Modernize DevOps with CyberArk Secrets Management and Red Hat OpenShift

Lab Guide

Introduction:

You can improve and simplify the security of Red Hat® OpenShift® containerized environments with CyberArk Secrets Management solutions, and out-of-the-box certified integrations. A centralized approach to Secrets Management can help accelerate deployments and strengthen containerized application security, without impacting developer velocity.

This Lab will cover:

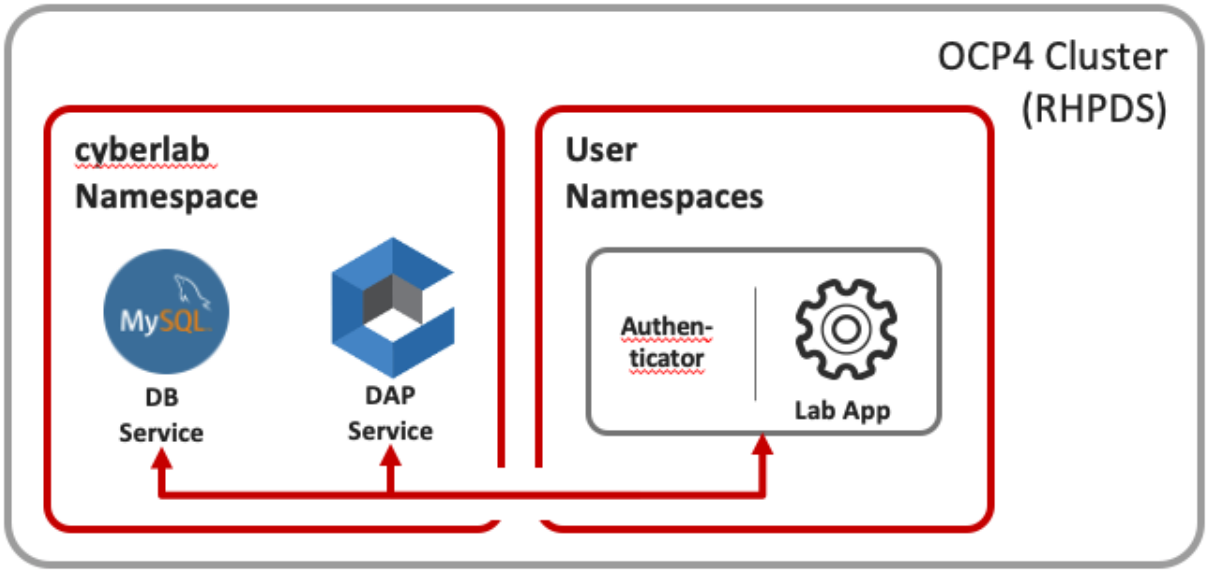
- How an application can retrieve secrets using the CyberArk Dynamic Access Provider REST API
- How an application can use vaulted secrets injected as environment variables using Summon.
- How an application can use vaulted secrets as native K8s/OpenShift secrets.
- How an application can connect to a database without access to credentials.

Table of contents:

