**here's an example Java code snippet that uses TestNG to run Selenium tests in parallel:**

import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.chrome.ChromeDriver;  
import org.testng.annotations.AfterMethod;  
import org.testng.annotations.BeforeMethod;  
import org.testng.annotations.Test;  
  
public class ParallelTest {  
    private WebDriver driver;  
  
    @BeforeMethod  
    public void setup() {  
        // Set up ChromeDriver  
        System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");  
        driver = new ChromeDriver();  
    }  
  
    @Test(threadPoolSize = 3, invocationCount = 6)  
    public void test() {  
        // Navigate to website  
        driver.get("[http://www.example.com](http://www.example.com/)");  
        // Perform actions on the page  
        // ...  
    }  
  
    @AfterMethod  
    public void teardown() {  
        // Quit the WebDriver instance  
        driver.quit();  
    }  
}

In this example, we use the **@Test** annotation to mark the test method as needing to be run in parallel. We set the **threadPoolSize** attribute to 3, indicating that we want to run 3 instances of the test method concurrently. We also set the **invocationCount** attribute to 6, indicating that we want to run the test method a total of 6 times.

Note that in order to run tests in parallel, you will need to have multiple instances of the WebDriver. In this example, we create a new instance of ChromeDriver for each test method using the **@BeforeMethod** and **@AfterMethod** annotations to set up and tear down the WebDriver instance.

To run the tests in parallel, you can use a test runner like TestNG or JUnit that supports parallel execution. For example, to run this test class in parallel using TestNG, you would create a testng.xml file with the following contents:

<!DOCTYPE suite SYSTEM "<http://testng.org/testng-1.0.dtd>" >  
<suite name="Parallel Test Suite" parallel="methods" thread-count="3">  
  <test name="Parallel Test">  
    <classes>  
      <class name="ParallelTest" />  
    </classes>  
  </test>  
</suite>

This will run the **ParallelTest** class in parallel using 3 threads. You can then run the tests using TestNG from the command line:

$ java -cp "path/to/testng.jar:path/to/project/classes" org.testng.TestNG testng.xml

This will execute the tests in parallel, with each test running on its own instance of the WebDriver.

To dockerize the above Java code for running Selenium tests in parallel, you can follow these steps:

1. Create a new file called **Dockerfile** in the root directory of your project.
2. Use an official Java runtime as the base image, such as **openjdk:11-jre-slim**.
3. Set the working directory to **/app** and copy the contents of your project to the container.
4. Install any dependencies needed for your project, such as ChromeDriver.
5. Set any necessary environment variables, such as the display for running tests with a graphical interface.
6. Build and run the container.

Here's an example **Dockerfile** that you can use:

# Use an official Java runtime as the parent image  
FROM openjdk:11-jre-slim  
  
# Set the working directory to /app  
WORKDIR /app  
  
# Copy the current directory contents into the container at /app  
COPY . /app  
  
# Install dependencies  
RUN apt-get update && \  
    apt-get install -y curl unzip xvfb && \  
    curl -sSLO <https://chromedriver.storage.googleapis.com/94.0.4606.61/chromedriver_linux64.zip> && \  
    unzip chromedriver\_linux64.zip && \  
    rm chromedriver\_linux64.zip && \  
    mv chromedriver /usr/bin/chromedriver && \  
    chown root:root /usr/bin/chromedriver && \  
    chmod +x /usr/bin/chromedriver  
  
# Set environment variables  
ENV DISPLAY=:99  
  
# Build and run the tests  
CMD ["./gradlew", "clean", "test", "--no-daemon"]

In this example, we install dependencies such as **curl**, **unzip**, and **xvfb** for running tests with a graphical interface. We also download and install ChromeDriver as we did in the previous example.

Note that we set the environment variable **DISPLAY** to **:99**, which is necessary for running tests with a graphical interface inside a container.

Finally, we use the **CMD** instruction to run the **gradlew** script with the **clean** and **test** tasks to build and run the tests.

To build and run the container, you can use the following commands:

# Build the Docker image  
docker build -t my-app .  
  
# Run the Docker container  
docker run --rm -it my-app

This will build the Docker image using the **Dockerfile** in the current directory and tag it as **my-app**. Then it will run the container and execute the **CMD** instruction to build and run the tests inside the container.

Note that the **--rm** option tells Docker to remove the container after it exits, and the **-it** option tells Docker to allocate a pseudo-TTY and keep stdin open so that you can see the output from the tests.

Here's the full **build.gradle** file:

plugins {  
    id 'java'  
    id 'io.qameta.allure' version '2.10.0'  
}  
  
group 'com.example'  
version '1.0-SNAPSHOT'  
  
repositories {  
    mavenCentral()  
}  
  
dependencies {  
    testCompile group: 'junit', name: 'junit', version: '4.12'  
    testCompile 'org.seleniumhq.selenium:selenium-java:3.141.59'  
    testCompile 'org.testng:testng:7.4.0'  
    testCompile 'io.qameta.allure:allure-testng:2.13.5'  
}  
  
test {  
    useTestNG()  
}  
  
allure {  
    version = '2.13.5'  
}

And here's the updated **Dockerfile**:

# Use an official Java runtime as the parent image  
FROM openjdk:11-jre-slim  
  
# Set the working directory to /app  
WORKDIR /app  
  
# Copy the current directory contents into the container at /app  
COPY . /app  
  
# Install dependencies  
RUN apt-get update && \  
    apt-get install -y curl unzip xvfb && \  
    curl -sSLO <https://chromedriver.storage.googleapis.com/94.0.4606.61/chromedriver_linux64.zip> && \  
    unzip chromedriver\_linux64.zip && \  
    rm chromedriver\_linux64.zip && \  
    mv chromedriver /usr/bin/chromedriver && \  
    chown root:root /usr/bin/chromedriver && \  
    chmod +x /usr/bin/chromedriver  
  
# Set environment variables  
ENV DISPLAY=:99  
  
# Install Allure command line tool  
RUN curl -s