K10 Workflows – Backup, Restore, and Disaster Recovery

In this example, we will walk through how to use Kasten K10 to backup and restore a sample WordPress application. This application namespace has one MySQL pod, to provide data persistence, and a WordPress pod to provide an application web frontend.

The MySQL database pod is configured to use the ocs-storagecluster-ceph-rbd storage class.

Objectives

- Deploy the sample application using Nutanix Volumes as Default Storage Class
- Backup and restore sample application using Kasten K10
- Disaster recovery scenario of sample application using Kasten K10 to a different cluster
- Disaster recovery scenario of recovering Kasten K10 Control Plane

Deploy the sample Wordpress application stack via Kustomize

This application deployment is based on https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/, but tailored for Openshift and Nutanix

- Create Storage Classe(s) [optional]
- Create MySQL resource configs
- Create WordPress resource configs
- Apply the kustomization directory by kubectl apply -k ./

Create Storage Classes [optional]

- Create Nutanix Volumes - Default Storage Class
- Create Nutanix Files Dynamic Storage Class

Create MySQL Deployment Resource Configs

The following manifest describes a single-instance MySQL Deployment. The MySQL container mounts the PersistentVolume at /var/lib/mysql. The MYSQL_ROOT_PASSWORD environment variable sets the database password from the Secret.

```
## Create local kustomize dir if it doesn't already exist and cd into it
mkdir -p ./kustomize/wordpress && cd ./kustomize/wordpress

## Create a kustomization.yaml with a Secret generator and target
resources (to be defined below)
cat <<EOF >./kustomization.yaml
secretGenerator:
- name: mysql-pass
  literals:
  - password=nutanix/4u
```

```
resources:
- mysql-deployment.yaml
- wordpress-deployment.yaml
EOF

## Create wordpress mysql backend deployment manifest
cat <<EOF >./mysql-deployment.yaml
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  namespace: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
  clusterIP: None
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: wordpress-sa
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
      serviceAccountName: wordpress-sa
EOF
```

Create WordPress resource configs

```
## create wordpress deployment manifest
cat <<EOF >./wordpress-deployment.yaml
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
  selector:
    app: wordpress
    tier: frontend
  type: ClusterIP
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
    spec:
      containers:
      - image: wordpress:4.8-apache
        name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql
        - name: WORDPRESS_DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
        ports:
        - containerPort: 80
          name: wordpress
        volumeMounts:
        - name: wordpress-persistent-storage
          mountPath: /var/www/html
      volumes:
      - name: wordpress-persistent-storage
        persistentVolumeClaim:
          claimName: wp-pv-claim
      serviceAccountName: wordpress-sa
EOF
```

Apply the kustomization directory

```
## create namespace
kubectl create ns wordpress --dry-run=client -o yaml | kubectl apply -f -

## openshift oc reconfigure project and set cluster role to wordpress
service account
oc project wordpress
oc adm policy add-cluster-role-to-user cluster-admin -z wordpress-sa

## apply files via kustomize
$ kubectl apply -k ./
serviceaccount/wordpress-sa created
secret/mysql-pass-dgc582mg4t created
service/wordpress-mysql created
service/wordpress created
persistentvolumeclaim/mysql-pv-claim created
persistentvolumeclaim/wp-pv-claim created
deployment.apps/wordpress created
deployment.apps/wordpress-mysql created
```

Access and Validate the Application

- Expose the Wordpress Application via Openshift Route
- Access the Wordpress Application via Preferred Browser
- Complete the Installation Wizard setup and Add a Few Sample Posts

