Operators

In this Lab you will learn about Kubernetes Operator basics and create your first Ansible based Operator.

The Operator Framework is an open source toolkit to manage Kubernetes native applications, called Operators, in an effective, automated, and scalable way.

- Operators are a **design pattern** made public in a 2016 CoreOS blog post.
- The goal of an Operator is to **put operational knowledge into software**. Previously this knowledge only resided in the minds of administrators, various combinations of shell scripts or automation software like Ansible. It was outside of your Kubernetes cluster and hard to integrate.
- Operators implement and **automate common Day-1** (installation, configuration, etc) **and Day-2** (re-configuration, update, backup, failover, restore, etc.) **activities** in a piece of software running inside your Kubernetes cluster, by integrating natively with Kubernetes concepts and APIs.



Operators extend Kubernetes by allowing you to define a **Custom Controller** to watch your application and perform custom tasks based on its state (a perfect fit to automate maintenance of the stateful application we described above).

The application you want to watch is defined in Kubernetes as a new object: a **Custom Resource** (CR) that has its own yaml spec and object type (in K8s, a kind) that is understood by the API server.

That way, you can define any specific criteria in the custom spec to watch out for, and reconcile the instance when it doesn't match the spec. The way an operator's controller reconciles against a spec is very similar to native Kubernetes' controllers, though it is using mostly custom components.

Elements of an Operator implementation

- A **Custom Resource Definition** (CRD) spec that defines the format of the Custom Resource
- A **Custom Controller** to watch our application
    - Custom code within the new controller that dictates how to reconcile our CR against the spec
- An **Operator** to manage the Custom Controller
- A **Custom Resource** (CR) spec that defines the application we want to watch
- A **deployment** for the Operator and Custom Resource

Lab 2. Create and deploy your first Kubernetes Ansible Operator

The Operator SDK makes it easier to build Kubernetes native applications.

Lab 2 - Create the Lab Operator Project

In this part of the lab we will create a demo Ansible operator and deploy it to our minikube instance.

1. Create the `ansible-operator-frontend` directory

```bash
cd
mkdir ansible-operator
cd ~/ansible-operator
```

2. Create the `ansible-operator-frontend` Project

```bash
operator-sdk new ansible-operator-frontend --type=ansible --api-version=ansibl
enlab.ibm.com/v1beta1 --kind=MyAnsibleLabDemo

> INFO[0000] Creating new Ansible operator 'ansible-operator-frontend'.
> INFO[0000] Created deploy/service_account.yaml
> INFO[0000] Created deploy/role.yaml
> INFO[0000] Created deploy/role_binding.yaml
> INFO[0000] Created deploy/crds/lab_v1beta1_MyAnsibleLabDemo_crd.yaml
...
```

> You can ignore the errors concerning git

3. Change to the `ansible-operator-frontend` directory

```bash
cd ~/ansible-operator/ansible-operator-frontend
```

4. The basic file structure for the Ansible Operator has been created

```bash
ll

> total 36
> drwxr-x--- 7 training training 4096 Jul  1 17:29 ./
> drwxrwxr-x 3 training training 4096 Jul  1 17:29 ../
> drwxr-x--- 3 training training 4096 Jul  1 17:29 build/
> drwxr-x--- 3 training training 4096 Jul  1 17:29 deploy/
> drwxrwxr-x 7 training training 4096 Jul  1 17:29 .git/
> drwxr-x--- 5 training training 4096 Jul  1 17:29 molecule/
> drwxr-x--- 3 training training 4096 Jul  1 17:29 roles/
> -rw-r--r-- 1 training training  140 Jul  1 17:29 .travis.yml
> -rw-r--r-- 1 training training  121 Jul  1 17:29 watches.yaml
```

Lab 2 - Create the Lab Operator API

With the above, the API has already been created (unlike for GO operators) and added to the new Custom Resource (CR), with APIVersion `ansiblenlab.ibm.com/v1beta1` and Kind `MyAnsibleLabDemo`.

