==========

Tutor: **Amuthan Sakhtivel**

Reference: **Youtube**

Course: **Github Actions in Test Automation**

==========

1. **Document prepared by:**
    a. **Rajat Verma**
        i. https://www.linkedin.com/in/rajat-v-3b0685128/
        ii. https://github.com/rajatt95
        iii. https://rajatt95.github.io/

2. **More documents:**
    a. https://github.com/rajatt95/Documents

3. **Last worked on this Document:**
    a. Dec 13, 2022

4. **Learnings from Tutor (Code Repository):**
    a. This course
        i. https://github.com/stars/rajatt95/lists/youtube-tmb-github-actions
    b. Other course(s):
        i. https://github.com/stars/rajatt95/lists/testing-mini-bytes-amuthan

—----------

5. **Softwares:**
    a. Programming language - **Java**
    b. IDE - **IntelliJ**
    c. CI/CD -
        i. Github Actions
            1. Tests Execution
                a. Web - **Selenoid, Selenium WebDriver**
                b. Mobile - **Appium**
    d. Build and Dependency Management Tool - **Maven**

—----------

—----------

6. **Course content:**
    a. Part 1 | Github Actions Tutorial 🔥 Step by Step | Introduction | Github Actions in Test Automation
    b. Part 2 - Advantages of Github Action over Jenkins | Actions Marketplace | Custom Github Actions
    c. Part 3 - Workflow Triggers | Different Types of Triggers in Github Actions | Scheduling Workflow |
    d. Part 4 | Contexts | Different Contexts in Github Actions | How to use Contexts in GitHub Actions |
    e. Part 5 | Jobs in Github Actions | Containers in Github Actions | Running Jobs in Parallel |
    f. Part 6 - Steps in Github Actions 🔥 | Step Failure | Timeout for a Step | Github Actions Tutorial |
    g. Part 7 | Running web test in Github Actions | Setup Selenoid in Github Actions | Passing secrets |
    h. Part 8 | Reduce workflow time | Caching Maven | Caching Docker Image Layers | Github Actions
    i. Part 9 | No Cloud Providers Needed 🔥 | Run Android Appium Tests inside Github Runner 🔥
    j. Part 10 | Run iOS Appium Tests in Github Runner 🔥 | No cloud providers needed |
    k. Part 11 | Trigger Automated Test workflow in one repo from Development Workflow in another repo

—----------

1. **Learnings from Course (Youtube - TMB - Github Actions)**
    a. **Links:**
        i. **Github**
            1. https://github.com/actions
            2. https://github.com/features/actions
            3. https://docs.github.com/en/actions/learn-github-actions/contexts
            4. **Github Marketplace**
                a. https://github.com/marketplace?type=
                b. https://github.com/marketplace/actions/maven-cache
                c. https://github.com/marketplace/actions/android-emulator-runner
                d. https://github.com/marketplace/actions/repository-dispatch
            5. **Runner Images**
                a. https://github.com/actions/runner-images
                b. Machines
                    i. https://github.com/actions/runner-images/blob/main/images/linux/Ubuntu2204-Readme.md

ii. https://github.com/actions/runner-images/blob/main/images/macos/macos-12-Readme.md

6. **Generate Personal Access Token**
    a. https://github.com/settings/tokens

—----------

b. Github Actions Advantages:

i. No installation needed
ii. Can execute jobs in parallel with no efforts
iii. Actions will be available for almost all possible setups
iv. Access rights to work with Github Actions

—----------

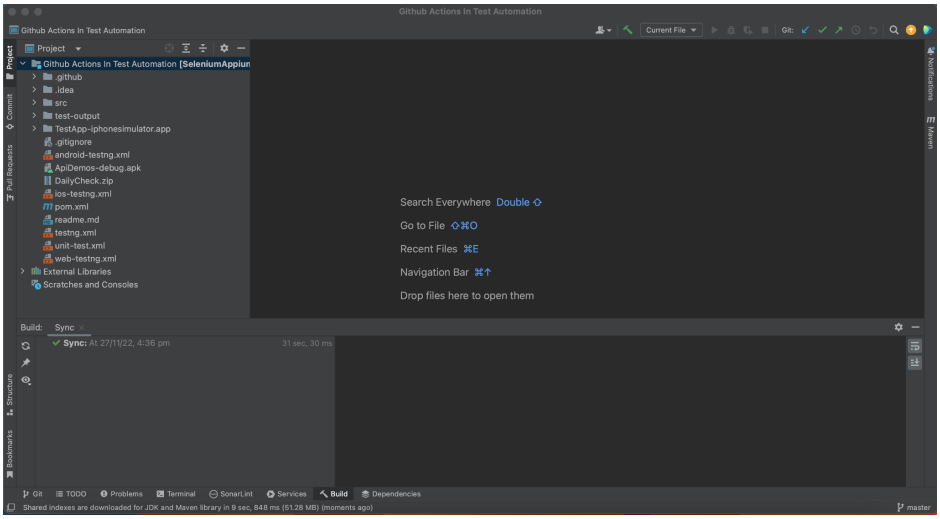
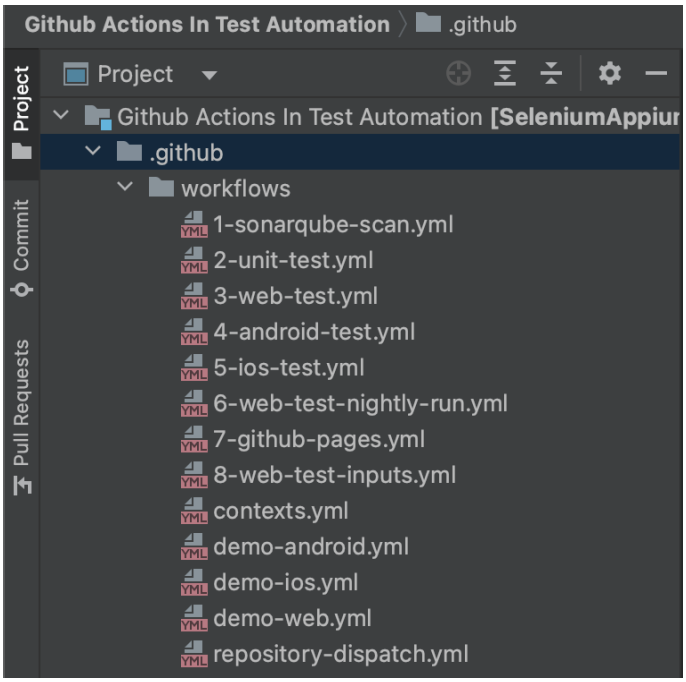
c. Terminologies

i. Workflows
    1. Execution
        a. Manual
        b. Automated
            i. Push
            ii. Pull_Request
            iii. Schedule
ii. Actions
iii. Workflow Triggers
iv. Jobs
v. Steps
vi. Parameters passing using secrets
vii. Contexts
viii. Job dependency
ix. Caching Maven | Caching Docker Image Layers
x. Tests
    1. Unit
    2. Web
    3. Mobile (Android, iOS)
xi. Trigger Automated Test workflow in one repo from Development Workflow in another repo

==========

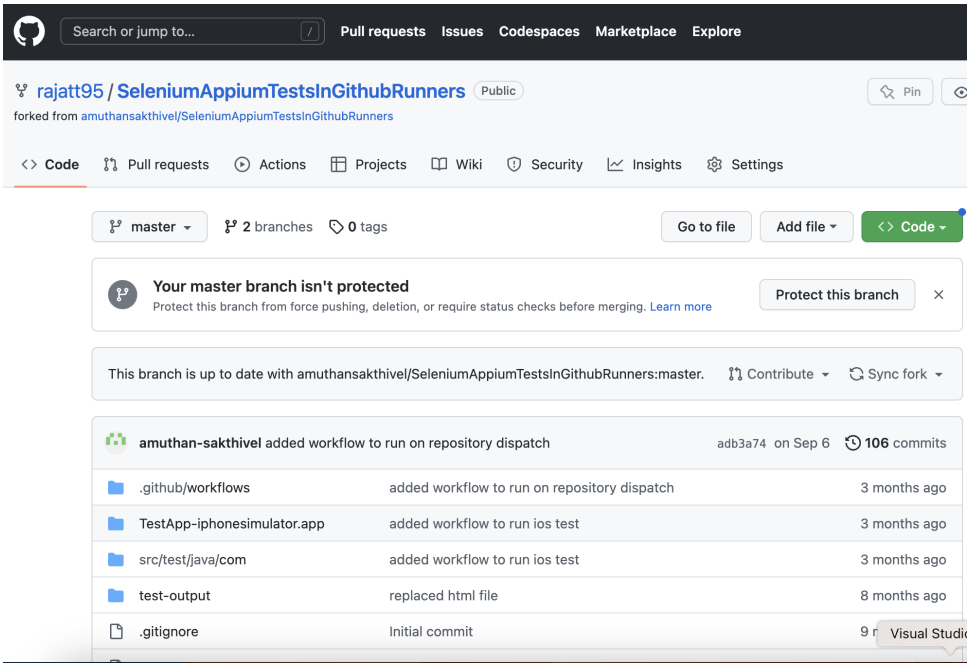
========1_Part 1 | Github Actions Tutorial 🔥 Step by Step | Introduction | Github Actions in Test Automation==========

