Deploying ELK Stack on Docker Container

DESCRIPTION

Project objective:

You have to deploy ELK Stack on a Docker container to implement continuous monitoring.

Background of the problem statement:

Your manager has asked to create an elegant user interface for data analysis and data visualization as you have worked on ELK stack previously and have the idea of how it works. This will help the DevOps team to monitor and analyze the application behavior.

You must use the following:

- Docker: To create a container for installing ELK Stack
- ELK Stack: To implement continuous monitoring
- Git: To connect and push files from the local system to GitHub
- GitHub: To store the Angular application

Following requirements should be met:

- A few of the source code should be tracked on GitHub repositories. You need to document the tracked files that are ignored during the final push to the GitHub repository.
- Submission of your GitHub repository link is mandatory. In order to track your task, you need to share the link of the repository in the document.
- The step-by-step process involved in completing this task should be documented.

CODE

Create Spring REST Project:-

```
package com.example.howtodoinjava.hellodocker;
import java.util.Date;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
public class HelloDockerApplication {
    public static void main(String[] args) {
        SpringApplication.run(HelloDockerApplication.class, args);
    }
}

@RestController
class HelloDockerRestController {
    @RequestMapping("/hello/{name}")
    public String helloDocker(@PathVariable(value = "name") String name) {
        String response = "Hello " + name + " Response received on : " + new Date();
            System.out.println(response);
        return response;

    }


}
```

application.properties:-

```
server.port = 9080
```

Dockerfile

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ADD target/hello-docker-0.0.1-SNAPSHOT.jar hello-docker-app.jar
ENV JAVA_OPTS=""
ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar
/hello-docker-app.jar" ]
```

pom.xml

```
<plugin>
    <groupId>com.spotify</groupId>
    <artifactId>dockerfile-maven-plugin</artifactId>
    <version>1.3.4</version>
    <configuration>
        <repository>${docker.image.prefix}/${project.artifactId}</repository>
    </configuration>
```

```
</plugin>
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-dependency-plugin</artifactId>
   <executions>
      <execution>
         <id>unpack</id>
         <phase>package</phase>
         <goals>
            <goal>unpack</goal>
         </goals>
         <configuration>
            <artifactItems>
               <artifactItem>
                  <groupId>${project.groupId}</groupId>
                  <artifactId>${project.artifactId}</artifactId>
                  <version>${project.version}</version>
               </artifactItem>
            </artifactItems>
         </configuration>
      </execution>
   </executions>
</plugin>
```

SpringBootDemoApplication.java:-

```
import java.util.Arrays;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.security.SecurityAutoConfiguration;

import org.springframework.context.ApplicationContext;



@SpringBootApplication (exclude = SecurityAutoConfiguration.class)

public class SpringBootDemoApplication {

  public static void main(String[] args)

  {
      ApplicationContext ctx = SpringApplication.run(SpringBootDemoApplication.class, args);

        String[] beanNames = ctx.getBeanDefinitionNames();
        Arrays.sort(beanNames);
        for (String beanName : beanNames)
        {
            System.out.println(beanName);
        }
    }
}
```

<u>EmployeeController.java</u>:-

```
import java.util.ArrayList;
import java.util.List;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.howtodoinjava.demo.model.Employee;

@RestController
public class EmployeeController
{
   @RequestMapping("/")
    public List<Employee> getEmployees()
{
      List<Employee> employeesList = new ArrayList<Employee>();
      employeesList.add(new Employee(1,"lokesh","gupta","howtodoinjava@gmail.com"));
      return employeesList;
    }
}
```

<u>Employee.java</u>:-

```
public class Employee {
   public Employee() {
   }
   public Employee(Integer id, String firstName, String lastName, String email) {
      super();
      this.id = id;
      this.firstName = firstName;
      this.lastName = lastName;
      this.email = email;
   }
private Integer id;
   private String firstName;
   private String lastName;
   private String email;

   //getters and setters

   @Override
   public String toString() {
      return "Employee [id=" + id + ", firstName=" + firstName

          + ", lastName=" + lastName + ", email=" + email + "]";

   }

}
```

<u>ElkExampleSpringBootApplication.java</u>:-

```
package com.example.howtodoinjava.elkexamplespringboot;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.Date;
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.HttpMethod;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class ElkExampleSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(ElkExampleSpringBootApplication.class, args);
    }
}

@RestController
class ELKController {
    private static final Logger LOG = Logger.getLogger(ELKController.class.getName());

    @Autowired
    RestTemplate restTemplete;