1. Add the `deployment.image` field to the Custom Resource

   ```Bash
   gedit ~/ansible-operator/ansible-operator-frontend/deploy/crds/ansiblenlab_v1beta1_myansiblelabdemo_cr.yaml &
   ```

   Add the last two lines from the following at the end of the file and save to make it look the same

   ```YAML
   apiVersion: ansiblenlab.ibm.com/v1beta1
   kind: MyAnsibleLabDemo
   metadata:
     name: example-MyAnsibleLabDemo
   spec:
     # Add fields here
     size: 3
     demo:
       image: niklaushirt/k8sdemo:1.0.0
   ```

2. Save and quit

We have now finished setting up the API and CRDs

Lab 2 - Create the Lab Operator Controller

By default, an Ansible role executes the tasks defined at `roles/tasks/main.yml`. For defining our deployment we will use the k8s module of Ansible.

1) Define the Lab Operator Controller

1. Edit the `ansible-operator-frontend` Controller

```bash
gedit ~/ansible-operator/ansible-operator-frontend/roles/myansiblelabdemo/tasks/main.yml &
```

2. Replace the content with the following:

```yaml
- name: Create the k8sdemo deployment
  k8s:
    definition:
      apiVersion: apps/v1
      kind: Deployment
      metadata:
        name: k8sdemo
        namespace: default
      labels:
        app: k8sdemo
      spec:
        selector:
         matchLabels:
           app: k8sdemo
        replicas: 1
        template:
          metadata:
            labels:
              app: k8sdemo
          spec:
            containers:
            - name: k8sdemo
              image: "{{demo.image}}"
              imagePullPolicy: IfNotPresent
              ports:
              - containerPort: 3000
              env:
                - name: PORT
                  value : "3000"
                - name: APPLICATION_NAME
```

```
                  value: k8sdemo
                - name: BACKEND_URL
                  value: http://k8sdemo-backend-service.default.svc:3000/api

- name: Create the k8sdemo service
  k8s:
    definition:
      apiVersion: v1
      kind: Service
      metadata:
        name: k8sdemo-service
        namespace: default
      spec:
        selector:
          app: k8sdemo
        ports:
          - protocol: TCP
            port: 3000
            targetPort: 3000
            nodePort: 32123
        type: NodePort
```

> This will ensure that the Pod will be created with the Image information defined in the Custom Resource (CR) definition ("{{deployment.image}}"). This picks up the value defined in the CR.

1. Save and Quit

---

2) Build the Lab Operator Controller

1. Now let's build the Operator container

```bash
operator-sdk build localhost:5000/ansible-operator-frontend:ansible

> INFO[0000] Building OCI image localhost:5000/ansible-operator-frontend:ansib
le
> Sending build context to Docker daemon  49.15kB
> Step 1/3 : FROM quay.io/operator-framework/ansible-operator:v0.10.0
>  ---> 168416e214f1
> Step 2/3 : COPY watches.yaml ${HOME}/watches.yaml
>  ---> Using cache
>  ---> 43f81409e05d
> Step 3/3 : COPY roles/ ${HOME}/roles/
>  ---> 0ad354c77a7a
> Successfully built 0ad354c77a7a
> Successfully tagged localhost:5000/ansible-operator-frontend:ansible
> INFO[0001] Operator build complete.
```

Where `localhost:5000/ansible-operator-frontend:ansible` is the name of the Docker image to be created.

2. And push the Operator container to the local registry

   First execute this in order to be able to access the private registry:

```bash
kubectl port-forward --namespace kube-system $(kubectl get po -n kube-system |
 grep kube-registry-v0 | \awk '{print $1;}') 5000:5000  > /dev/null&

> Forwarding from 127.0.0.1:5000 -> 5000
> Forwarding from [::1]:5000 -> 5000
```

> If you don't get a command line prompt, press `enter` several times. If you get an error `Unable to listen on port 5000` you can ignore as you have already started the port-forwarding.