---

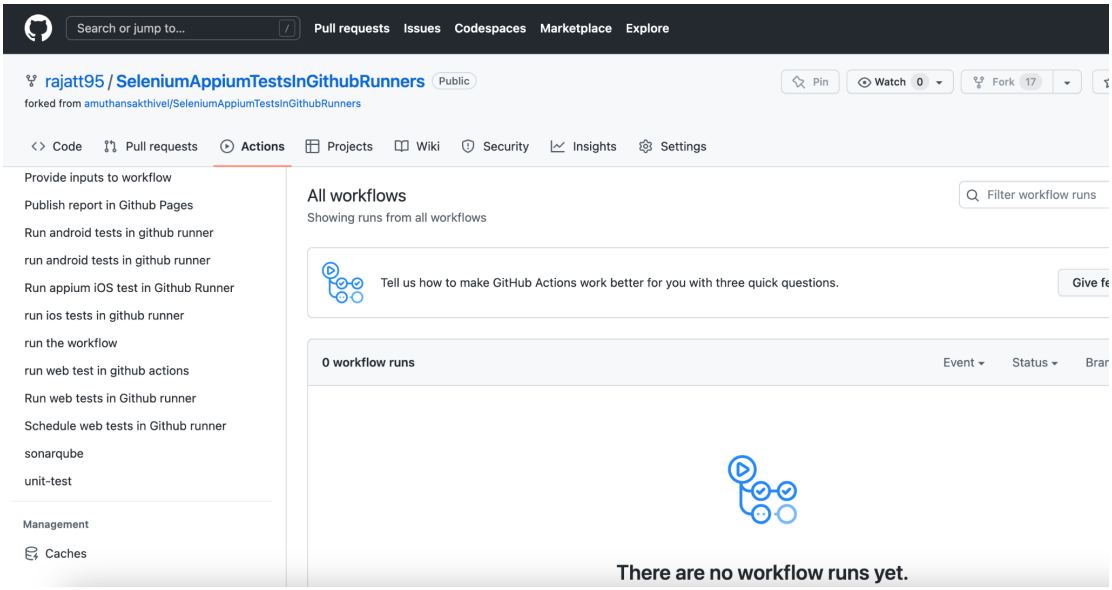
1. Github link for the Tests
    a. https://github.com/rajatt95/SeleniumAppiumTestsInGithubRunners
    b. Clone/Pull/Download this project and import in an IDE
    c. 
    d. You'll see any folder/directory named .github
        i. 
            1. **Workflow** in Github Actions is similar to **Pipelines/Jobs** in Jenkins.

---

1. Go to https://github.com/rajatt95/SeleniumAppiumTestsInGithubRunners

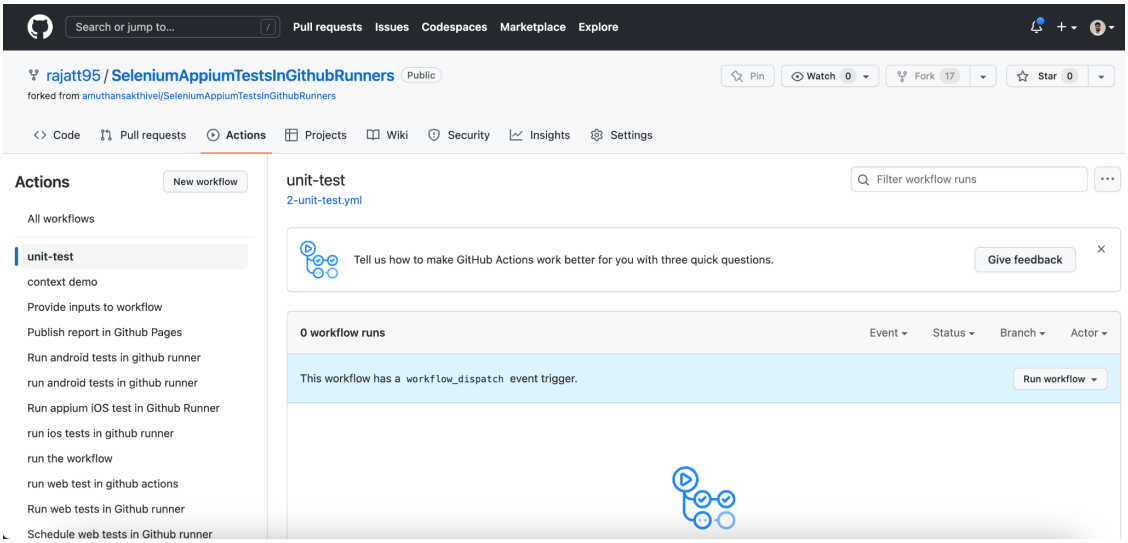
a. 

2. Click on the **Actions** (Tab #3)

a. 

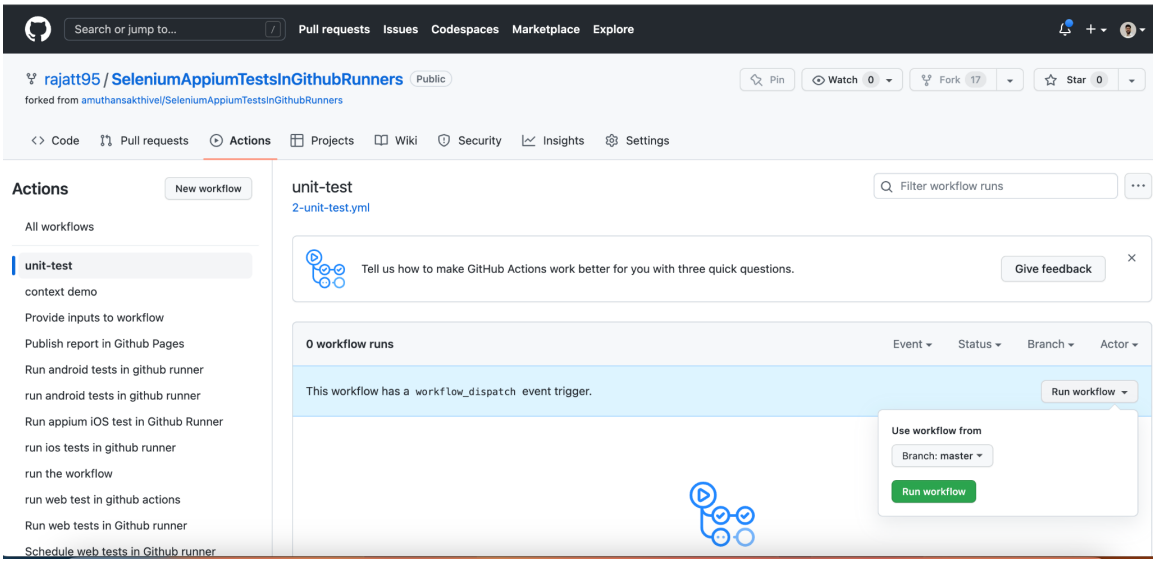
   i. We'll be able to see all the Workflows

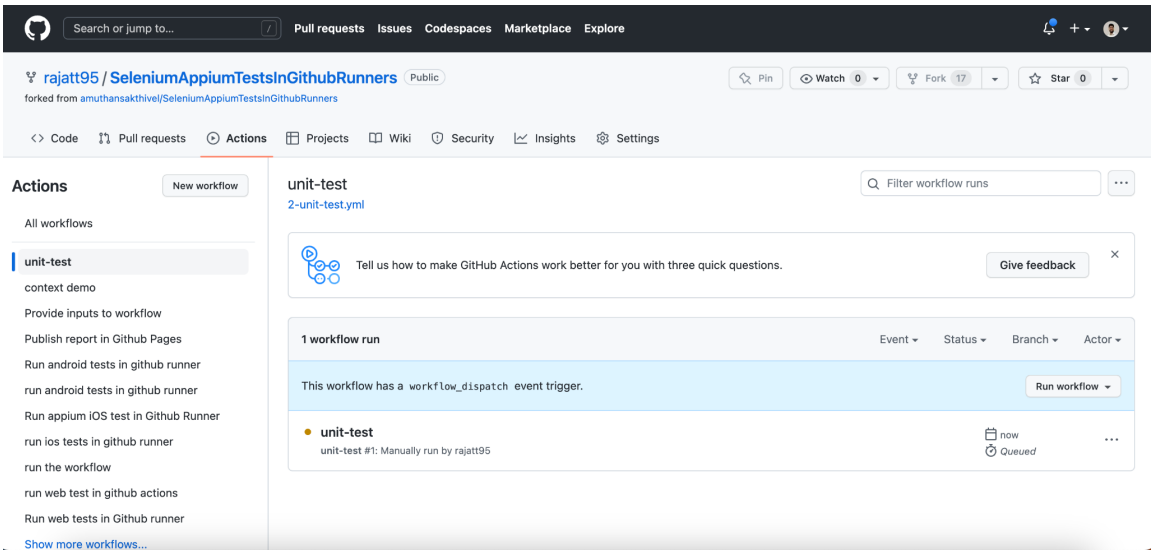
3. Click on the unit-test

a. 

i. We can see all details for Workflow - **unit-test.**

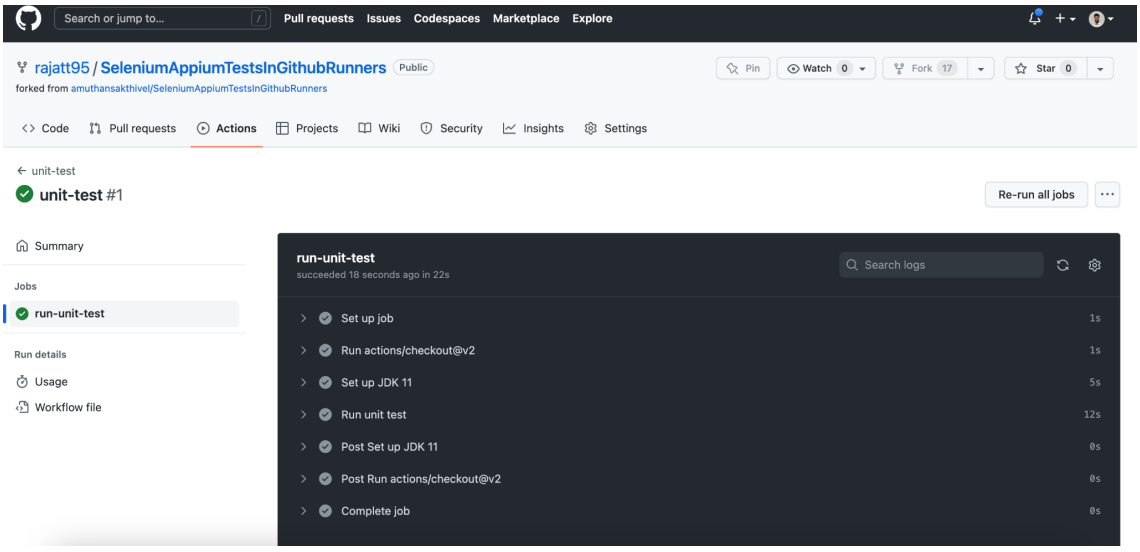
4. Click on the **Run Workflow** button

a. 

b. 