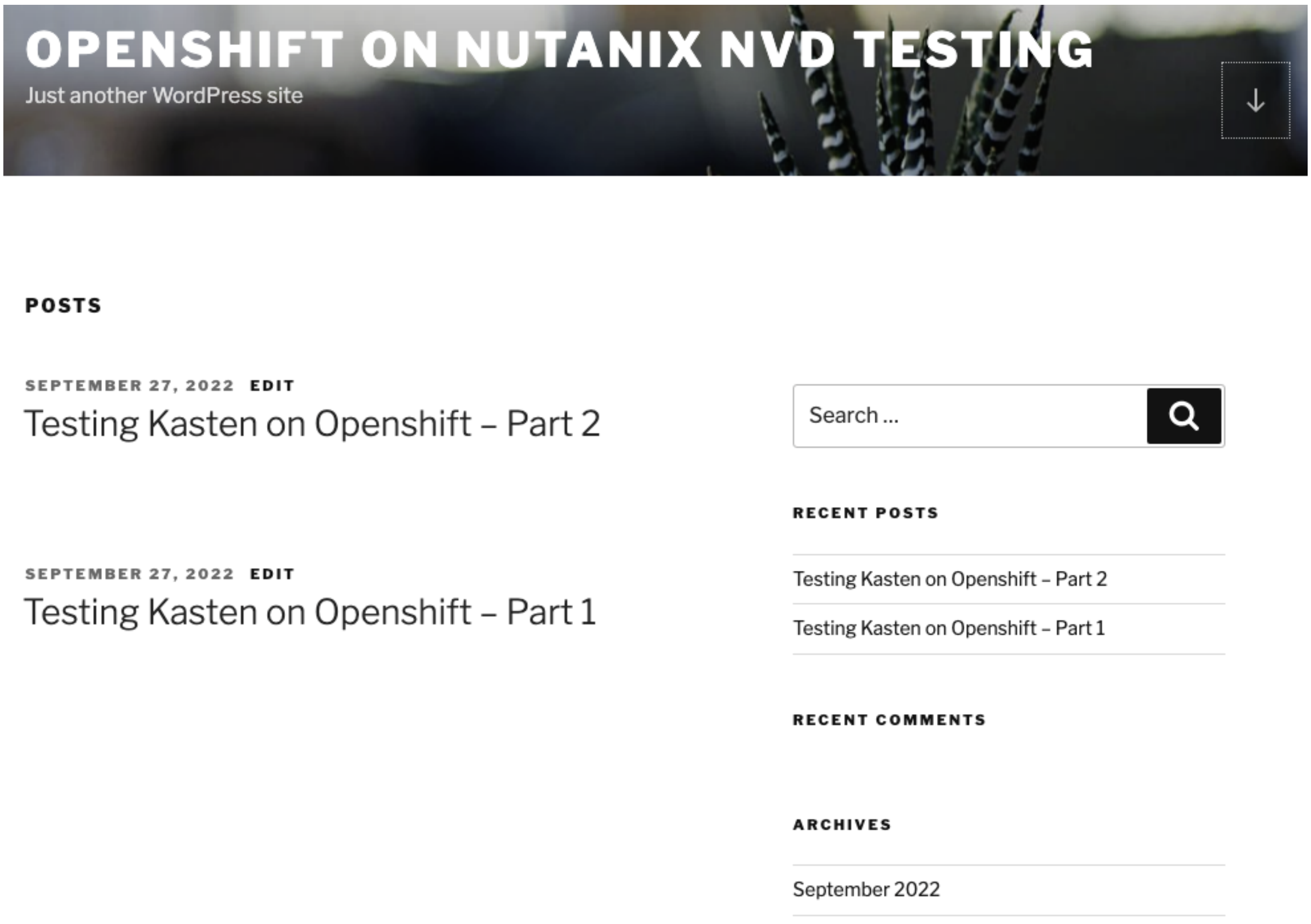
```
## 1. Expose the Wordpress Application via Openshift Route

export INGRESS_NAME=wordpress.apps.ocp-az1.dachlab.net

cat <<EOF | kubectl apply -n wordpress -f -
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    app: wordpress
  name: wordpress
  namespace: wordpress
spec:
  host: $( echo $INGRESS_NAME )
  port:
    targetPort: 80
  tls:
    insecureEdgeTerminationPolicy: Redirect
    termination: edge
  to:
    kind: Service
    name: wordpress
    weight: 100
  wildcardPolicy: None
EOF
```

```
## 2. Get route information and Access the Wordpress Application via
Preferred Browser
❯ kubectl get route wordpress -n wordpress
NAME        HOST/PORT                             PATH   SERVICES    PORT
TERMINATION     WILDCARD
wordpress   wordpress.apps.ocp-az1.dachlab.net          wordpress   80
edge/Redirect   None

## 3. Complete the Installation Wizard setup and Add a Few Sample Posts
1. Select language
2. Input name for WordPress site
3. Configure admin credentials
4. Login WordPress and Add at few arbitrary sample posts
```


OPENSHIFT ON NUTANIX NVD TESTING
Just another WordPress site
POSTS
SEPTEMBER 27, 2022 EDIT
Testing Kasten on Openshift – Part 2
SEPTEMBER 27, 2022 EDIT
Testing Kasten on Openshift – Part 1
Search …
RECENT POSTS
Testing Kasten on Openshift – Part 2
Testing Kasten on Openshift – Part 1
RECENT COMMENTS
ARCHIVES
September 2022

Backup and restore workflow using Kasten K10

In this example, we will walk through how to use Kasten K10 to backup and restore a sample WordPress application. This application namespace has one MySQL pod, to provide data persistence, and a WordPress pod to provide an application web frontend.

The MySQL database pod is configured to use the ocs-storagecluster-ceph-rbd storage class.

Disaster recovery workflow using Kasten K10 to a different cluster