Kontain Your Spring Workshop Step-by-step

Spring Image Essentials

- Create a new Spring Boot "hello" app
  - With web and actuator dependencies only
- Build it as a JAR: Run it to prove it works
- Build it as an image
- `docker run` the image, using `-p8080:8080` to expose the port
  - curl, httpie, or whatever it to see the results

Spring and Kubernetes

- Create a fresh kind cluster
  - Explore it a bit
- Create a new namespace ("craig") and switch into it (assumes that ns plugin has been installed with Krew)
  - With Krew-installed "ns" plugin:
    - `$ kubectl create namespace craig`
    - `$ kubectl ns craig`
    - `$ k ns` to confirm the current namespace
  - WIthout "ns" plugin
    - `$ kubectl create namespace craig`
    - `$ kubectl config set-context --current --namespace=craig`
    - `$ kubectl config view --minify | grep namespace` to verify current namespace
- Create deploy.yaml and apply
  - Use "deploy" template in STS or...
    - `$ kubectl create deployment hello-k8s --image habuma/hello-k8s:latest -o yaml --dry-run=client > deployment.yaml`
  - `$ kubectl apply -f k8s/deploy.yaml`
  - It probably won't work. Notice image pull error
- Either push to docker hub...
  - `$ docker push habuma/hello-k8s`
- ...or load it into Kind...
  - `$ kind load docker-image habuma/hello-k8s`
- Try applying again (may need to delete first):
  - `$ kubectl delete -f k8s/deploy.yaml; kubectl apply -f k8s/deploy.yaml`
  - Should work this time. Notice "Running" status
- Expose a port:
  - `$ kubectl port-forward [[POD ID]] 8080:8080`
- Curl it:
  - `$ curl localhost:8080/hello`
- Scale up/down in deploy.yaml and re-apply.
  - Observe changes in `kubectl get all`

- Observe changes in `kubectl get all`
- Delete the deployment
  - `$ kubectl delete -f k8s/deploy.yaml`
  - CTRL-C out of port-forward

Skaffold

- Initialize skaffold
  - `$ skaffold init --skip-build`
  - Replace skaffold.yaml with "skaffold" template (and edit image name)
- Run skaffold in dev mode:
  - `$ skaffold dev`
  - Also talk about `skaffold run` and `skaffold delete`. Maybe mention `skaffold build`
- Now make a change...
  - Perhaps enabling ALL actuator endpoints...
  - ...or change the greeting message
  - Observe the change roll out by Skaffold

BREAK TIME

Graceful Shutdown

- Enable graceful shutdown
  - First show what happens if a pod goes away mid-request
  - In application.yml: `server.shutdown: graceful`
  - Try again, shutting down pod while a request is in progress

Liveness and Readiness Probes

- Take a look at liveness and readiness probes
  - Liveness: Is the app alive and healthy or should Kubernetes shut it down and replace it?
  - Readiness: Is the app ready and able to accept traffic?
- Add a new controller that handles POST requests by sending liveness/readiness events

```
@PostMapping("/notready")
public String notReady() {
  AvailabilityChangeEvent.publish(
      applicationContext, ReadinessState.REFUSING_TRAFFIC);
  return "notready";
}

@PostMapping("/ready")
```

```
@PostMapping("/ready")
public String ready() {
  AvailabilityChangeEvent.publish(
      applicationContext, ReadinessState.ACCEPTING_TRAFFIC);
  return "ready";
}

@PostMapping("/dead")
public String dead() {
  AvailabilityChangeEvent.publish(
      applicationContext, LivenessState.BROKEN);
  return "notready";
}

@PostMapping("/alive")
public String alive() {
  AvailabilityChangeEvent.publish(
      applicationContext, LivenessState.CORRECT);
  return "ready";
}
```

- Enable liveness and readiness probes in the deploy.yaml manifest (these are part of the container description and should line up with name and image):

```
livenessProbe:
  initialDelaySeconds: 2
  periodSeconds: 5
  httpGet:
    path: /actuator/health/liveness
    port: 8080
readinessProbe:
  initialDelaySeconds: 2
  periodSeconds: 5
  httpGet:
    path: /actuator/health/readiness
    port: 8080
```

- Try them out,
  - Keep an eye on the `/actuator/health/readiness` and/or `/actuator/health/liveness` endpoints
  - Watch the restarts in `kubectl get all`
  - `stern hello-k8s` to see the Spring app shutdown

- `stern hello-k8s` to see the Spring app shutdown
- Maybe even look at `kubectl describe` to see log of events

Configuration with ConfigMaps

- Create a `ConfigMap` :
  - Use "configmap" template in STS to create the file and `kubectl apply` to create the ConfigMap
  - Or, here's the actual ConfigMap manifest YAML I used:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: hello-config-map
data:
  greeting.message: Hello Kubernetes!
  favorite.color: Blue
  hometown.name: Jal
```

- Add entries to deploy.yaml to mount `ConfigMap` as mounted volume:
  - In deploy.yaml (volumeMounts aligns with image, livenessProbe, readinessProbe, etc. volumes is 2 spaces back)

```
    volumeMounts:
    - name: hello-config
      mountPath: /etc/config/
  volumes:
  - name: hello-config
    configMap:
      name: hello-config-map
```

- SSH into pod (after it has been redeployed) to explore the `/etc/config` folder
  - `$ kubectl exec --stdin --tty {POD ID} -- /bin/bash`
- Use `configtree:/` in deploy to set `SPRING_CONFIG_IMPORTS`
  - In deploy.yaml (env aligns with image, livenessProbe, etc)

```
    env:
    - name: SPRING_CONFIG_IMPORT
      value: "configtree:/etc/config/"
```

- And change the controller to pull greeting from a property instead of hard-coded
  - Create GreetingProps and inject into HelloController
- Hit the hello endpoint to see the greeting from the `ConfigMap`
- Change the `ConfigMap` greeting somehow and reapply
- Examine filesystem in `/etc/config` to confirm the change
- Try the hello endpoint again. Did it change? Probably not
- Restart the app, by hitting the `/dead` endpoint with a POST request

- Try the hello endpoint once more...Did it change? It should have
- Add the refresh endpoint, which is better than killing the app:
  - In pom.xml:

```
<properties>
  <spring-cloud.version>2020.0.1</spring-cloud.version>
</properties>

...

<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter</artifactId>
</dependency>

...

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```