---

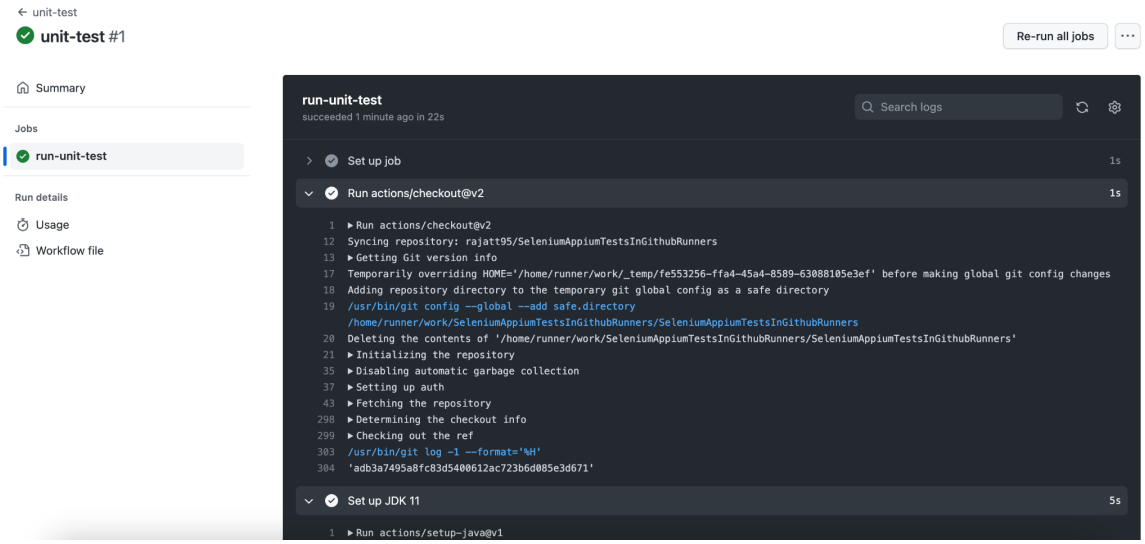
5. Click on **unit-test** to see the Logs

a. 

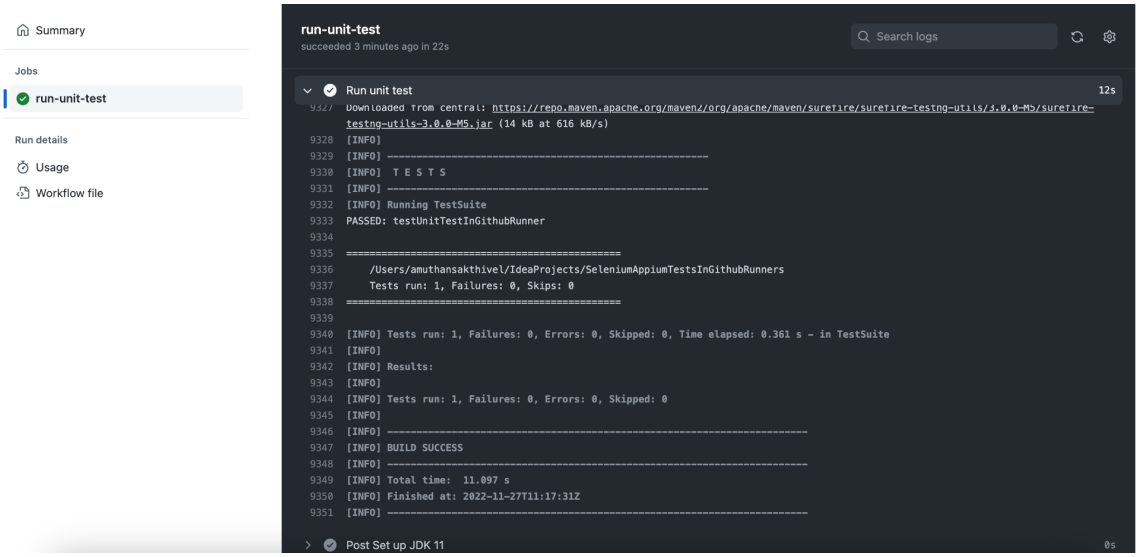
---

6. We can expand these links to see the details logs

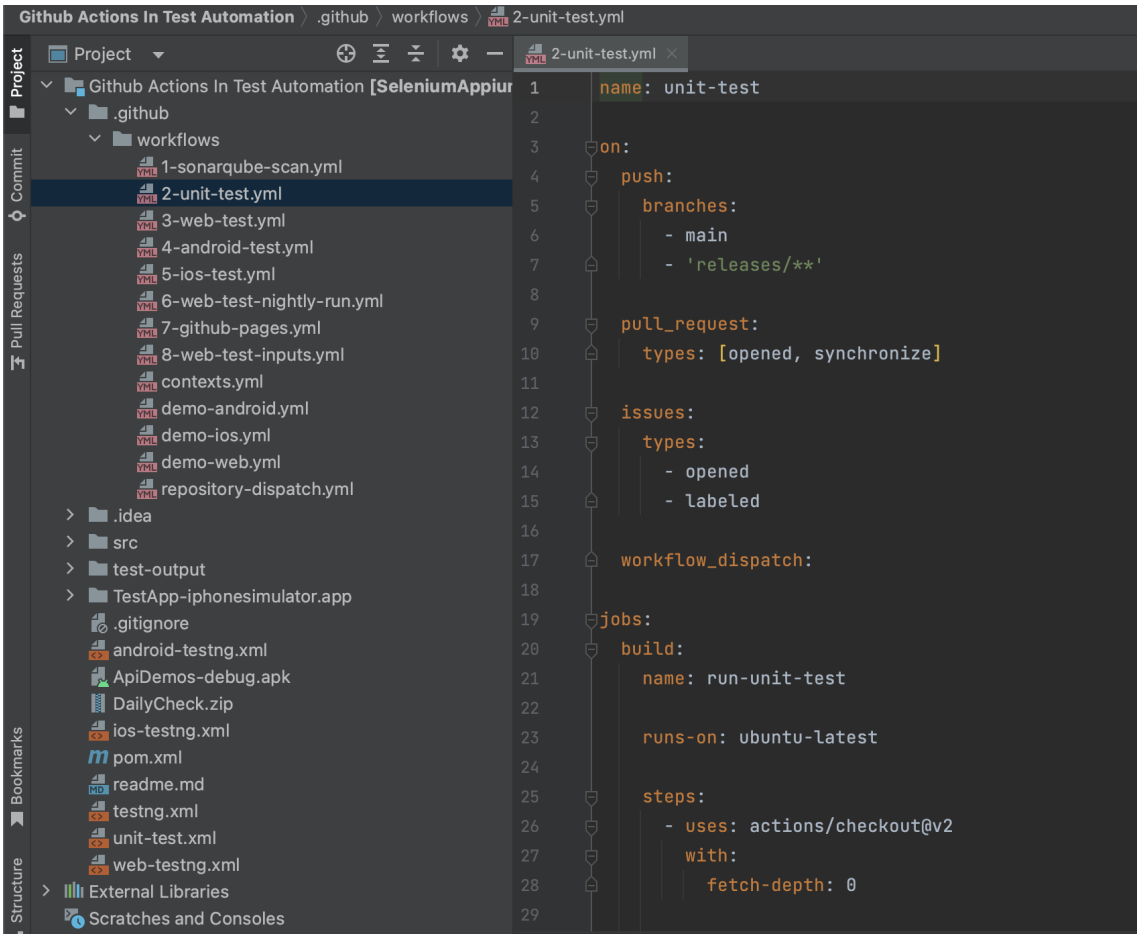
a.


unit-test
unit-test #1
Re-run all jobs
Summary
Jobs
run-unit-test
Run details
Usage
Workflow file
run-unit-test
succeeded 1 minute ago in 22s
Search logs
Set up job 1s
Run actions/checkout@v2 1s
1 ▶ Run actions/checkout@v2
12 Syncing repository: rajatt95/SeleniumAppiumTestsInGithubRunners
13 ▶ Getting Git version info
17 Temporarily overriding HOME='/home/runner/work/_temp/fe553256-ffa4-45a4-8589-63088105e3ef' before making global git config changes
18 Adding repository directory to the temporary git global config as a safe directory
19 /usr/bin/git config --global --add safe.directory
/home/runner/work/SeleniumAppiumTestsInGithubRunners/SeleniumAppiumTestsInGithubRunners
20 Deleting the contents of '/home/runner/work/SeleniumAppiumTestsInGithubRunners/SeleniumAppiumTestsInGithubRunners'
21 ▶ Initializing the repository
35 ▶ Disabling automatic garbage collection
37 ▶ Setting up auth
43 ▶ Fetching the repository
298 ▶ Determining the checkout info
299 ▶ Checking out the ref
303 /usr/bin/git log -1 --format='%H'
304 'adb3a7495a8fc83d5400612ac723b6d085e3d671'
Set up JDK 11 5s
1 ▶ Run actions/setup-java@v1

b.


