Logging and Monitoring in Kubernetes

Learning Outcomes

After completing the lab, you will be able to:

1. Use logback to implement application level logs
2. Externalize log messages
3. Configure application to use actuators
4. Configure observability using probes
5. Access container logs for debugging and troubleshooting

Configure logback logging

We will make use of logback which is intendend to be a successor for slf4j. The choice for using logback also comes for the benefits that we want to use to add multiple appenders for our usecase. For more information on logback refer to http://logback.qos.ch

1. Add the following test method to `HomeApiTest` class

```
    @Test
    public void healthTest(){
        String body = this.restTemplate.getForObject("/actuator/health", String.class)
        assertThat(body).contains("UP");
    }
```

1. Add the acutator dependency in `build.gradle` file and verify all the test cases pass.

   `implementation 'org.springframework.boot:spring-boot-starter-actuator'`

2. We dont need to add any external dependecy for logback, as it comes as a part of web starter dependency.

3. Create a file called `logback.xml` in `src/main/resources` directory and update the file with the configuration as below. Make sure to replace `student-name]` with your namespace name created in K8s.

   This example contains the basic logging configuration for FILE and STDOUT appender

```
<?xml version = "1.0" encoding = "UTF-8"?>
<configuration>
    <appender name="FILE" class="ch.qos.logback.core.FileAppender">
        <file>[student-name]/logs/app.log</file>

        <encoder>
            <pattern>%date %level [%thread] %logger{10} [%file:%line]
%msg%n</pattern>
        </encoder>
    </appender>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppende
r">
        <encoder>
            <pattern>%date %level [%thread] %logger{10} [%file:%line]
%msg%n</pattern>
        </encoder>
    </appender>

    <root level="info">
        <appender-ref ref="FILE" />
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

4. Add imports to refer to logback in PageController

   `import ch.qos.logback.classic.Logger;`

   `import org.slf4j.LoggerFactory;`

5. Create Logger instance in PageController

   `Logger logger =(Logger)LoggerFactory.getLogger(this.getClass());`

6. Update methods in PageController to log messages upon invocation of CRUD operations.

   Refer the code below for example:

```
 @GetMapping("{id}")
    public ResponseEntity<Page> read(@PathVariable long id) {

        logger.info("READ-INFO:Fetching page with id = " + id);
        logger.debug("READ-DEBUG:Fetching page with id = " + id);

        Page page = pageRepository.read(id);
        if(page!=null)
            return new ResponseEntity<Page>(page, HttpStatus.OK);
        else {
            logger.error("READ-ERROR:Could not find page with id = " +
id);
            return new ResponseEntity(HttpStatus.NOT_FOUND);
        }
    }
```

7. Verify the code builds and test cases pass.

8. Run the code. Perform crud operations using curl or postman. Verify the log messages are logged in console and in the external log file.

9. Delete the log file and the directory which was generated by logback as we dont want to checkin the auto-generated directory and file as part of the code.

10. Update the `pages-deployment.yaml` to add `volume` , `volume mount` and `logger configuration` as below

    Replace `[student-name]` with namespace name

```
Create a volume with name -> node-dir
             type  -> hostpath
             path  ->  /[student-name]

Create a volume mount with name -> node-dir
              mountPath -> /[student-name]

Add environment variable with key:value pairs as below:
       DEBUG: "true"
       LOGGING_FILE_NAME: "[student-name]/logs/app.log"
       LOGGING_LEVEL_ORG_SPRINGFRAMEWORK_WEB: debug
       LOGGING_LEVEL_ROOT: debug
```

11. Update the `pages-deployment.yaml` file to enable the actuator endpoints

```
Add the environment variable:
    MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE: *
```

12. Introduce `Liveness probe` to instruct the `kubelet` on when to kill the unhealthy container, subjected to its restart policy. Update the `pages-deployment.yaml` file to add the below spec:

```
  livenessProbe
    type -> httpGet
    path -> /actuator/health
    port ->  8080
    initialDelaySeconds -> 15
    periodSeconds -> 30
```

13. Introduce Readiness probe which indicates whether the application is ready to respond to requests. Update the pages-deployment manifest to add the below spec:

```
  readinessProbe
    type -> tcpSocket
    port ->  8080
    initialDelaySeconds -> 15
    periodSeconds -> 30
```

14. Dockerize the application by creating a new tag `monitor` . Test it once on minikube with the new image. By now, you should be familiar with dockerizing and kubernetizing. If not, refer back to the earlier lab instructions.

Reference Code Snippet for pages-deployment manifest

```
          env:
            - name: PAGE_CONTENT
              valueFrom:
                configMapKeyRef:
                  name: pages-config-map
                  key: PAGE_CONTENT
            - name: SPRING_DATASOURCE_URL
              value: jdbc:mysql://pages-mysql/pages?useSSL=false
            - name: SPRING_DATASOURCE_USERNAME
              value: "root"
            - name: SPRING_DATASOURCE_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-pass
                  key: password
            - name: DEBUG
              value: "true"
            - name: LOGGING_FILE_NAME
              value: "student-name/logs/app.log"
            - name: LOGGING_LEVEL_ORG_SPRINGFRAMEWORK_WEB
              value: debug
            - name: LOGGING_LEVEL_ROOT
              value: debug
            - name: MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE
              value: "*"
          volumeMounts:
            - name: node-dir
              mountPath: /[student-name]
          readinessProbe:
            tcpSocket:
              port: 8080
            initialDelaySeconds: 15
            periodSeconds: 30
          livenessProbe:
            httpGet:
              path: /actuator/health
              port: 8080
            initialDelaySeconds: 15
            periodSeconds: 30
      volumes:
        - name: node-dir
          hostPath:
            path: /[student-name]
```

Viewing Logs in K8s

1. Ensure kubectl context is set to minikube.
2. Use `kubectl logs [pod-name]` for fetching application logs. Refer to http://kubernetes.io/docs for more information.
3. Type the command `minikube dashboard` , which can also be used to view the logs.

Deploy

1. Change the value of tag in `pipeline.yaml` to match the `pages-deployment.yaml` with the tag name `monitor` .
2. Push the code to github repository to start the pipeline, which will deploy the application to the production cluster.

Debug and troubleshoot

1. Delete and recreate the `mysql-deployment`

   Ensure your `kubectl` context is set to minikube.

```
kubectl config current-context

kubectl config get-contexts

kubectl config use-context minikube

kubectl config set-context --current --namespace [student-name]

kubectl delete deploy mysql

kubectl apply -f deployment/mysql-deployment.yaml

kubectl get deploy
```

2. Try to access the pages application

   ```
   kubectl port-forward svc/pages 8080:8080

   curl http://locahost:8080/pages
   ```

3. What error do you see?

4. Find out the root cause of the problem and fix it.

Advanced usecase challenges

1. Implement centralized logging by aggregating the logs and persisting it from all the pods and nodes into a single datastore.

2. Add instrumentation such as counters and guages

3. Implement correlation of events by using correlation ids for all the requests

4. Implement service mesh pattern for designing a unified logging layer.

5. In order to view the logs using Kibana dashboard with EFK on minikube, enable EFK addons on minikube.

   Ensure your kubectl context is set to minikube.

   ```
   minikube addons list

   minikube addons enable efk
   ```

   Explore the Kibana dashboard

   `minikube addons open efk`

   Use `*` when prompted to create index pattern. Select timestamp for filtering and click ok. From the left side menu, select `Discover` and use queries for searching your log messages. Note that, this addon might consume more memory and might not start as expected which is a known issue.