**Setup: Configuring your lab environment**

**Objective:** Setup your environment to connect to the lab cluster.

**Prerequisites:**

1. A MacOS or Linux bash shell with internet connectivity.
2. Confirmation email with:
   * Your lab user name, e.g. user12
   * The lab cluster server URL, e.g “https://api.foo.bar.com:6443”

**Tasks:**

1. Install the oc CLI executable on your local machine.   
   Instructions are at: <https://docs.openshift.com/container-platform/4.4/cli_reference/openshift_cli/getting-started-cli.html>

After installation, run “which oc” to see if the oc CLI binary is in the current path. If it is not found, you’ll need to add its directory to your PATH variable in step 3 below.

1. Download/unzip or clone this repository to your local machine:

<https://github.com/jodyhuntatx/ocp4-workshop-labs>

1. Cd to ocp4-workshop-labs & edit dap-app.config:
   * set KUBECONFIG to refer to the kubeconfig file in the current directory  
     e.g. “export KUBECONFIG=<absolute-path-to-current-directory>/kubeconfig”  
     DO NOT use a relative path, e.g. “./kubeconfig”
   * set CLUSTER\_DOMAIN to the domain of the OpenShift lab cluster,  
     e.g. if the lab cluster server URL is: <https://api.foo.bar.com:6443>  
     then “export CLUSTER\_DOMAIN=foo.bar.com”
   * If necessary, add the oc CLI binary’s directory to your PATH variable,  
     e.g. “export $PATH:/usr/local/bin”
   * Save the file.
2. Source dap-app.config to set all env vars:

Run: source dap-app.config

1. Set env vars for your DAP admin creds:

Run: export AUTHN\_USERNAME=admin

Run: export AUTHN\_PASSWORD=<value-to-be-provided>

1. Login with your lab user id:

Run: oc login -u <username> --server=$LAB\_CLUSTER

Answer ‘y’ to “Use insecure connections? (y/n):”

1. Wait for further instructions.

**Lab 1: Authenticator running as a Sidecar**

**Objective:** Show how an application can retrieve secrets using the CyberArk Dynamic Access Provider REST API.

**Tasks:**

1. From the ocp4-workshop-labs directory, cd to the 1-sidecar directory.
2. Create deployment yaml:

Run: ./labctl yaml

1. Cat & study contents of dap-config-cm.yaml:

Run: cat dap-config-cm.yaml

1. Create the dap-config config map:

Run: oc apply -f dap-config-cm.yaml

1. Cat & study contents of app-sidecar-policy.yaml:

Run: cat app-sidecar-policy.yaml

1. Load the DAP policy:

Run: ../load\_policy.sh <your-lab-user-name> app-sidecar-policy.yaml

1. Cat & study contents of app-sidecar-manifest.yaml:

Run: cat app-sidecar-manifest.yaml

1. Deploy the app by applying the manifest:

Run: oc apply -f app-sidecar-manifest.yaml

1. Get pod name. Repeat until STATUS is "Running"

Run: oc get pods

1. Exec into pod & run an interactive bash shell:

Run: oc exec -it <pod-name> bash

1. Cat & study contents of mysql\_REST.sh:

Run: cat mysql\_REST.sh

1. Run the script to retrieve DB creds and connect to DB:

Run: ./mysql\_REST.sh

1. Issue “show databases;” command in MySQL DB:

Run: show databases;

1. exit database:

Run: exit

1. exit pod:

Run: exit

1. Delete the deployment:

Run: oc delete -f app-sidecar-manifest.yaml

## **Bonus exercise:**

1. Edit app-sidecar-manifest.yaml
2. Change the value of CONJUR\_AUTHN\_LOGIN to the host identity of another user (just change the user number to a different value)
3. oc apply -f app-sidecar-manifest.yaml
4. oc get pods

Note: Get pod name but DO NOT wait until STATUS is "Running"

1. oc logs <pod-name> -c authenticator -f
2. What is happening? Why?
3. Delete the deployment:

Run: oc delete -f app-sidecar-manifest.yaml

### **End of lab**

**Lab 2: Authenticator running as Init Container**

**Objective:** Show how an application can use vaulted secrets as environment variables using Summon.

**Tasks:**

1. From the ocp4-workshop-labs directory, cd to the 2-initcontainer directory.
2. Create the deployment yaml:

Run: ./labctl yaml

1. Cat & study contents of app-init-policy.yaml - compare to the sidecar policy

Run: cat ./app-init-policy.yaml

Run: sdiff -s ./app-init-policy.yaml ../1-sidecar/app-sidecar-policy.yaml

1. Load the DAP policy:

Run: ../load\_policy.sh <your-lab-user-name> app-init-policy.yaml

1. Cat & study contents of app-init-manifest.yaml - compare to the sidecar manifest:

Run: cat ./app-init-manifest.yaml

Run: sdiff -s ./app-init-manifest.yaml ../1-sidecar/app-sidecar-manifest.yaml

1. Deploy the app by applying the manifest:

Run: oc apply -f app-init-manifest.yaml

1. Get pod name. Repeat every few seconds until STATUS is "Running"

Run: oc get pods

1. Exec into pod & run an interactive bash shell:

Run: oc exec -it <pod-name> bash

1. Cat & study contents of secrets.yml and mysql\_summon.sh:

Run: cat secrets.yml

Run: cat mysql\_summon.sh

1. Run the script to inject DB creds as env vars and connect to DB:

Run: summon ./mysql\_summon.sh

1. Issue “show databases;” command in MySQL DB:

Run: show databases;