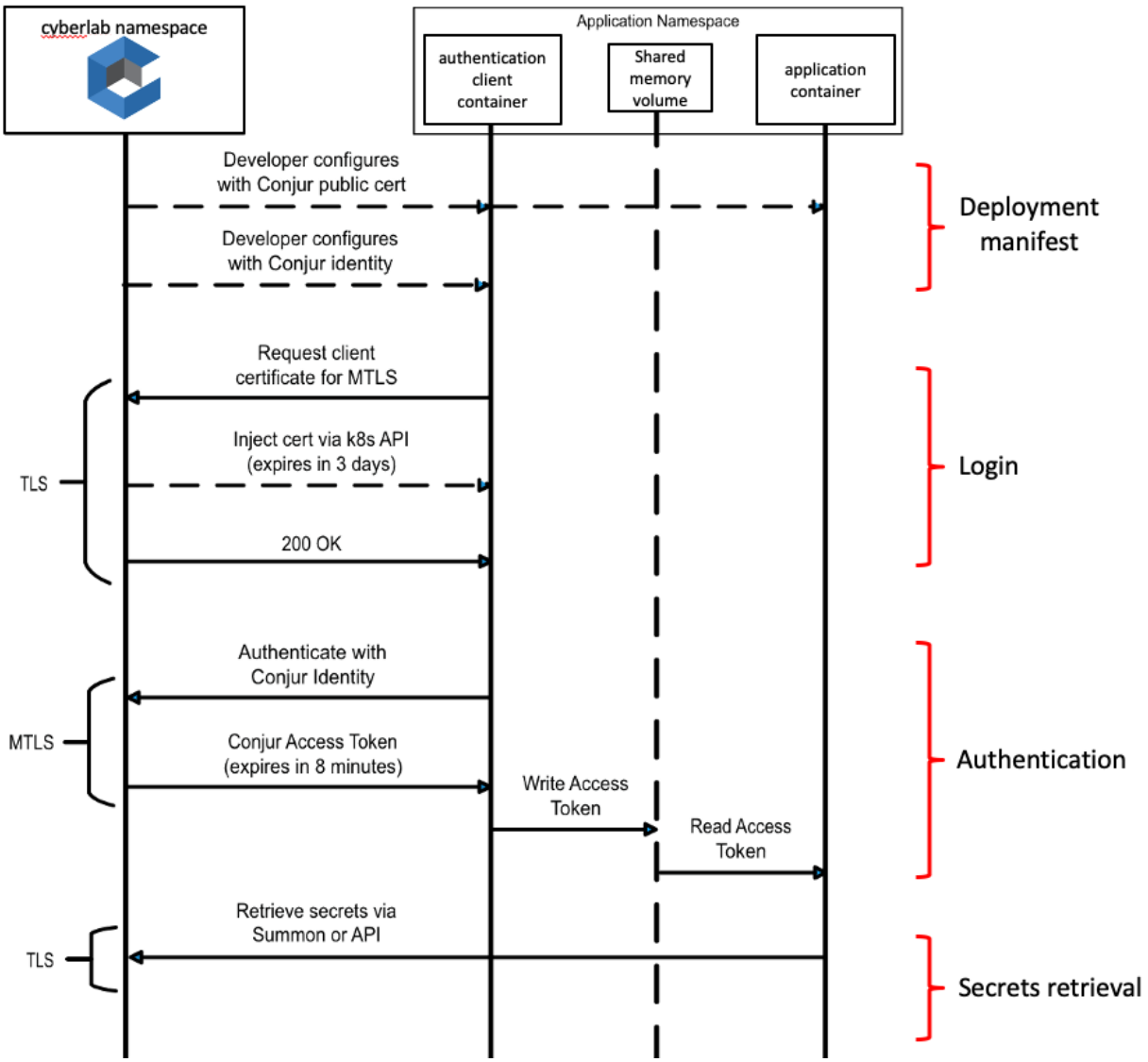
Lab Architecture

From your computer, you will deploy and run applications in a Red Hat-hosted OpenShift cluster. You will connect to the cluster using the oc CLI. You will also load CyberArk Dynamic Access Provider (DAP) policies that define identities for your applications that permit them to authenticate to the DAP service node running in the cyberlab namespace, and retrieve a MySQL database username and password. The application will use those credentials to connect to the database.

DAP authentication in K8s/OpenShift is performed by a separate container running in the same pod with the application container. Authentication is based on a mutual TLS (MTLS) workflow similar to that used by Istio's service mesh and is based on x509 certificates that comply with the SPIFFE x.509 SVID format. This authentication strategy is extremely secure, while eliminating the need for API keys or other stored credentials that must be protected from being stolen, spoofed or otherwise compromised.



Secrets Retrieval Workflow



Applications must first authenticate before they can retrieve secrets. In effect, each Kubernetes (K8s) cluster is considered a trust domain within which DAP Servers act as a certificate authority (CA) that issues x509 certificates for use as application authentication credentials. The certificates establish trust between application pods running in the cluster and the DAP Server. The DAP Server must first verify an application pod is using a known identity that was pre-defined by DAP policy to run in a given namespace, using an optionally specified service account, and permitted to call the DAP Server's authentication endpoint for the cluster. This verification process is called Login.