Summary
Jobs
run-unit-test
Run details
Usage
Workflow file
run-unit-test
succeeded 3 minutes ago in 22s
Search logs
Run unit test 12s
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-testng-utils/3.0.0-M5/surefire-testng-utils-3.0.0-M5.jar (14 kB at 616 kB/s)
[INFO]
[INFO] ---
[INFO] T E S T S
[INFO] ---
[INFO] Running TestSuite
PASSED: testUnitTestInGithubRunner
Tests run: 1, Failures: 0, Skips: 0
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.361 s - in TestSuite
[INFO] Results:
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] BUILD SUCCESS
[INFO] Total time: 11.097 s
[INFO] Finished at: 2022-11-27T11:17:31Z
Post Set up JDK 11 0s

---

1. Open the project in IDE
2. We have a Workflow created named **2-unit-test.yml**
   a. 

      i. This Workflow has only 1 job
         1. But, we can have multiple jobs for 1 Workflow
         2. When we have **multiple jobs**,
            a. All those jobs run **parallel**.
            b. Customization
               i. Possible to do - One job is dependent on another.

—----------

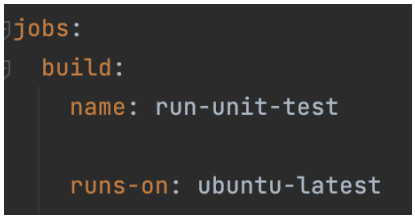
b. Keywords

i. **on -** When I want to trigger this Workflow

1.
2. push
    a. branches
3. pull_request
    a. types
4. **workflow_dispatch** - I want an option to trigger this Workflow manually as well

ii. jobs

1.
2. build



    a.
    b. name
    c. **runs-on -** Where I want to run/execute the job (Github Actions support WIN/MAC/Linux)
    d. Steps

i.
```yaml
steps:
  - uses: actions/checkout@v2
    with:
      fetch-depth: 0

  - name: Set up JDK 11
    uses: actions/setup-java@v1
    with:
      java-version: 11

  - name: Run unit test
    run: mvn clean test -Punit-test
```

ii. **name** - name of step

iii. **uses -** Pull the project in the machine selected (Here, we have taken ubuntu-latest)

iv. with
    1. Fetch-depth

v. run
    1. Which command to execute

—----------

1. We'll cover these **Trigger events** in detail later.

—----------

1. For execution, we need
    a. Machine - On which tests will execute
    b. Project - which will have the tests (Web, Mobile, API)
    c. Softwares setup - Java

—----------

1. In Workflow **(unit-test)**
    a. We are executing the command -

i.
```yaml
  - name: Run unit test
    run: mvn clean test -Punit-test
```

    1. -P is for Profile.
    2. Tutor has used the Maven Profile
        a. In pom.xml, he has maintained a Maven Profile named 'unit-test'.

b.

2-unit-test.yml | pom.xml (SeleniumAppiumTestsInGithubRunners)

```xml
<profiles>
    <profile>
        <id>unit-test</id>
        <build>
            <pluginManagement>
                <plugins>
                    <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-surefire-plugin</artifactId>
                        <version>3.0.0-M5</version>
                        <configuration>
                            <suiteXmlFiles>
                                <suiteXmlFile>unit-test.xml</suiteXmlFile>
                            </suiteXmlFiles>
                        </configuration>
                    </plugin>
                </plugins>
            </pluginManagement>
        </build>
    </profile>
```

—----------

1. Till now, we have only executed 1 Unit test, but, we want to execute Tests created for
    a. Web application using Selenium WebDriver
    b. Mobile application using Appium

—----------

2. For **Web**

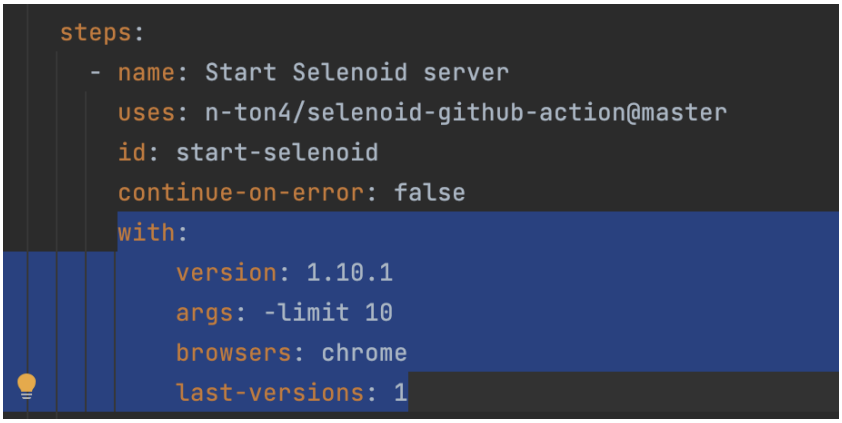
a.

3-web-test.yml

```yaml
name: Run web tests in Github runner

on:
  push:
    branches: [ main ]

  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Start Selenoid server
        uses: n-ton4/selenoid-github-action@master
        id: start-selenoid
        continue-on-error: false
        with:
            version: 1.10.1
            args: -limit 10
            browsers: chrome
            last-versions: 1

      - name: checkout
        uses: actions/checkout@v2

      - name: Run the tests
        run:
            mvn clean test -Pweb
```

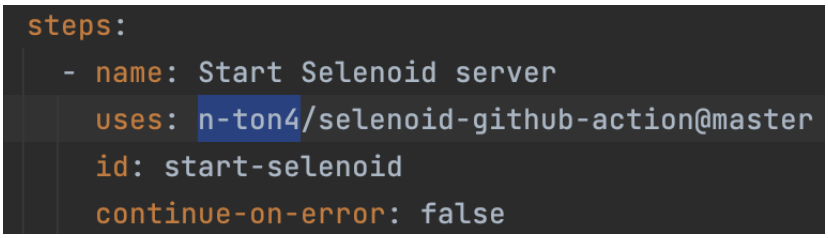
i. This machine 'ubuntu-latest' already has a set of software installed
    1. Browser - Chrome
    2. Android Emulators

ii. Chrome

1. 

    a. Allowing 10 instances of Browers for parallel execution

iii. One thing to note, we do not have actions keyword here

1. 

    a. What does this **n-ton4** mean?
    b. This means this is not developed by Github. Someone has developed this Utility and pushed it to the Github marketplace.

—----------

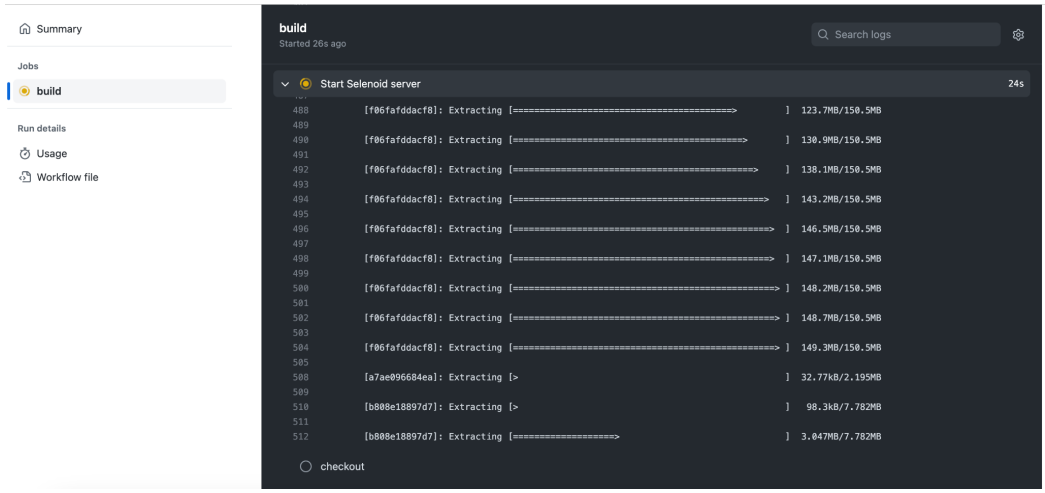
1. Test case written for **Web**

a.

```java
package com.web;

import ...

2 usages  amuthan-sakthivel
public class WebTest {

    amuthan-sakthivel
    @SneakyThrows
    @Test(groups = "web")
    public void testGoogleSearchUsingSelenoid() {
        System.out.println("username :" + System.getProperty("username"));
        System.out.println("password :" + System.getProperty("password"));

        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability( capabilityName: "browserName", value: "chrome");
        capabilities.setCapability( capabilityName: "enableVNC", value: true);
        capabilities.setCapability( capabilityName: "enableVideo", value: false);
        WebDriver driver = new RemoteWebDriver(new URL( spec: "http://localhost:4444/wd/hub"),capabilities);
        driver.get("https://google.co.in");
        Assert.assertEquals(driver.getTitle(), expected: "Google");
        driver.quit();
    }
}
```

2. To execute this Workflow manually,
    a. Go to https://github.com/rajatt95/SeleniumAppiumTestsInGithubRunners
    b. Click on **Actions**
    c. Click on **Run Web Tests in Github Runner**

i.



—----------

1. **Solenoid**
    a. It is a customized Selenium Hub Selenium Grid setup
        i. Very optimized

—----------

1. Till now, we have used the machine that Github Action has provided us
    a. That is also possible if you want to set up your custom machine.

