WRITING YOUR FIRST ANSIBLE OPERATOR FOR OPENSHEET
Operators are application aware Kubernetes objects.

Active throughout the application’s lifecycle, they manage instantiation, ongoing state, and destruction.
FROM VISION TO PROBLEM
problem: turnkey management of stateless application

solution: kubernetes (we just saw this)
S2I, Helm
build image from source
intra-cluster traffic management
application runtime configuration
external traffic
problem:

I’m a vendor or I create stateful apps, kubernetes doesn’t know anything about me
etcd is a distributed key value store that provides a reliable way to store data across a cluster of machines.
problem: I’m a vendor or I create stateful apps, kubernetes doesn’t know anything about me

solution: create custom resource definitions (CRD)
---

```yaml
apiVersion: v1
kind: Service
metadata:
  name: simpleapp
spec:
  ports:
    - name: 8080-tcp
      port: 8080
      protocol: TCP
      targetPort: 8080
  selector:
    deploymentconfig: simpleapp
    sessionAffinity: None
  type: ClusterIP
```

---

defining a service resource

service resources are a built in object type.
defining an `EtcdCluster` resource

Our custom resource looks pretty similar.

```yaml
---
apiVersion: etcd.database.coreos.com/v1beta2
group: EtcdCluster
metadata:
  name: example-etcd-cluster
spec:
  size: 3
  version: "3.2.13"
```
problem: golang isn’t going to fly

solution: skip go, succeed with helm charts or ansible
EVERY PROBLEM BRINGS A SOLUTION
Compare desired state with actual state
Reconcile process converges to desired state
Native K8s objects like...
Pods
Services
Routes
etc.
* operator

watch reconciliate

action

AS
Ansible operator

Ansible playbook or role
This is the only component you need to worry about!
Phase I
Manage native K8s objects
# tasks/main.yml
- name: Create k8s resources
  k8s:
    state: present
    definition: 'select("template", item) | from_yamls_all | list'
  with_items:
  - rbac.yaml
  - pod.yaml
  vars:
    ns: example-app

# templates/rbac.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: example-app-cluster
  namespace: '{{ ns }}'
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: example-app-cluster
  namespace: '{{ ns }}'
rules:
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: [ "get", "list", "watch", "create", "update", "delete" ]

# Allow the pods in this namespace to work with configmaps
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: example-app-cluster
  namespace: '{{ ns }}'
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: example-app-cluster
subjects:
  - kind: ServiceAccount
    name: example-app-cluster
    namespace: '{{ ns }}'
# tasks/main.yml

- name: Create k8s resources
  k8s:
    state: present
    definition: '{{ lookup("template", item) | from_yaml_all | list }}'
  with_items:
  - rbac.yaml
  - pod.yaml
  vars:
    ns: example-app
# templates/pod.yaml

---

apiVersion: v1
kind: Pod
metadata:
  name: example-app
  namespace: '{{ ns }}'
spec:
  serviceAccountName: example-app-cluster
  containers:
  - name: example-app
    image: busybox:latest
    command: ['sleep']
    args: ['3600']
Phase II
Manage application objects

Ansible operator

watch
reconcile

ansible-runner

ETCD
pod

01001
etcd data

01001
etcd data

application layer

kubernetes layer

@KeithResar
A GIFT OF THE DEMO TO YOU
Demo Operator for data service **SimpleDB**, that manages instantiation and version upgrades.
Create service account, role, and role binding. Our operator uses these to monitor events and reconcile desired and actual states.
---
apiVersion: v1
class: ServiceAccount
metadata:  
  name: simpledb
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:  
  name: simpledb
rules:...
---

type: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:  
  name: simpledb
subjects:  
  - kind: ServiceAccount
    name: simpledb
roleRef:  
  kind: Role
  name: simpledb
  apiGroup: rbac.authorization.k8s.io
Define the custom resource SimpleDB. This extends what Kubernetes accepts, but doesn’t actually change any behavior.
apiVersion: apiextensions.k8s.io/v1beta1
crdd: CustomResourceDefinition
metadata:
  name: simplesbs.example.com
spec:
  group: example.com
  names:
    kind: SimpleDB
    listKind: SimpleDBList
    plural: simplesbs
    singular: simplesdb
  scope: Namespaced
  version: v1alpha1
Define and deploy the Ansible Operator container which executes an ansible-runner process.
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: simpledb
spec:
template:
spec:
  serviceAccountName: simpledb
  containers:
  - name: simpledb
    image: hk1232/operator-simpledb-runner:0.1
    env:
      - name: WATCH_NAMESPACE
        valueFrom:
          fieldRef:
            fieldPath: metadata.namespace
      - name: OPERATOR_NAME
        value: "simpledb"
# Dockerfile

FROM quay.io/water-hole/ansible-operator

USER root

RUN yum -y install MySQL-python && \
    pip --no-cache-dir install dnspython

COPY roles/ ${HOME}/roles/ \
COPY playbook.yaml ${HOME}/playbook.yaml \
COPY watches.yaml ${HOME}/watches.yaml
Ansible operator

- watch
- reconcile

ansible-runner
FROM quay.io/water-hole/ansible-operator
USER root

RUN yum -y install MySQL-python && \
ip --no-cache-dir install dnspython

COPY roles/ ${HOME}/roles/
COPY playbook.yaml ${HOME}/playbook.yaml
COPY watches.yaml ${HOME}/watches.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: simpledb
spec:
template:
spec:
serviceAccountName: simpledb
containers:
  - name: simpledb
    image: hk1232/operator-simpledb-runner:0.1
    env:
      - name: WATCH_NAMESPACE
        valueFrom:
          fieldRef:
            fieldPath: metadata.namespace
      - name: OPERATOR_NAME
        value: "simpledb"
Ansible operator

DS

watch

reconcile

ansible-runner

AS
# watches.yml

```yaml
- version: v1alpha1
  group: example.com
  kind: SimpleDB
  playbook: /opt/ansible/playbook.yaml
```
# playbook.yml
---
- hosts: localhost
gather_facts: no
tasks:
  - import_role:
      name: "SimpleDB"
RBAC → CRD → DC → CR

# roles/SimpleDB/tasks/main.yml
# roles/SimpleDB/tasks/main.yml

---

# ... (skip setting some variables)
# roles/SimpleDB/tasks/main.yml

---

# ... (skip setting some variables)

# If no service defined then run our install playbook
# This is idempotent so we could run it regardless
- include_tasks: mariadb_install.yml
  when: mysql_ip == "NXDOMAIN"
```yaml
# roles/SimpleDB/tasks/main.yml
---

# ... (skip setting some variables)

# If no service defined then run our install playbook
# This is idempotent so we could run it regardless
- include_tasks: mariadb_install.yml
  when: mysql_ip == "NXDOMAIN"

# Run our upgrade path if we need to change versions
- include_tasks: mariadb_upgrade.yml
  when: version != version_query.json.version
```
Define and deploy the Ansible Operator container which executes an ansible-runner process.
Ansible operator

DS

watch

reconcile

ansible-runner

AS
Instantiate our custom resource object. The operator is listening for any SimpleDB events in our namespace.
apiVersion: example.com/v1alpha1
kind: SimpleDB
metadata:
  name: simpledb
spec:
  # Add fields here
  version: 1
Ansible operator

watch

reconcile

ansible-runner

Ansible playbook or role
This is the only component you need to worry about!
GO FARTHER WITH THESE RESOURCES

- Introducing the operator framework
- water-hole’s ansible-operator repo
- ansible-operator-demo repo
- Awesome operators in the wild
THANKS