3. And then push the image:

```bash
docker push localhost:5000/ansible-operator-frontend:ansible
```

1) Prepare the Deployment for the Lab Operator Controller

The deployment manifest for the Operator (that was generated automatically) looks like this:

YAML

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ansible-operator-frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      name: ansible-operator-frontend
  template:
    metadata:
      labels:
        name: ansible-operator-frontend
    spec:
      serviceAccountName: ansible-operator-frontend
      containers:
        - name: ansible
          command:
          - /usr/local/bin/ao-logs
          - /tmp/ansible-operator/runner
          - stdout
          # Replace this with the built image name
          image: "{{ REPLACE_IMAGE }}"
          imagePullPolicy: "{{ pull_policy|default('Always') }}"
          volumeMounts:
          - mountPath: /tmp/ansible-operator/runner
            name: runner
            readOnly: true
        - name: operator
          # Replace this with the built image name
          image: "{{ REPLACE_IMAGE }}"
          imagePullPolicy: "{{ pull_policy|default('Always') }}"
          volumeMounts:
          - mountPath: /tmp/ansible-operator/runner
            name: runner
          env:
            - name: WATCH_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: POD_NAME
              valueFrom:
```

```
              fieldRef:
                fieldPath: metadata.name
          - name: OPERATOR_NAME
            value: "ansible-operator-frontend"
      volumes:
        - name: runner
          emptyDir: {}
```

Modify the YAML to use the image that we have pushed to the registry in the Operator deployment

```bash
cp ~/ansible-operator/ansible-operator-frontend/deploy/operator.yaml ~/ansible-op
erator/ansible-operator-frontend/deploy/operator.yaml.bak

sed -i 's|{{ REPLACE_IMAGE }}|niklaushirt/ansible-operator-frontend:ansible|g' ~/
ansible-operator/ansible-operator-frontend/deploy/operator.yaml
sed -i 's|{{ pull_policy.* }}|Always|g' ~/ansible-operator/ansible-operator-front
end/deploy/operator.yaml

more ~/ansible-operator/ansible-operator-frontend/deploy/operator.yaml
```

> To the attentive observer, we're using the prepared image from the docker hub, due to some routing issues for Kubernetes to pull the image internally - thanks minikube.

2) Deploy the Lab Operator

1. Deploy the `ansible-operator-frontend` Custom Resource Definition

   ```bash
   kubectl create -f  ~/ansible-operator/ansible-operator-frontend/deploy/crds/an
   siblenlab_v1beta1_myansiblelabdemo_crd.yaml

   > customresourcedefinition.apiextensions.k8s.io/MyAnsibleLabDemos.ansiblenlab.
   ibm.com created
   ```

2. Create the `ansible-operator-frontend` Service Account

```bash
kubectl create -f  ~/ansible-operator/ansible-operator-frontend/deploy/service_account.yaml
kubectl create -f  ~/ansible-operator/ansible-operator-frontend/deploy/role.yaml
kubectl create -f  ~/ansible-operator/ansible-operator-frontend/deploy/role_binding.yaml

> serviceaccount/ansible-operator-frontend created
> role.rbac.authorization.k8s.io/ansible-operator-frontend created
> rolebinding.rbac.authorization.k8s.io/ansible-operator-frontend created
```

3. Create the `ansible-operator-frontend` Operator

```bash
kubectl create -f  ~/ansible-operator/ansible-operator-frontend/deploy/operator.yaml

> deployment.apps/ansible-operator-frontend created
```

4. Check and wait for the Operator to be running

```bash
kubectl get pods

> NAME                                        READY   STATUS    RESTARTS   AGE
> ansible-operator-frontend-fd78bcf5-zxgws    1/1     Running   0          43m
```

5. Get the log of the Operator Pod (use the Pod name from the previous step)

```
kubectl logs -c operator ansible-operator-frontend-fd78bcf5-zxgws

> {"level":"info","ts":1593618304.1191652,"logger":"cmd","msg":"Go Version: go
1.12.5"}
> {"level":"info","ts":1593618304.1194193,"logger":"cmd","msg":"Go OS/Arch: li
nux/amd64"}
> {"level":"info","ts":1593618304.1194592,"logger":"cmd","msg":"Version of ope
rator-sdk: v0.8.0"}
> {"level":"info","ts":1593618304.1194792,"logger":"cmd","msg":"Watching names
pace.","Namespace":"default"}
> {"level":"info","ts":1593618304.1632428,"logger":"leader","msg":"Trying to b
ecome the leader."}