========2_Part 2 - Advantages of Github Action over Jenkins | Actions Marketplace | Custom Github Actions==========

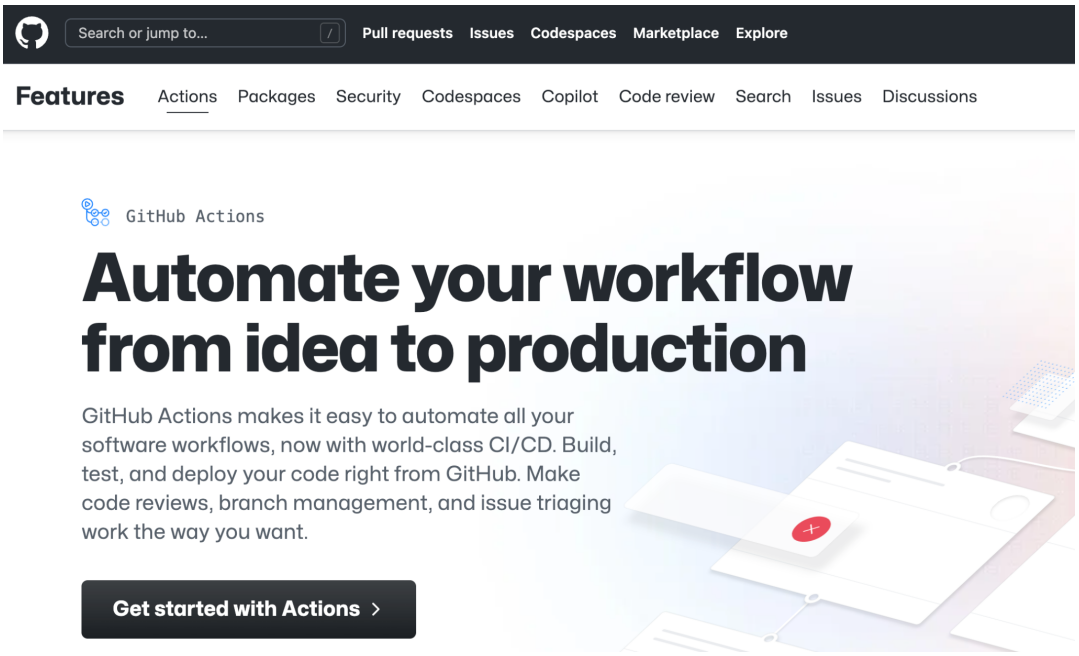
---

1. **Advantages:**
    a. No installation needed
        i. Earlier, we were working with Jenkins
            1. We had to maintain a machine (that was working as a Host)
            2. Software setup in that machine
                a. Java path
                b. Maven path
                c. Download Plugins
        ii. Now, with Github Actions
            1. Everything is set up in cloud
            2. Ready-to-use
    b. Can execute jobs in parallel with no efforts
        i. Add one job in the .yaml file and that's all.
    c. Actions will be available for almost all possible setups
        i. Setup is ready to use - Calling it **Actions**
        ii. Actions are pushed into Github marketplace
            1. Re-Usable
            2. People have contributed and pushed
    d. Access rights to work with Github Actions
        i. Earlier, Jenkins -> Access
            1. Test Automation Team (Very rare)
        ii. Now, we maintain a code repository for test case
            1. We have the access rights to perform different operations

—----------

—----------

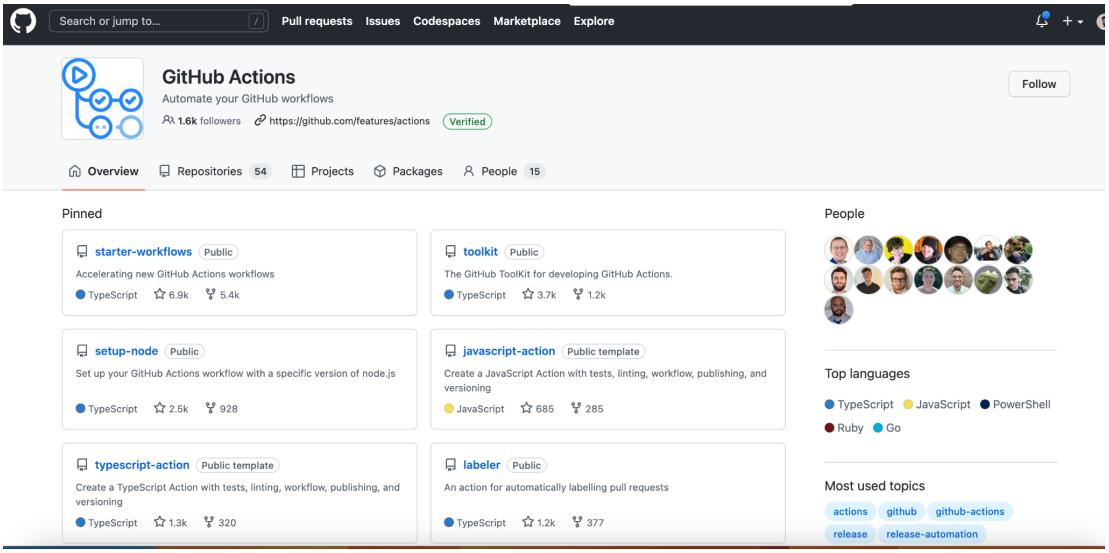
1. **Github Actions**
    a. https://github.com/features/actions



i.

—----------

b. https://github.com/actions



i.

—----------

c. To set up Java
    i. https://github.com/actions/setup-java
    ii. This is an official action (Developed by Github Team)
        1. Actions present under
            a. https://github.com/actions

README.md

Basic Configuration

Eclipse Temurin

```yaml
steps:
- uses: actions/checkout@v3
- uses: actions/setup-java@v3
  with:
    distribution: 'temurin' # See 'Supported distributions' for available options
    java-version: '17'
- run: java HelloWorldApp.java
```

Azul Zulu OpenJDK

```yaml
steps:
- uses: actions/checkout@v3
- uses: actions/setup-java@v3
  with:
    distribution: 'zulu' # See 'Supported distributions' for available options
    java-version: '17'
- run: java HelloWorldApp.java
```

2.
3.
4. These actions

—----------

1. **Github Marketplace**
    a. https://github.com/marketplace?type=

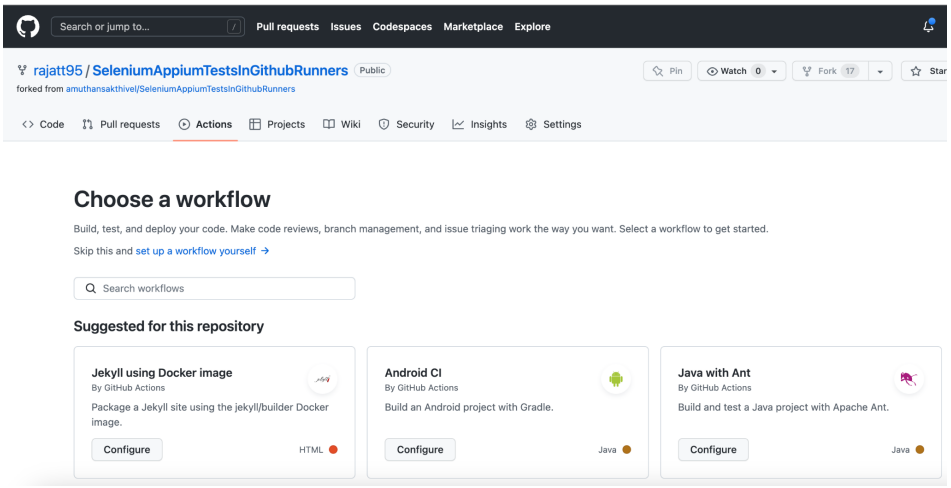
—----------

========3_Part 3 - Workflow Triggers | Different Types of Triggers in Github Actions | Scheduling Workflow |==========

---

1. **Workflow Triggers**
    a. Workflow Triggers are events that cause Workflows to run
    b. There are multiple ways to trigger a Workflow
    c. Triggers
        i. Example
            1. Execute test cases daily
            2. Execute on code push
            3. Execute on PR raised

—----------

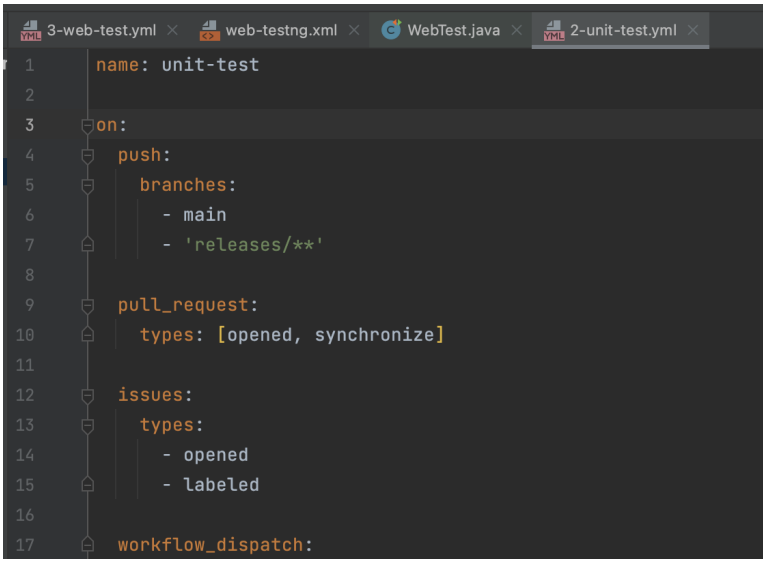
1. **To create Workflow**
    a. Go to the code repo
    b. Click on Actions
    c. Click on New workflow
        i. https://github.com/rajatt95/SeleniumAppiumTestsInGithubRunners/actions/new
            1. There are many Templates available as well
            2. 

        ii. Click on Setup a Workflow yourself
            1. These workflows will be saved inside the **.github folder**
            2. You can give names accordingly
                a. **Execution-daily.yml**
                b. # is used for comment in .yml file

—----------

---

1. Unit-test.yml file

a. 

b. Here, the Triggers are

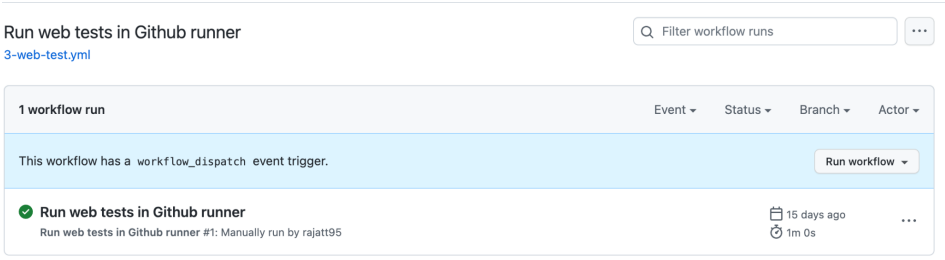
   i. **Push**

      1. Trigger the job when there is a code push on the main branch or "releases/**"

   ii. **Pull Request**

      1. Types
         a. Opened
         b. Synchronize

   iii. **Workflow Dispatch**

      1. For manual trigger -
         a. We'll be able to trigger the Workflow manually from Github repository
         b. 

   iv. There are many other ways as well for Triggers

      1. **branches-ignore**

—----------

1. Schedule over a period of time

a.

