CICS TS Voras ecosystem demonstration Kubernetes instructions

Introduction

The following instructions will create a full Voras ecosystem as you would use it in a production system. The ecosystem will be installed into the CICS TS Kubernetes cluster so that the developers can have access to diagnose defects etc. External trials and demos will likely be installed using Docker Compose as those instructions will be considerably easier but the Compose files have not been created yet. All the ecosystem servers will be present, except for the API and Web servers.

All the following steps could be scripted, but for educational purposes, the individual instructions have been listed.

There are similar Helm instructions.

Pre-requisite requirements

1) The Kubernetes command “kubectl” is installed and working.
2) The supplied ~/.kube/config is in place and the namespace set to the one you have been allocated.
3) The etcd3 command “etcdctl” command is installed and working.
4) The etcd3 api has been set to 3 via “export ETCDCTL_API=3” or your O/S equivalent.
5) Maven is installed and working.
6) Apache Karaf has been downloaded and unziped.

Instructions

Create the maven settings and get the k8s yaml files

1) Download the runtime zip from maven and extract into an empty directory from http://archiva.cics-ts.hur.hdclab.intranet.ibm.com:8080/repository/demo/dev.voras/runtime/0.3.0-SNAPSHOT/runtime-0.3.0-SNAPSHOT.zip

2) Create/amend the ~/.m2/settings.xml file with the m2/settings.xml file from step 1.

Deploy the Ecosystem

3) Create the external exposed ports the clients will use to access the ecosystem

```
kubectl apply –f expose.yaml
```

4) List and extract 4 of the ports Kubernetes has allocated to the various servers. We could use ingresses instead. But not at this point (easier if scripted, or helmed).

```
kubectl get svc
```

example output:-

```
NAME                                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE
bootstrap                           ClusterIP   10.99.2.0        <none>        80/TCP              42h
bootstrap-external                  NodePort    10.102.98.104    <none>        80:31339/TCP        42h
cps                                 ClusterIP   10.111.226.36    <none>        2379/TCP,2380/TCP   42h
cps-external                        NodePort    10.98.245.234    <none>        2379:30256/TCP      42h
engine-controller                   ClusterIP   10.100.164.169   <none>        9010/TCP,9011/TCP   42h
engine-controller-metrics-external  NodePort    10.99.84.64      <none>        9010:31203/TCP      42h
metrics                             ClusterIP   10.96.184.107    <none>        9010/TCP,9011/TCP   21h
metrics-external                    NodePort    10.105.93.132    <none>        9010:30672/TCP      21h
ras                                 ClusterIP   10.107.223.126   <none>        5984/TCP            42h
ras-external                        NodePort    10.109.187.158   <none>        5984:32602/TCP      42h
resource-monitor                    ClusterIP   10.99.154.131    <none>        9010/TCP,9011/TCP   42h
resource-monitor-metrics-external   NodePort    10.108.81.147    <none>        9010:31232/TCP      42h
```

You will need to extract 4 ports from this output, the bootstrap-external, in this case 31339, cps-external = 30256, ras-external = 32602 and grafana-external = 88888. The following instructions will refer to these ports as {bootstrap}, {cps}, {ras} and {grafana}.

5) Edit the config-static-bootstrap.yaml file and replace {cps} with the correct port and save

6) Create the config files in k8s

```
kubectl apply -f config-static-bootstrap.yaml
kubectl apply -f config.yaml
kubectl apply -f config-prometheus.yaml
kubectl apply -f config-grafana.yaml
```

7) Apply security rules

```
kubectl apply -f security.yaml
```

8) Start the bootstrap and CPS servers

```
kubectl apply -f server-bootstrap-static.yaml
kubectl apply -f server-cps-etcd3.yaml
```

9) Check the etcd server has started correctly before proceeding

```
kubectl logs -f cps-0
```

and look for “ready to serve client requests”

10) Amend the CPS to point to the DSS and RAS servers, dont forget to substitute the port numbers. (the DSS is sharing the same etcd as the CPS)

```
etcdctl --endpoints=http://cicsk8sm.hursley.ibm.com:{cps} put
framework.dynamicstatus.store
etcd:http://cicsk8sm.hursley.ibm.com:{cps}

etcdctl --endpoints=http://cicsk8sm.hursley.ibm.com:{cps} put
framework.resultarchive.store
couchdb:http://cicsk8sm.hursley.ibm.com:{ras}
```