Login

With DAP in K8s, application containers do not perform authentication for themselves, but instead delegate that to a CyberArk-provided authentication client container running in the pod. To obtain its x509 authentication credentials, the authentication client submits a Certificate Signing Request (CSR) to the DAP Server's authentication endpoint for the cluster. The metadata in the CSR includes the DAP application login identity that the pod would like to use to authenticate, plus metadata attributes for the pod (namespace name, pod name, service account, IP address). The DAP Server first verifies that the login identity in the CSR was pre-defined via DAP RBAC policy and is permitted to call the DAP Server's authentication endpoint. If so, the DAP Server then calls the Kubernetes API to verify the pod's metadata attributes. This will confirm there is indeed a pod with the specified name, running in the specified namespace, at the specified IP address, and using the specified service account.

If this verification succeeds, the DAP Server issues a self-signed client certificate and private key, writing them directly into the authenticator container's file system using the Kubernetes API. By using the API to perform this credential injection out-of-band (instead of returning it in the request response), the process ensures that the client certificate is delivered to the pod that matches the metadata in the CSR.

The certificate expires and is renewed automatically on a regular basis to reduce the chances of a malicious third party being able to use a compromised certificate to assume the application pod's identity.

Authentication

After receiving a client certificate and key, the authentication client uses them to establish a mutual TLS connection with the DAP server and requests a short-lived access token. The access token is persisted to a file (typically in a shared memory volume) where the token is used to retrieve secrets from the DAP server. The token is short-lived to reduce the possibility that it can be used by a malicious third party if it is somehow compromised.

Secrets retrieval

If the authentication client writes the token to a shared memory volume, the application container can mount the memory volume and use the access token to retrieve secrets using one of the supported APIs (REST, Go, Java, .Net, Ruby). Alternatively, the open source project Summon can use the token to retrieve secrets, run the application as a sub-process, and inject secret values as environment variables. Variants of the authentication client can also retrieve secrets and make them available as Kubernetes secrets, or establish proxied connections to target systems such as databases and web services.

Each of the above secrets retrieval scenarios is demonstrated in the labs.

Role-Based Access Control

Openshift supports strong Role-Based Access Control (RBAC) for its resources via Roles & Role bindings, defined with YAML manifests at the cluster and namespace levels. CyberArk DAP also supports a highly flexible RBAC system managed via YAML policies. These policies define application identities and their permissions to access credentials. A goal of the labs is to show how DAP's policies can provide "sandboxes with high walls around them" for OpenShift application development teams. Users are given the latitude to define their own application identities and credential access permissions, but only for their OpenShift project/namespace.

The DAP policy model for the workshop makes each lab user the administrator of their own policy space within which they create application identities and role permissions to access database credentials. A bonus exercise for Lab 1 invites the user to test the effectiveness of these access controls, by trying to deploy their application using another user's application identity.

Setup: Accessing your lab environment

Objective:

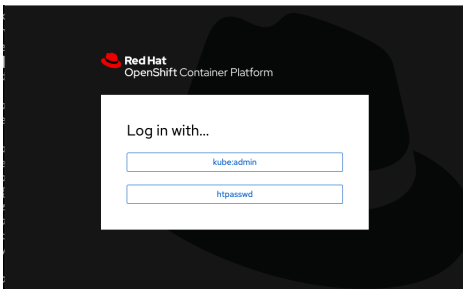
Login to the OpenShift cluster and open a terminal in your labadmin pod.

Prerequisites:

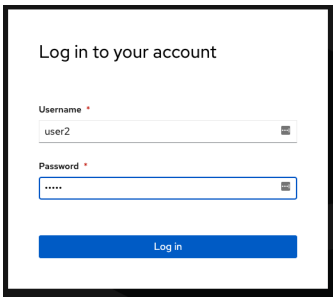
- A web browser with internet connectivity.