6-web-test-nightly-run.yml

```yaml
name: Schedule web tests in Github runner
#UTC
on:
  schedule:
    - cron: '0 22 * * *'
jobs:
  build:
    runs-on: ubuntu-latest
```

   i. Every night 10 PM
      1. **schedule** is the keyword
      2. The default time followed is **UTC**

—----------

1. **Parameters passing**
   a. **Jenkins**
      i. We had the option to pass the parameters value
   b. **Github Actions**

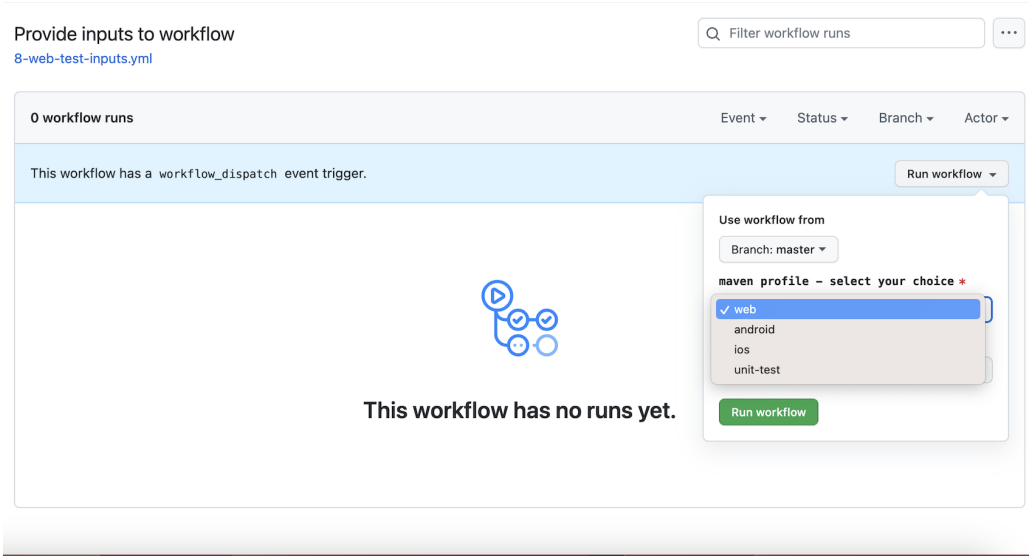
i.

6-web-test-nightly-run.yml | 8-web-test-inputs.yml | 3-web-test.yml

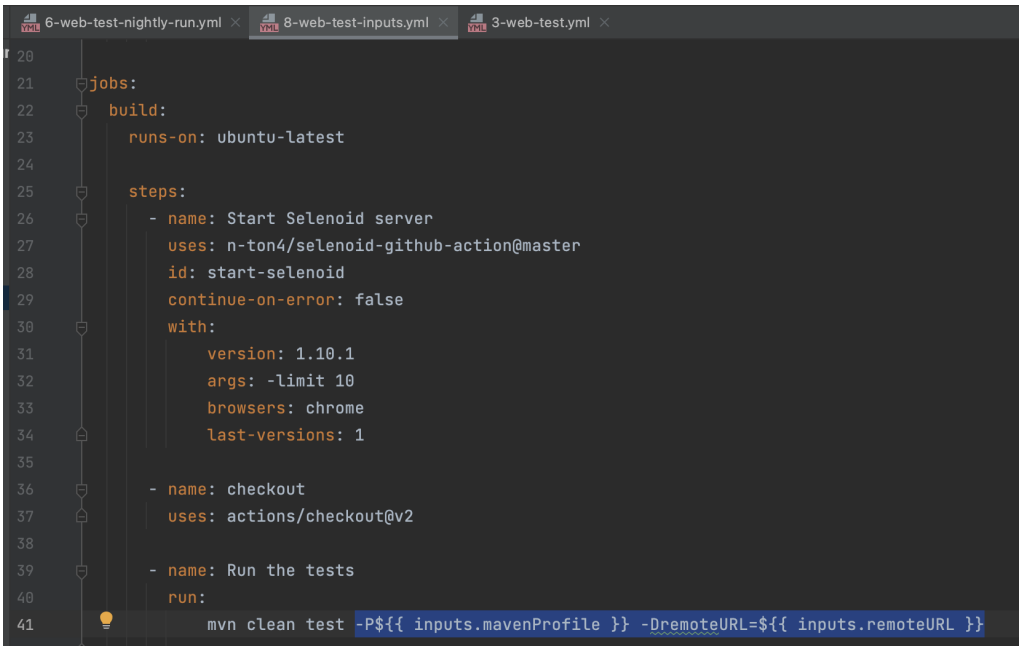
```yaml
name: Provide inputs to workflow

on:
  workflow_dispatch:
    inputs:
      mavenProfile:
        description: 'maven profile - select your choice'
        options:
          - 'web'
          - 'android'
          - 'ios'
          - 'unit-test'
        required: true
        default: 'web'
        type: choice
      remoteURL:
        description: 'selenoid url if hosted outside'
        required: true
        default: 'http://localhost:4444/wd/hub'
```

   1. When the User tries to execute the Workflow manually (Line no. 5 to 19 is under **workflow_dispatch**), he/she has to provide the inputs first
      a. **required**: true (It is a mandatory parameter)
      b. **default:** web

2.

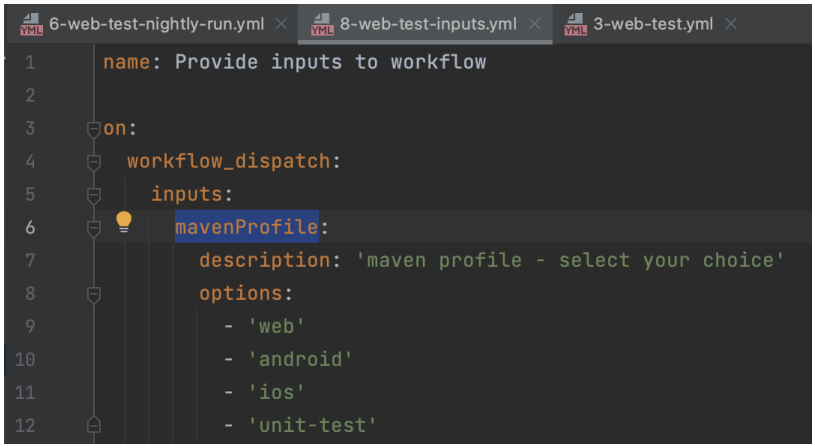


ii.



1. **${{inputs.mavenProfile}}**
    a. This is how we use variables in Github Workflows
2. **mavenProfile**
    a. Customized

b.



—----------

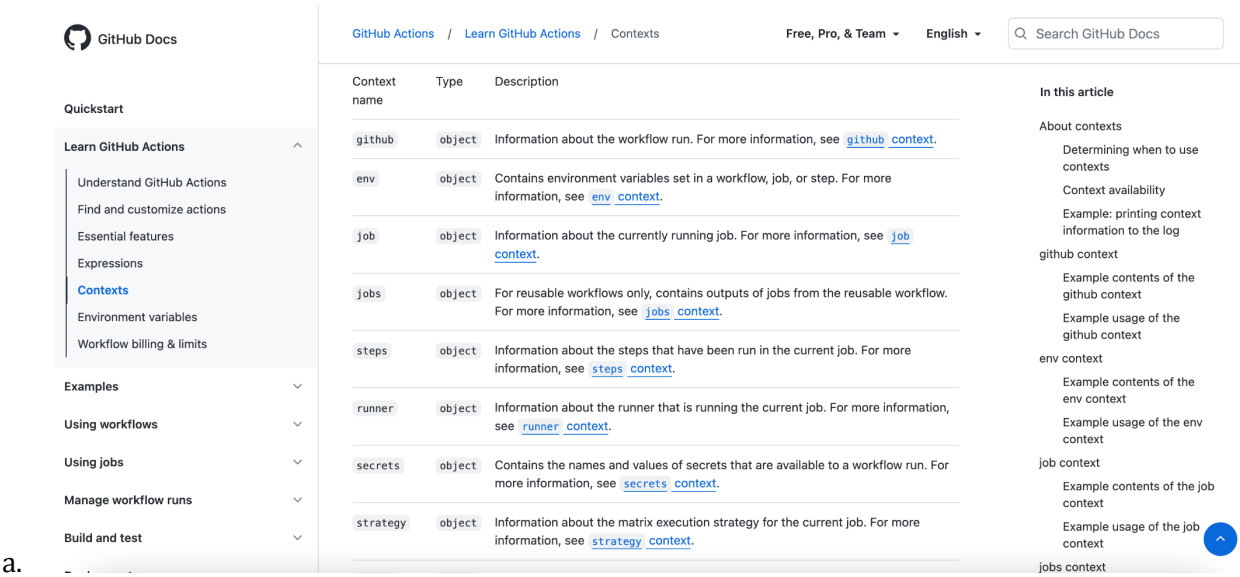
========4_Part 4 | Contexts | Different Contexts in Github Actions | How to use Contexts in GitHub Actions |==========