11) Start the RAS server, the engine controller and the resource management server

```
kubectl apply -f server-ras-couchdb.yaml
kubectl apply -f server-resource-monitor.yaml
kubectl apply -f server-engine-controller.yaml
kubectl apply -f server-metrics.yaml
kubectl apply -f server-prometheus.yaml
kubectl apply -f server-grafana.yaml
```

At this point the ecosystem is up and running

Testing properties

To allow the zos tests to run correctly, the following properties should be set:-

```
etcdctl --endpoints=http://cicsk8sm.hursley.ibm.com:{cps} put
zos.cluster.default.images MV2C,MV2D
etcdctl --endpoints=http://cicsk8sm.hursley.ibm.com:{cps} put
zos.image.max.slots 1
etcdctl --endpoints=http://cicsk8sm.hursley.ibm.com:{cps} put
zos.image.MV2D.max.slots 0
```

All other defaults should be fine. How do we know what properties to set, good question.

Running a test locally

To run a test locally we need to create the basic maven project structure using an archetype, then build it and then we can run it.

1) Setup the local configuration files
mkdir ~/.voras
touch ~/.voras/overrides.properties
touch ~/.voras/cps.properties
touch ~/.voras/dss.properties
touch ~/.voras/credentials.properties

2) In an empty directory, create the demo project, replace {group} with something unique as the tests will deployed to a shared maven repository later on.

```
mvn archetype:generate -DarchetypeGroupId=dev.voras
-DarchetypeArtifactId=voras-demo-archetype
-DarchetypeVersion=0.3.0-SNAPSHOT -DgroupId={group}
```

3) Build the project

```
mvn install
```

4) run the test locally, the voras-boot.jar is from the runtime zip you extracted earlier

```
java -jar {runtimedir}/voras-boot.jar --bootstrap
http://cicsk8sm.hursley.ibm.com:{bootstrap} --obr
mvn:dev.voras/dev.voras.uber.obr/0.3.0-SNAPSHOT/obr --obr mvn:
{group}/demo-obr/1.0-SNAPSHOT/obr --test  demo.tests/demo.FastCore
```

Run the test in automation

In order run the test in automation, the test and obr projects will need to be deployed to a maven repository. We will also need the Voras Karaf command environment to submit the run.

1) Deploy the test project to a maven repository (the distribution repository URL has been hardcoded)

```
mvn deploy
```

2) Start up Apache Karaf
```
{karafdirectory}/bin/karaf
```

3) Install the Karaf maven feature
```
feature:install maven
```

4) Add the internal Voras maven repository to Karaf
```
maven:repository-add -id voras
http://cicscit.hursley.ibm.com/voras/maven@snapshots
```

5) Add the Voras features to Karaf
```
feature:repo-add mvn:dev.voras/dev.voras.uber.karaffeature/0.3.0-
SNAPSHOT/xml
feature:repo-add
mvn:dev.voras/dev.voras.devtools.karaffeature/0.3.0-SNAPSHOT/xml
```

6) Install the Voras framework and devtools
```
feature:install voras-devtools
```

7) Initiallise the Voras framework so that it talks to our new CPS, DSS and RAS
```
voras:init --bootstrap http://cicsk8sm.hursley.ibm.com:{bootstrap}
```

8) In another terminal window you may want to monitor the pods running in k8s
```
watch kubectl get pods -o wide
```

8) Submit your test run
```
run:submit --repo http://archiva.cics-
ts.hur.hdclab.intranet.ibm.com:8080/repository/demo --obr mvn:
{group}/demo-obr/1.0-SNAPSHOT/obr demo.tests demo.SlowCore
```

9) on the O/S command line view the log of the automated run, it may take a few seconds for the pod to start (controller polls every 20 seconds)
```
kubectl logs -f k8s-standard-engine-u1
```

10) in Karaf, submit you can list and display the runs
```
run:list
run:display U1
```

11) If enough development was done, you would be able to retrieve the test structure, the run log and the artifacts
```
ras:display U1
ras:log U1
ras:download U1 1
```

11) Submit 5 zos tests to demonstrate throttling, only 1 slot is available, so they will run 1 by 1, waiting 5 minutes before retrying
```
run:submit --repo http://archiva.cics-
ts.hur.hdclab.intranet.ibm.com:8080/repository/demo --obr mvn:
{group}/demo-obr/1.0-SNAPSHOT/obr --count 10 demo.tests
demo.ThrottleZos
```

12) In a browser, view the Grafana charts http://cicsk8sm.hursley.ibm.com:{grafana}