> {"level":"info","ts":1593618338.328806,"logger":"leader","msg":"Became the l
eader."}
> {"level":"info","ts":1593618338.3861306,"logger":"proxy","msg":"Starting to
serve","Address":"127.0.0.1:8888"}
> {"level":"info","ts":1593618338.386716,"logger":"manager","msg":"Using defau
lt value for workers 1"}
> {"level":"info","ts":1593618338.3867893,"logger":"ansible-controller","msg":
"Watching resource","Options.Group":"ansiblenlab.ibm.com","Options.Version":"v
1beta1","Options.Kind":"MyAnsibleLabDemo"}
> {"level":"info","ts":1593618338.387424,"logger":"kubebuilder.controller","ms
g":"Starting EventSource","controller":"myansiblelabdemo-controller","source":
"kind source: ansiblenlab.ibm.com/v1beta1, Kind=MyAnsibleLabDemo"}
> {"level":"info","ts":1593618338.487868,"logger":"kubebuilder.controller","ms
g":"Starting Controller","controller":"myansiblelabdemo-controller"}
> {"level":"info","ts":1593618338.5881243,"logger":"kubebuilder.controller","m
sg":"Starting workers","controller":"myansiblelabdemo-controller","worker coun
t":1}
```

Bash

This means that the Operator is working and ready.

Lab 2 - Deploy the Custom Resource

1. Deploy the `ansible-operator-frontend` Custom Resource

```Bash
kubectl create -f  ~/ansible-operator/ansible-operator-frontend/deploy/crds/ansiblenlab_v1beta1_myansiblelabdemo_cr.yaml

> MyAnsibleLabDemo.ansiblenlab.ibm.com/example-MyAnsibleLabDemo created
```

   From the resource that we defined earlier:

```YAML
apiVersion: ansiblenlab.ibm.com/v1beta1
kind: MyAnsibleLabDemo
metadata:
  name: example-MyAnsibleLabDemo
spec:
  # Add fields here
  size: 3
  demo:
    image: niklaushirt/k8sdemo:1.0.0
```

2. Check that the CustomResource is running

```Bash
kubectl get pods

> NAME                                     READY   STATUS    RESTARTS   AGE
> ansible-operator-frontend-fd78bcf5-zxgws     2/2     Running   0          3m11s
> k8sdemo-7fc8554dff-2krkz                 1/1     Running   0          45s
```

3. Check the version of the deployed Image

```Bash
kubectl describe deployment k8sdemo | grep Image

> Image:      niklaushirt/k8sdemo:1.0.0
```

4. Once the status reads `Running`, we need to expose that deployment as a service so we can access it through the IP of the worker nodes. The `k8sdemo` application listens on port 3000.

   Run:

```bash
kubectl expose deployment k8sdemo --name k8sdemoansible-service -n default --t
ype="NodePort" --port=3000

> service "k8sdemoansible-service" exposed
```

5. Open the application in your Browser

```bash
minikube service k8sdemoansible-service
```

6. Get the name of the Operator Pod (if you have not done that in the previous task)

```bash
kubectl get pods

> NAME                                      READY   STATUS    RESTARTS   AGE
> ansible-operator-frontend-fd78bcf5-zxgws   2/2     Running   0          87s
> k8sdemo-7c5f4f5895-g7ft7                   1/1     Running   0          2s
```

7. Get the log of the Operator Pod

```bash
kubectl logs -c operator ansible-operator-frontend-fd78bcf5-zxgws
```

We can see that the Operator has picked up on the CustomResource of type `MyAnsibleLabDemo` and handed it to the CustomResource Controller.

```bash
> {"level":"info","ts":1593618338.3867893,"logger":"ansible-controller","msg":
"Watching resource","Options.Group":"ansiblenlab.ibm.com","Options.Version":"v
1beta1","Options.Kind":"MyAnsibleLabDemo"}
> {"level":"info","ts":1593618338.387424,"logger":"kubebuilder.controller","ms
g":"Starting EventSource","controller":"myansiblelabdemo-controller","source":
"kind source: ansiblenlab.ibm.com/v1beta1, Kind=MyAnsibleLabDemo"}
> {"level":"info","ts":1593618338.487868,"logger":"kubebuilder.controller","ms
g":"Starting Controller","controller":"myansiblelabdemo-controller"}
> {"level":"info","ts":1593618338.5881243,"logger":"kubebuilder.controller","m
sg":"Starting workers","controller":"myansiblelabdemo-controller","worker coun
t":1}
> {"level":"info","ts":1593618669.4742577,"logger":"logging_event_handler","ms
g":"[playbook task]","name":"example-myansiblelabdemo","namespace":"default","
gvk":"ansiblenlab.ibm.com/v1beta1, Kind=MyAnsibleLabDemo","event_type":"playbo
ok_on_task_start","job":"6354677430737639809","EventData.Name":"Gathering Fact
s"}
...
```

And that it has executed the Ansible Playbook to create it

Bash