    @Bean
    RestTemplate restTemplate() {
        return new RestTemplate();
    }
    @RequestMapping(value = "/elkdemo")
    public String helloWorld() {
        String response = "Hello user ! " + new Date();
        LOG.log(Level.INFO, "/elkdemo - &gt; " + response);

        return response;
    }

    @RequestMapping(value = "/elk")
    public String helloWorld1() {

        String response = restTemplete.exchange("http://localhost:8080/elkdemo",
HttpMethod.GET, null, new ParameterizedTypeReference() {
        }).getBody();
        LOG.log(Level.INFO, "/elk - &gt; " + response);

        try {
            String exceptionrsp = restTemplete.exchange("http://localhost:8080/exception",
HttpMethod.GET, null, new ParameterizedTypeReference() {
            }).getBody();
            LOG.log(Level.INFO, "/elk trying to print exception - &gt; " + exceptionrsp);
            response = response + " === " + exceptionrsp;
        } catch (Exception e) {
            // exception should not reach here. Really bad practice :)
        }

        return response;
    }

    @RequestMapping(value = "/exception")
    public String exception() {
        String rsp = "";
        try {
            int i = 1 / 0;
            // should get exception
        } catch (Exception e) {
            e.printStackTrace();
            LOG.error(e);

            StringWriter sw = new StringWriter();
            PrintWriter pw = new PrintWriter(sw);
            e.printStackTrace(pw);
            String sStackTrace = sw.toString(); // stack trace as a string
            LOG.error("Exception As String :: - &gt; "+sStackTrace);
            rsp = sStackTrace;
        }

        return rsp;
    }
}
```

application.properties:-

```
logging.file=elk-example.log
spring.application.name = elk-example
```

Logstash Configuration

```
input {
  file {
    type => "java"
    path => "F:/Study/eclipse_workspace_mars/elk-example-spring-boot/elk-example.log"
    codec => multiline {
      pattern => "^%{YEAR}-%{MONTHNUM}-%{MONTHDAY} %{TIME}.*"
      negate => "true"
      what => "previous"
    }
  }
}

filter {
  #If log line contains tab character followed by 'at' then we will tag that entry as stacktrace
  if [message] =~ "\tat" {
    grok {
      match => ["message", "^(\tat)"]
      add_tag => ["stacktrace"]
    }
  }

 grok {
    match => [ "message",
               "(?<timestamp>%{YEAR}-%{MONTHNUM}-%{MONTHDAY}
%{TIME})  %{LOGLEVEL:level} %{NUMBER:pid} --- \[(?<thread>[A-Za-z0-9-]+)\] [A-Za-
z0-9.]*\.(?<class>[A-Za-z0-9#_]+)\s*:\s+(?<logmessage>.*)",
               "message",
               "(?<timestamp>%{YEAR}-%{MONTHNUM}-%{MONTHDAY}
%{TIME})  %{LOGLEVEL:level} %{NUMBER:pid} --- .+? :\s+(?<logmessage>.*)"
             ]
  }
  date {
    match => [ "timestamp" , "yyyy-MM-dd HH:mm:ss.SSS" ]
  }
}

output {

  stdout {
    codec => rubydebug
  }

  # Sending properly parsed log events to elasticsearch
  elasticsearch {
    hosts => ["localhost:9200"]
  }
}
```

Kibana Configuration

```
pipeline {
    agent {
        docker {
            image 'maven:3-alpine'
            args '-v /root/.m2:/root/.m2'
        }
    }
    stages {
  stage('Build') {
            steps {
                sh 'mvn -B -DskipTests clean package'
            }
     }
       }
}
```

test stage to your Pipeline

```
  stage('Test') {
  steps {
            sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }
```

```
pipeline {
    agent {
        docker {
            image 'maven:3-alpine'
            args '-v /root/.m2:/root/.m2'
        }
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn -B -DskipTests clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
                  }
        }
    }
}
```

Test stage of your Jenkinsfile:

```
        stage('Deliver') {
            steps {
                sh './jenkins/scripts/deliver.sh'
            }
        }
```

and add a skipStagesAfterUnstable option so that you end up with:

```
pipeline {
    agent {
        docker {
            image 'maven:3-alpine'
            args '-v /root/.m2:/root/.m2'
        }
    }
    options {
        skipStagesAfterUnstable()
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn -B -DskipTests clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }
        stage('Deliver') {
            steps {
                sh './jenkins/scripts/deliver.sh'
            }
        }
    }
}
```