---

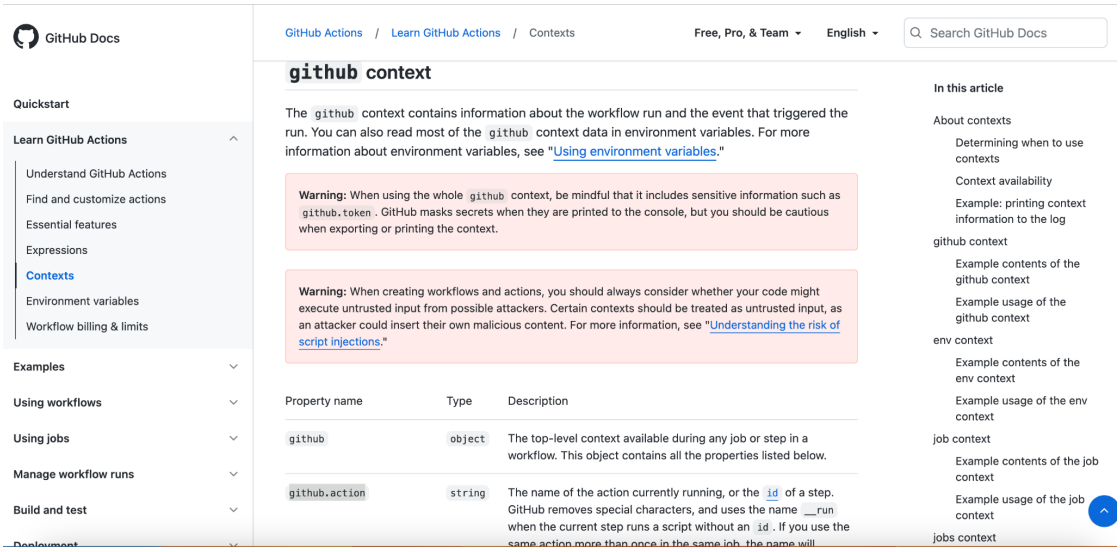
1. **Contexts**
    a. Way to access information about
        i. The workflow runs
            1. Who triggered it?
            2. When triggered?
            3. What is the Event that triggered it?
        ii. Runner environments
        iii. Jobs
        iv. Steps

—----------

1. https://docs.github.com/en/actions/learn-github-actions/contexts

a. 

b. Context - **Github**

i. 

1. github.**action** - The name of the action currently running.
2. github.**action_path** - The path where an action is located.
3. github.**actor** - The username of the user that triggered the initial workflow run.
4. github.**api_url** - The URL of the GitHub REST API.

ii. We can get complete information

1.

```
{
  "token": "***",
  "job": "dump_contexts_to_log",
  "ref": "refs/heads/my_branch",
  "sha": "c27d339ee6075c1f744c5d4b200f7901aad2c369",
  "repository": "octocat/hello-world",
  "repository_owner": "octocat",
  "repositoryUrl": "git://github.com/octocat/hello-world.git",
  "run_id": "1536140711",
  "run_number": "314",
  "retention_days": "90",
  "run_attempt": "1",
  "actor": "octocat",
  "workflow": "Context testing",
  "head_ref": "",
  "base_ref": "",
  "event_name": "push",
  "event": {
    ...
  },
  "server_url": "https://github.com",
  "api_url": "https://api.github.com",
  "graphql_url": "https://api.github.com/graphql",
  "ref_name": "my_branch",
  "ref_protected": false,
  "ref_type": "branch",
  "secret_source": "Actions",
  "workspace": "/home/runner/work/hello-world/hello-world",
  "action": "github_step",
  "event_path": "/home/runner/work/_temp/_github_workflow/event.json",
  "action_repository": "",
  "action_ref": "",
  "path": "/home/runner/work/_temp/_runner_file_commands/add_path_b037e7b5-1c88-48e2-b
  "env": "/home/runner/work/_temp/_runner_file_commands/set_env_b037e7b5-1c88-48e2-bf7
}
```

—----------

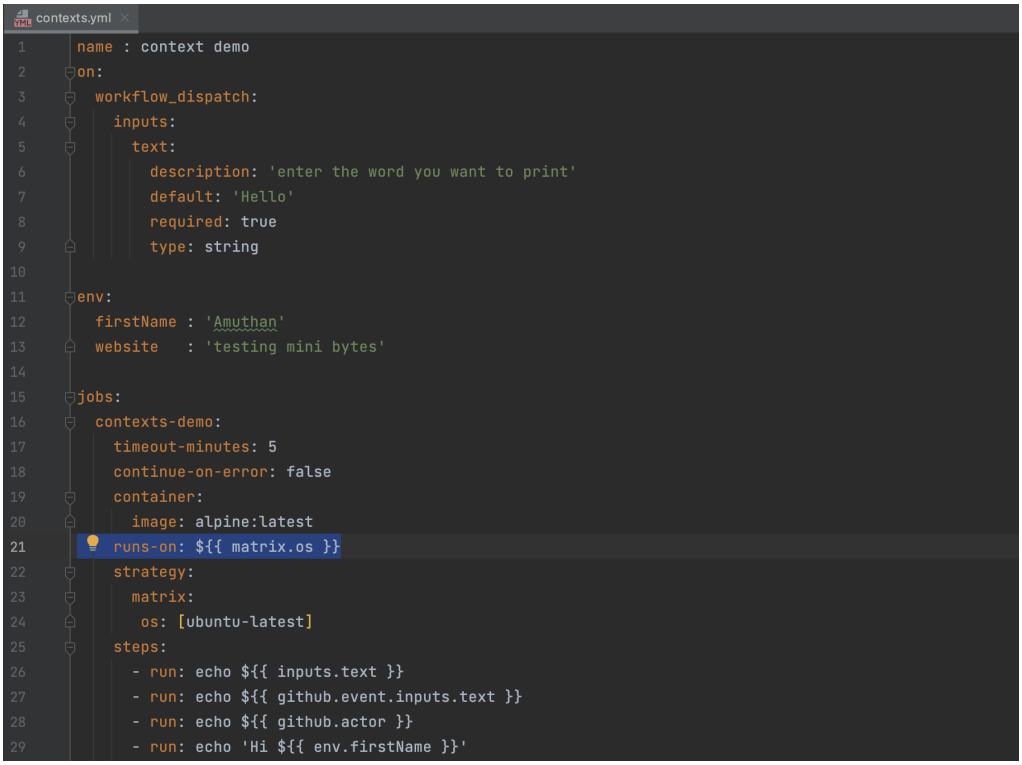
1. **More contexts**
    a. env
    b. job
    c. steps
    d. secrets
    e. runner

—----------

—----------

1. **contexts.yml**

a.



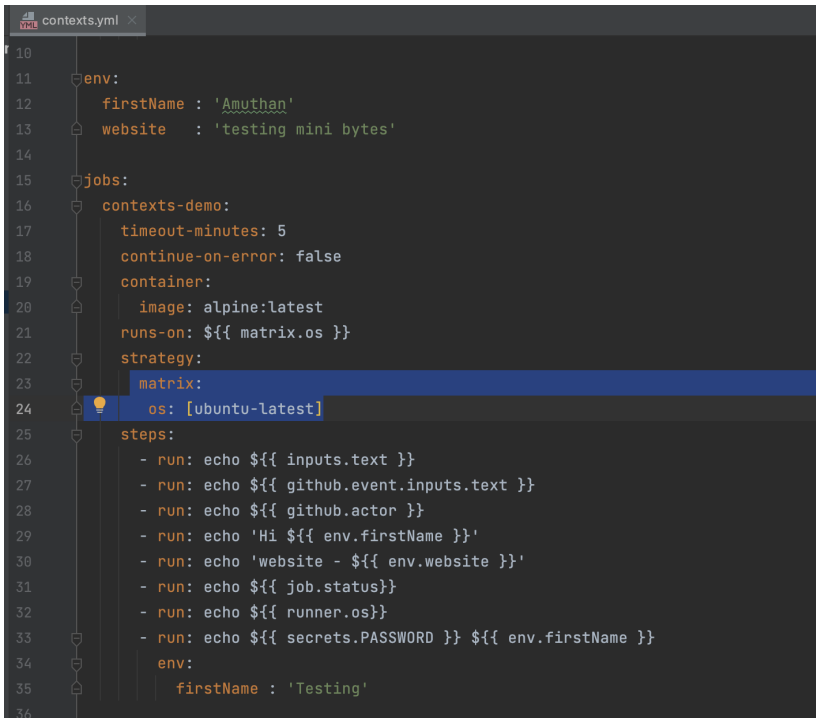
i. This Workflow can only be triggered manually
   1. Line #3 (**workflow_dispatch**)

ii. Also, it will ask the User for input (**text**)
   1. Line #4 (**inputs)**
      a. It is a required parameter
         i. Line #8 (**required:** true**)**

iii. 2 **environments variables** added
   1. Line #12 and 13 (firstName and website)
   2. These env variables will be applicable for all the jobs and steps mentioned in the .yml file
      a. Jobs
         i. context-demo
         ii. Context-demo-2
   3. **Environment variables can be used at the jobs/steps level as well**
   4. If an environment variable with the same name is provided at the job and step level,
      a. Then, the preference will be given to the nearest one
         i. In this case, to the Step level.