Tasks:

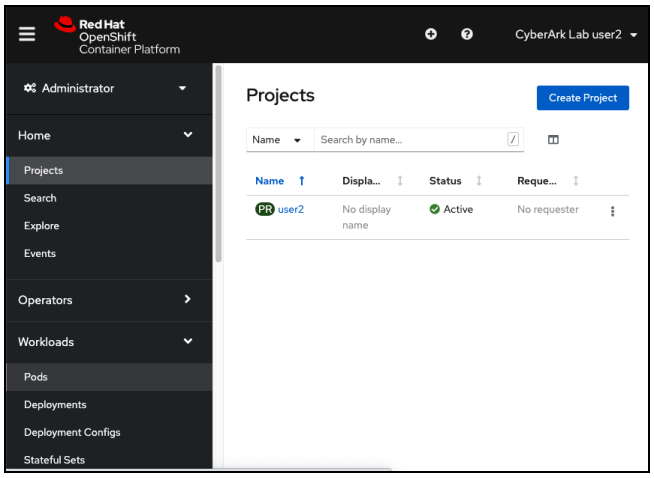
1. Connect to the OpenShift cluster URL. If given a choice of login options, click “htpasswd”.



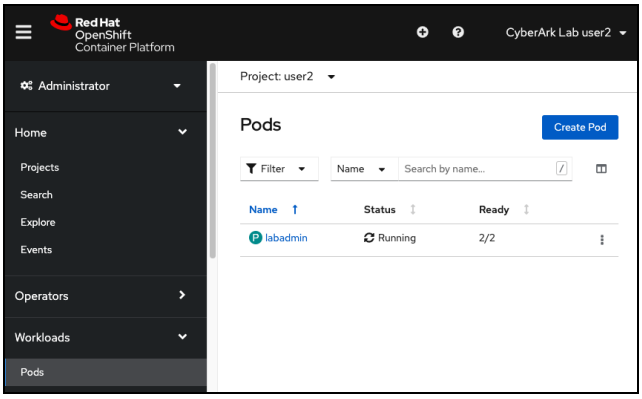
2. Login with the username and password you were assigned.



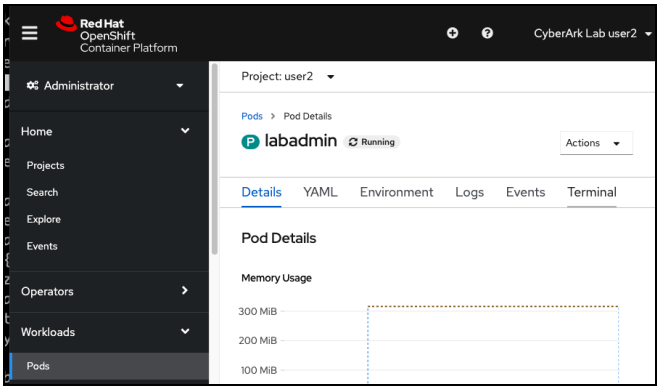
3. Make sure you are in “Administrator” mode (upper left), click on the Project with your username, then under Workloads click Pods.



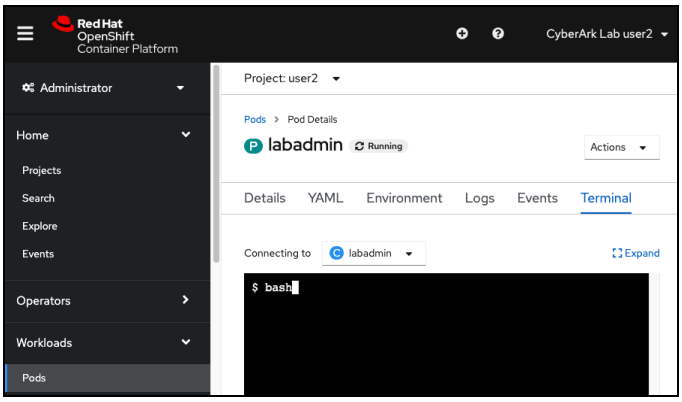
4. Click on “labadmin” in the pods list.



5. Click on Terminal to open a terminal in the labadmin pod.



6. Type “bash” at the terminal prompt to run a bash shell in the terminal.



7. Click “expand” to make the terminal window bigger.

8. You’re now ready to start Lab 1.

End of Lab setup

Lab 1: Retrieve secrets using the DAP REST API

Objective:

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

Instructor workflow:

- Short walkthrough of lab
- Allow at least 15 minutes for students to complete lab
- Lab summary/Q&A

Tasks:

1. From the ocp4-workshop-labs directory, cd to the 1-sidecar directory.
2. Create deployment yaml:
   Run: ./labctl yaml
3. Cat & study contents of dap-config-cm.yaml:
   Run: cat dap-config-cm.yaml
4. Create the dap-config config map:
   Run: oc apply -f dap-config-cm.yaml
5. Cat & study contents of app-sidecar-policy.yaml:
   Run: cat app-sidecar-policy.yaml
6. Load the DAP policy:
   Run: ../load_policy.sh <your-lab-user-name> app-sidecar-policy.yaml
7. Cat & study contents of app-sidecar-manifest.yaml:
   Run: cat app-sidecar-manifest.yaml
8. Deploy the app by applying the manifest:
   Run: oc apply -f app-sidecar-manifest.yaml
9. Get pod name. Repeat until STATUS is "Running"

Run: oc get pods

10. Exec into pod & run an interactive bash shell:
    Run: oc exec -it <pod-name> bash
11. Cat & study contents of mysql_REST.sh:
    Run: cat mysql_REST.sh
12. Run the script to retrieve DB creds and connect to DB:
    Run: ./mysql_REST.sh
13.
14. Issue"show databases;"command in MySQL DB:
    Run: show databases;
15. Exit database:
    Run: exit
16. Exit pod:
    Run: exit
17. Delete the deployment:
    Run: oc delete -f app-sidecar-manifest.yaml

Bonus exercise:

1. Edit app-sidecar-manifest.yaml and change the value of CONJUR_AUTHN_LOGIN to the host identity of another user (just change the user number to a different value)
2. Apply your changes:
    Run: oc apply -f app-sidecar-manifest.yaml
3. Get pod name but DO NOT wait until STATUS is "Running"
    Run: oc get pods
4. Follow the authenticator container log:
    Run: oc logs <pod-name> -c authenticator -f
5. What is happening? Why?
6. Delete the deployment:
    Run: oc delete -f app-sidecar-manifest.yaml

End of Lab 1

Lab 2: Inject secrets as environment variables with Summon

Objective:

Show how an application can use vaulted secrets as environment variables injected using Summon.

Instructor workflow:

- Short walkthrough of lab
- Allow at least 15 minutes for students to complete lab
- Lab summary/Q&A

Tasks:

1. From the ocp4-workshop-labs directory, cd to the 2-initcontainer directory.
2. Create the deployment yaml:
   Run: ./labctl yaml
3. 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
4. Load the DAP policy:
   Run: ../load_policy.sh <your-lab-user-name> app-init-policy.yaml
5. 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
6. Deploy the app by applying the manifest:
   Run: oc apply -f app-init-manifest.yaml
7. Get pod name. Repeat every few seconds until STATUS is "Running"
   Run: oc get pods

8. Exec into pod & run an interactive bash shell:
   Run: oc exec -it <pod-name> bash

9. Cat & study contents of secrets.yml and mysql_summon.sh:
   Run: cat secrets.yml
   Run: cat mysql_summon.sh

10. Run the script to inject DB creds as env vars and connect to DB:
    Run: summon ./mysql_summon.sh

11. Issue"show databases;"command in MySQL DB:
    Run: show databases;

12. Exit the mysql prompt:
    Run: exit

13. Wait 8 minutes the try running step #10 again.
    Run: summon ./mysql_summon.sh

14. What happens? Why?

15. Exit pod:
    Run: exit

16. Delete the deployment:
    Run: oc delete -f app-init-manifest.yaml

End of Lab 2

Lab 3: Provide secrets as native Kubernetes secrets

Objective:

Show how an application can use vaulted secrets as native K8s/OpenShift secrets.

Instructor workflow:

- Short walkthrough of lab
- Allow at least 15 minutes for students to complete lab
- Lab summary/Q&A

Tasks:

1. From the ocp4-workshop-labs directory, cd to the 3-k8sprovider directory.
2. Create deployment yaml:
   Run: ./labctl yaml
3. Cat & study contents of app-k8ssecrets-policy.yaml:
   Run: cat app-k8ssecrets-policy.yaml
4. Load the DAP policy:
   Run: ../load_policy.sh <your-lab-user-name> app-k8ssecrets-policy.yaml
5. Cat & study contents of db-credentials.yaml:
   Run: cat db-credentials.yaml
6. Deploy k8s secret db-credential:
   Run: oc apply -f ./db-credentials.yaml
7. Edit db-credentials, notice there are no base64 encoded values for username and password:
   Run: oc edit secret db-credentials
   Exit without changing anything – vi command is :q! (colon q exclamation point)
8. Cat & study contents of provider-k8ssecrets-manifest.yaml:
   Run: cat provider-k8ssecrets-manifest.yaml

