Integration test

Create file integration-test.sh

```
sleep 5s
PORT=$(kubectl -n default get svc ${serviceName} -o json | jq .spec.ports[].nodePort)
echo $PORT
echo $applicationURL:$PORT/$applicationURI
if [[ ! -z "$PORT" ]];
then
   response=$(curl -s $applicationURL:$PORT$applicationURI)
   http_code=$(curl -s -o /dev/null -w "%{http_code}" $applicationURL:$PORT$applicationURI)
   if [[ "$response" == 100 ]];
      then
         echo "Increment Test Passed"
      else
         echo "Increment Test Failed"
         exit 1;
   fi;
   if [[ "$http_code" == 200 ]];
      then
         echo "HTTP Status Code Test Passed"
      else
         echo "HTTP Status code is not 200"
         exit 1;
   fi;
else
      echo "The Service does not have a NodePort"
      exit 1;
fi;
```

The code performs automated testing on a service running on a Kubernetes cluster. Here's what each section of the script does:

Retrieve the NodePort for a Kubernetes service specified by `$serviceName`:

PORT=$(kubectl -n default get svc ${serviceName} -o json | jq .spec.ports[].nodePort)

This command uses `kubectl` to retrieve the service's details in JSON format, and `jq` to extract the NodePort value from the JSON output. The NodePort is stored in the `$PORT` variable.

Output the service's URL: echo $applicationURL:$PORT/$applicationURI

This command outputs the URL that the service can be accessed at, constructed by combining the `$applicationURL`, `$PORT`, and `$applicationURI` variables.

Send HTTP requests to the service and perform tests:

```
if [[ ! -z "$PORT" ]];
```

```
then
   response=$(curl -s $applicationURL:$PORT$applicationURI)
   http_code=$(curl -s -o /dev/null -w "%{http_code}" $applicationURL:$PORT$applicationURI)
   if [[ "$response" == 100 ]];
      then
         echo "Increment Test Passed"
      else
         echo "Increment Test Failed"
         exit 1;
   fi;
   if [[ "$http_code" == 200 ]];
      then
         echo "HTTP Status Code Test Passed"
      else
         echo "HTTP Status code is not 200"
         exit 1;
   fi;
else
   echo "The Service does not have a NodePort"
   exit 1;
fi;
```

In this section, two HTTP requests are sent to the service. The first request retrieves the response body and stores it in the `$response` variable.

Since this application has function +1 value per request, and now we have applicationURL="/increment/99". So we need "$response" == 100 in code. If the response body is "100", the "Increment Test Passed" message is printed to the console.

The second request retrieves the HTTP response code and stores it in the `$http_code` variable. The two `if` statements then perform tests on these values. If the HTTP response code is "200", the "HTTP Status Code Test Passed" message is printed. If either of these tests fail, an error message is printed and the script exits with a non-zero status code. If the `$PORT` variable is empty, an error message is printed and the script exits with a non-zero status code.

Test application:

```
curl -s -o /dev/null -w "%{http_code}" $applicationURL:$PORT$applicationURI
```

This is a `curl` command that retrieves the HTTP status code of a URL's response without printing the response body to the console. Here's what each piece of the command does:

`curl`: the command that makes HTTP requests to web servers.

`-s`: a parameter that tells `curl` to operate in silent mode, meaning that it won't display any progress information or error messages in the console. It's useful for automated testing since it avoids cluttering the output.

`-o /dev/null`: a parameter that redirects the output of the command to `/dev/null`, a special file that discards all output. This means that the body of the response is not returned to the script, which would interfere with the tests.

`-w "%{http_code}"`: a parameter that outputs only the HTTP status code, using a predefined format. `"%"{http_code}` is a `curl` variable that represents the HTTP status code of the response.

`$applicationURL:$PORT$applicationURI`: a dynamic URL that combines the `$applicationURL`, `$PORT`, and `$applicationURI` variables that the script has defined earlier. `$applicationURL` represents the base URL of the service, `$PORT` the NodePort number of the service, and `$applicationURI` a specific endpoint or resource path of the service.

Together, these parameters tell `curl` to send an HTTP request to the constructed URL, retrieve only the HTTP status code from the response headers, and discard the response body. The response status code is then assigned to the `$http_code` variable so that it can be used later in the script for testing purposes to verify that the service is functioning as expected.

Create pipeline Integration Tests and build pipeline.

```
stage('Integration Tests - DEV') {
    steps {
     script {
      try {
       withKubeConfig([credentialsId: 'kubeconfig']) {
        sh "bash integration-test.sh"
       }
      } catch (e) {
       withKubeConfig([credentialsId: 'kubeconfig']) {
        sh "kubectl -n default rollout undo deploy ${deploymentName}"
       }
       throw e
      }
     }
    }
  }
```