```
> {"level":"info","ts":1593618737.0413835,"logger":"logging_event_handler","msg":"[playbook task]","name":"example-myansiblelabdemo","namespace":"default","gvk":"ansiblenlab.ibm.com/v1beta1, Kind=MyAnsibleLabDemo","event_type":"playbook_on_task_start","job":"520010200565329963","EventData.Name":"myansiblelabdemo : Create the k8sdemo deployment"}
> {"level":"info","ts":1593618737.89282,"logger":"logging_event_handler","msg":"[playbook task]","name":"example-myansiblelabdemo","namespace":"default","gvk":"ansiblenlab.ibm.com/v1beta1, Kind=MyAnsibleLabDemo","event_type":"playbook_on_task_start","job":"520010200565329963","EventData.Name":"myansiblelabdemo : Create the k8sdemo service"}
> {"level":"info","ts":1593618738.7778618,"logger":"runner","msg":"Ansible-runner exited successfully","job":"520010200565329963","name":"example-myansiblelabdemo","namespace":"default"}
```

Lab 2 - Update the Custom Resource

We now proceed to modifying the Custom Resource that we have created in order to demonstrate how the Operator is able to update an existing deployment based on modifications done to the Custom Resource.

1. Modify the Custom Resource

```Bash
sed -i 's|image: niklaushirt/k8sdemo:1.0.0|image: niklaushirt/k8sdemo:1.0.1|g'
 ~/ansible-operator/ansible-operator-frontend/deploy/crds/ansiblenlab_v1beta1_my
```

2. Check that the Image tag (version) has been changed to 1.0.1

```Bash
more ~/ansible-operator/ansible-operator-frontend/deploy/crds/ansiblenlab_v1be
ta1_myansiblelabdemo_cr.yaml
```

   The yaml manifest should look like this

```YAML
apiVersion: ansiblenlab.ibm.com/v1beta1
kind: MyAnsibleLabDemo
metadata:
  name: example-myansiblelabdemo
spec:
  # Add fields here
  size: 3
  demo:
    image: niklaushirt/k8sdemo:1.0.1
```

3. Update the Custom Resource

```Bash
kubectl apply -f ~/ansible-operator/ansible-operator-frontend/deploy/crds/ansi
blenlab_v1beta1_myansiblelabdemo_cr.yaml
```

4. Check that the Operator has picked up the modification in the CustomResource and that the version of the deployed Image has been changed by the Operator to 1.0.1

```Bash
kubectl describe deployment k8sdemo | grep Image

> Image:      niklaushirt/k8sdemo:1.0.1
```

5. And if you reload the browser you should see some nice orange peppers....

Lab 2 - Clean-up the Lab

Delete the `ansible-operator-frontend` Resources

```bash
kubectl delete -f  ~/ansible-operator/ansible-operator-frontend/deploy/crds/ansiblenlab_v1beta1_myansiblelabdemo_crd.yaml
kubectl delete -f  ~/ansible-operator/ansible-operator-frontend/deploy/service_account.yaml
kubectl delete -f  ~/ansible-operator/ansible-operator-frontend/deploy/role.yaml
kubectl delete -f  ~/ansible-operator/ansible-operator-frontend/deploy/role_binding.yaml
kubectl delete -f  ~/ansible-operator/ansible-operator-frontend/deploy/operator.yaml
kubectl delete -f ~/ansible-operator/ansible-operator-frontend/deploy/crds/ansiblenlab_v1beta1_myansiblelabdemo_cr.yaml
kubectl delete service -n default k8sdemoansible-service
```

Congratulations!!! This concludes Lab 2 on Kubernetes Ansible Operators