9. Apply the secrets-access role binding by applying the manifest:
   Run: oc apply -f provider-k8ssecrets-manifest.yaml

10. Cat & study contents of app-k8ssecrets-manifest.yaml:
    Run: cat app-k8ssecrets-manifest.yaml

11. Deploy the app by applying the manifest:
    Run: oc apply -f app-k8ssecrets-manifest.yaml

12. Get pod name. DO NOT WAIT until STATUS is "Running"
    Run: oc get pods

13. 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

14. Edit db-credentials, notice base64 encoded values for username and password:
    Run: oc edit secret db-credentials
    Exit without changing anything -':q!'

15. Exec into pod & run an interactive bash shell:
    Run: oc exec -it <pod-name> bash

16. Cat & study contents of mysql_provider.sh:
    Run: cat mysql_provider.sh

17. Examine the secrets mounted as environment variables and volumes:
    Run: env | grep ^DB_
    Run: cat /etc/secret-volume/password; echo

18. Run the script to retrieve DB creds and connect to DB:
    Run: ./mysql_provider.sh

19. Issue"show databases;"command in MySQL DB:
    Run: show databases;

20. Exit the mysql prompt:
    Run: exit

21. Exit the pod:
    Run: exit

22. 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 3

Lab 4: Connect without secrets through the Secretless Broker

Objective:

Show how an application can connect to a database without access to credentials using the Secretless Broker.

Instructor workflow:

- Short walkthrough of lab
- Allow at least 15 minutes for students to complete lab
- Lab summary/Q&A

Tasks:

1. From the ocp4-workshop-labs directory, cd to the 4-secretless directory.
2. Create deployment yaml:
   Run: ./labctl yaml
3. Cat & study contents of app-secretless-policy.yaml:
   Run: cat app-secretless-policy.yaml
4. Load the DAP policy:
   Run: ../load_policy.sh <your-lab-user-name> app-secretless-policy.yaml
5. Cat & study contents of secretless.yaml:
   Run: cat secretless.yaml
6. Create secretless config map:
   Run: oc create cm secretless-config --from-file=secretless.yaml
7. Cat & study contents of app-secretless-manifest.yaml:
   Run: cat app-secretless-manifest.yaml
8. Deploy the app by applying the manifest:
   Run: oc apply -f app-secretless-manifest.yaml

9. Get pod name. Wait until STATUS is "Running"
   Run: oc get pods
10. Get the secretless broker log. What is it doing?
    Run: oc logs <pod-name> -c secretless-broker
11. Exec into pod & run an interactive bash shell:
    Run: oc exec -it <pod-name> bash
12. Cat & study contents of mysql_secretless.sh. How will it connect to the DB w/o creds?:
    Run: cat mysql_secretless.sh
13. Run the script to connect to DB:
    Run: ./mysql_secretless.sh
14. Issue “show databases;” command in MySQL DB:
    a. Run: show databases;
15. Exit the mysql prompt:
    a. Run: exit
16. Exit the pod:
    a. Run: exit
17. Delete the deployment & config map:
    a. Run: oc delete -f app-secretless-manifest.yaml
    b. Run: oc delete cm secretless-config

End of Lab 4

End of Workshop!

Links/contacts for more information

Whitepaper, Blogs, Resources

- www.cyberark.com/RedHat

Partner EcoSystems

- CyberArk Marketplace page for Red Hat
  https://cyberark-customers.force.com/mplace/s/#--red+hat
- RedHat Ecosystem page for CyberArk
  https://catalog.redhat.com/software/containers/search?q=cyberark

Open Source – Secrets Management - www.conjur.org

- Secretless -- https://www.conjur.org/api/secretless-broker/
- Summon -- https://cyberark.github.io/summon
- Blog – www.conjur.org/blog

CyberArk documentation -- https://docs.cyberark.com

- See lower left at bottom for Dynamic Access Provider