1. Exit the mysql prompt:

Run: exit

1. Wait 8 minutes the try running step #10 again.   
     
   Run: summon ./mysql\_summon.sh  
     
   What happens? Why?
2. Exit pod:

Run: exit

1. Delete the deployment:

Run: oc delete -f app-init-manifest.yaml

**End of lab**

**Lab 3: Authenticator running as Secrets Provider for K8s**

**Objective:** Show how an application can used vaulted secrets as native K8s/OpenShift secrets.

**Tasks:**

1. From the ocp4-workshop-labs directory, cd to the 3-k8sprovider directory.
2. Create deployment yaml:

Run: ./labctl yaml

1. Cat & study contents of app-k8ssecrets-policy.yaml:

Run: cat app-k8ssecrets-policy.yaml

1. Load the DAP policy:

Run: ../load\_policy.sh <your-lab-user-name> app-k8ssecrets-policy.yaml

1. Cat & study contents of db-credentials.yaml:

Run: cat db-credentials.yaml

1. Deploy k8s secret db-credential:  
     
   Run: oc apply -f ./db-credentials.yaml
2. Edit db-credentials, notice there are no base64 encoded values for username and password:

Run: oc edit secret db-credentials  
  
Exit without changing anything - ‘:q!’

1. Cat & study contents of provider-k8ssecrets-manifest.yaml:

Run: cat provider-k8ssecrets-manifest.yaml

1. Apply the secrets-access role binding by applying the manifest:

Run: oc apply -f provider-k8ssecrets-manifest.yaml

1. Cat & study contents of app-k8ssecrets-manifest.yaml:

Run: cat app-k8ssecrets-manifest.yaml

1. Deploy the app by applying the manifest:

Run: oc apply -f app-k8ssecrets-manifest.yaml

1. Get pod name. DO NOT WAIT until STATUS is "Running"

Run: oc get pods

1. Follow the secrets provider log. Watch it authenticate, retrieve secrets and update the db-credentials secret:

Run: oc logs <pod-name> -c secrets-provider -f  
  
Watch the log until it exits with the message:  
DAP/Conjur Secrets updated in Kubernetes successfully

1. Edit db-credentials, notice base64 encoded values for username and password:

Run: oc edit secret db-credentials  
  
Exit without changing anything - ‘:q!’

1. Exec into pod & run an interactive bash shell:

Run: oc exec -it <pod-name> bash

1. Cat & study contents of mysql\_provider.sh:

Run: cat mysql\_provider.sh

1. Examine the secrets mounted as environment variables and volumes:  
   Run: env | grep ^DB\_  
   Run: cat /etc/secret-volume/password; echo
2. Run the script to retrieve DB creds and connect to DB:

Run: ./mysql\_provider.sh

1. Issue “show databases;” command in MySQL DB:

Run: show databases;

1. Exit the mysql prompt:

Run: exit

1. Exit the pod:

Run: exit

1. Delete the deployment & secret:

Run: oc delete -f app-k8ssecrets-manifest.yaml

Run: oc delete -f db-credentials.yaml

## **Bonus exercise:**

1. Edit db-credentials.yaml - change the name of the secret retrieved for password in the conjur-map  
   Run: vi db-credentials.yaml, save with ‘:wq’
2. Deploy the modified db-credentials secret:  
   Run: oc apply -f db-credentials.yaml
3. Redeploy the application:  
   Run: oc apply -f app-k8ssecrets-manifest.yaml
4. Get the pod name, do not wait for its Status to be “Running”:  
   Run: oc get pods
5. Watch the secrets-provider container log:  
   Run: oc logs <pod-name> -c secrets-provider -f
6. What is happening in the pod log? Why?
7. Delete the application pod and secret:  
   Run: oc delete -f app-k8ssecrets-manifest.yaml  
   Run: oc delete -f db-credentials.yaml

### **End of lab**

**Lab 4: Authenticator running as Secretless Broker**

**Objective:** Show how an application can connect to a database without access to credentials.

**Tasks:**

1. From the ocp4-workshop-labs directory, cd to the 4-secretless directory.
2. Create deployment yaml:

Run: ./labctl yaml

1. Cat & study contents of app-secretless-policy.yaml:

Run: cat app-secretless-policy.yaml

1. Load the DAP policy:

Run: ../load\_policy.sh <your-lab-user-name> app-secretless-policy.yaml

1. Cat & study contents of secretless.yaml:

Run: cat secretless.yaml

1. Create secretless config map:

Run: oc create cm secretless-config --from-file=secretless.yaml

1. Cat & study contents of app-secretless-manifest.yaml:

Run: cat app-secretless-manifest.yaml

1. Deploy the app by applying the manifest:

Run: oc apply -f app-secretless-manifest.yaml

1. Get pod name. Wait until STATUS is "Running"

Run: oc get pods

1. Get the secretless broker log. What is it doing?

Run: oc logs <pod-name> -c secretless-broker

1. Exec into pod & run an interactive bash shell:

Run: oc exec -it <pod-name> bash

1. Cat & study contents of mysql\_secretless.sh. How will it connect to the DB w/o creds?:

Run: cat mysql\_secretless.sh

1. Run the script to connect to DB:

Run: ./mysql\_secretless.sh

1. Issue “show databases;” command in MySQL DB:

Run: show databases;

1. Exit the mysql prompt:

Run: exit

1. Exit the pod:

Run: exit

1. Delete the deployment & config map:

Run: oc delete -f app-secretless-manifest.yaml

Run: oc delete cm secretless-config

### **End of lab**

End of Workshop!