—----------

2. contexts.yml



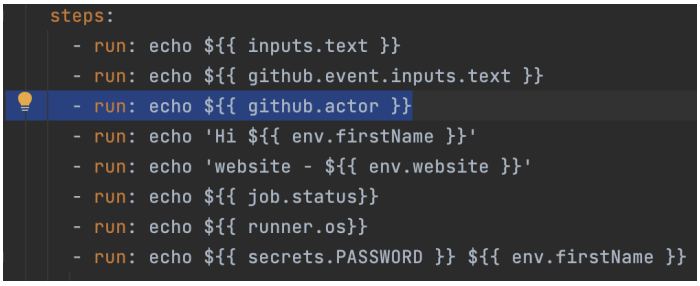
```yaml
env:
  firstName : 'Amuthan'
  website   : 'testing mini bytes'

jobs:
  contexts-demo:
    timeout-minutes: 5
    continue-on-error: false
    container:
      image: alpine:latest
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest]
    steps:
      - run: echo ${{ inputs.text }}
      - run: echo ${{ github.event.inputs.text }}
      - run: echo ${{ github.actor }}
      - run: echo 'Hi ${{ env.firstName }}'
      - run: echo 'website - ${{ env.website }}'
      - run: echo ${{ job.status}}
      - run: echo ${{ runner.os}}
      - run: echo ${{ secrets.PASSWORD }} ${{ env.firstName }}
        env:
          firstName : 'Testing'
```

   a. **Line #21**
      i. OS value is also passed from Matrix
         1. Refer to Lines **#23 and 24**
      ii. **The benefit** of using matrix -
         1. We can parametrize the job
            a. Job is running on one machine
               i. **os: [ubuntu-latest]**
            b. The job may run on multiple machines as well
               i. **os [ubuntu-latest, windows-latest]**

—----------

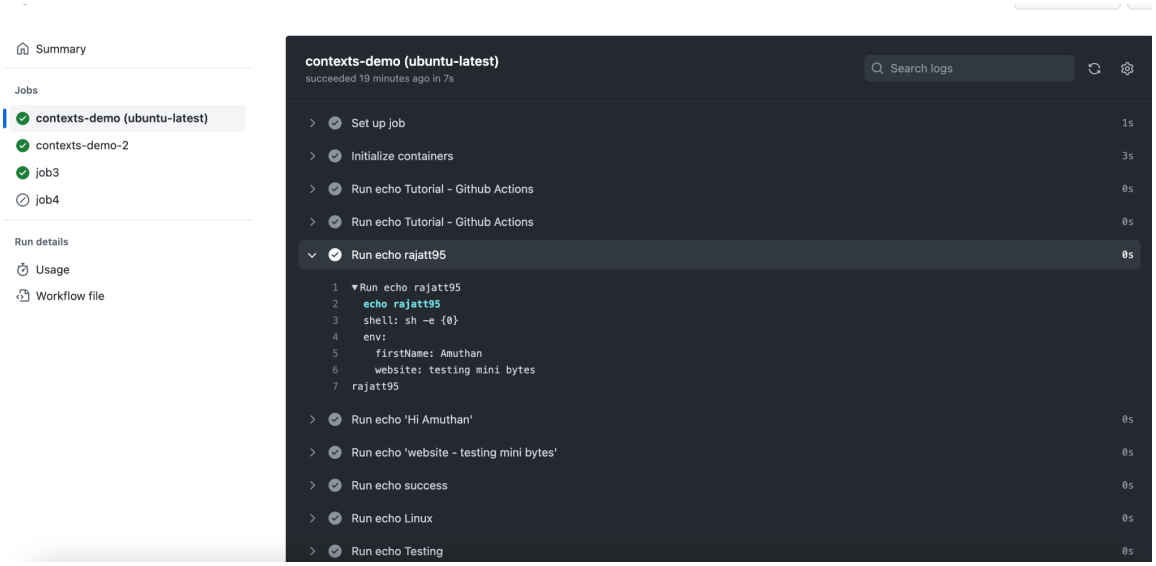
3. **Who has triggered this Workflow?**

   a.



```yaml
steps:
  - run: echo ${{ inputs.text }}
  - run: echo ${{ github.event.inputs.text }}
  - run: echo ${{ github.actor }}
  - run: echo 'Hi ${{ env.firstName }}'
  - run: echo 'website - ${{ env.website }}'
  - run: echo ${{ job.status}}
  - run: echo ${{ runner.os}}
  - run: echo ${{ secrets.PASSWORD }} ${{ env.firstName }}
```

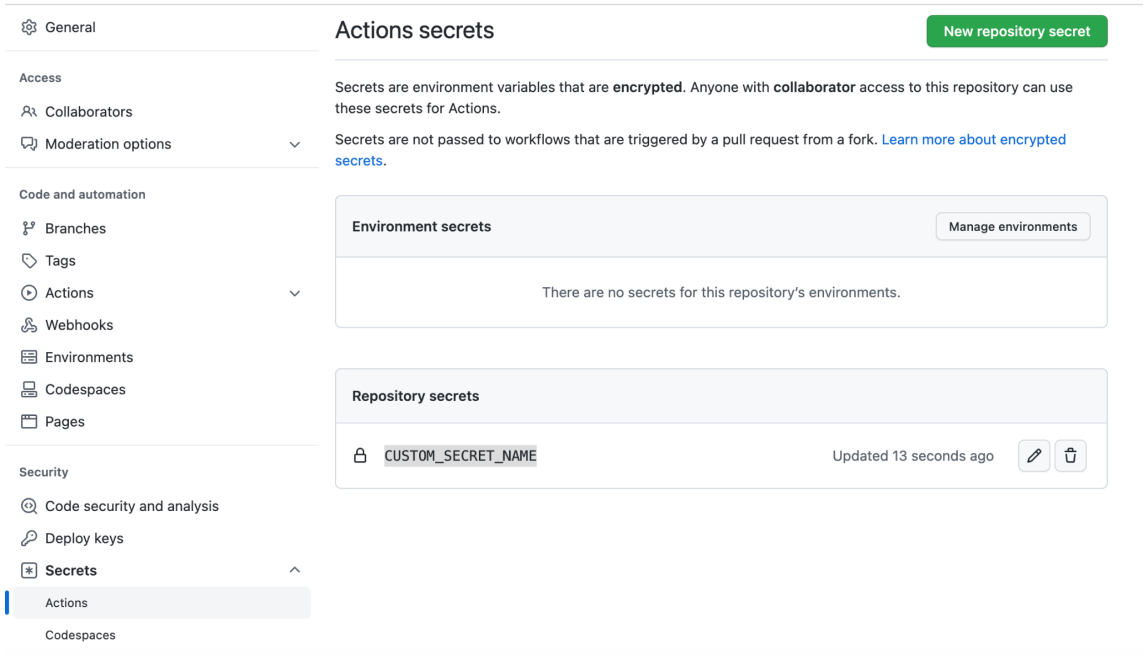
b.



i. We did not provide the Step names in the .yml file
   1. That's why we are getting the commands directly
      a. **Run echo rajatt95**

---

1. **Setup Secrets**
   a. https://github.com/rajatt95/SeleniumAppiumTestsInGithubRunners/settings/secrets/actions
      i. Go to the Github repository
      ii. Click on Secrets (under Security)
      iii. Click on Actions
      iv. Add one **(CUSTOM_SECRET_NAME)**



v.

b.

```yaml
contexts.yml
steps:
  - run: echo ${{ inputs.text }}
  - run: echo ${{ github.event.inputs.text }}
  - run: echo ${{ github.actor }}
  - run: echo 'Hi ${{ env.firstName }}'
  - run: echo 'website - ${{ env.website }}'
  - run: echo ${{ job.status}}
  - run: echo ${{ runner.os}}
#   - run: echo ${{ secrets.PASSWORD }} ${{ env.firstName }}
  - run: echo ${{ secrets.CUSTOM_SECRET_NAME }} ${{ env.firstName }}
    env:
      firstName : 'Testing'
```

---

1. If you want a configuration where **1 job is dependent on another**
   a. We have that setup done here already

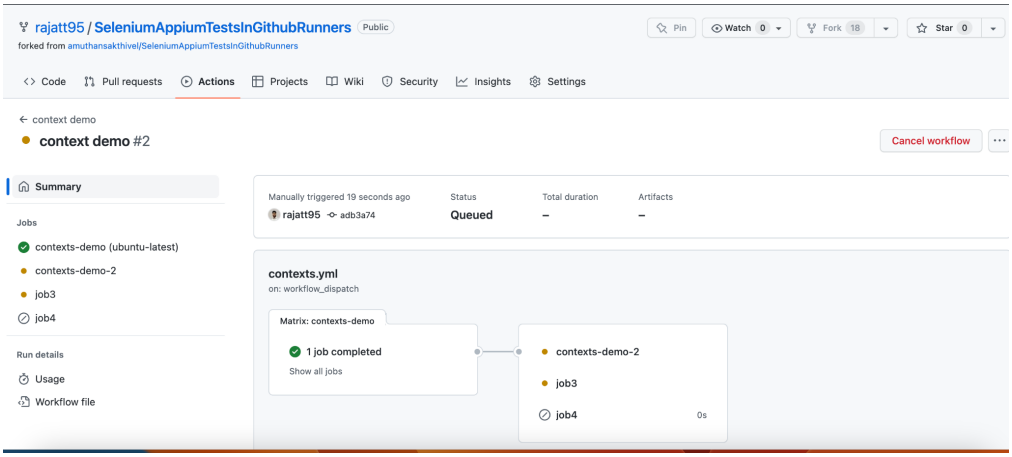
i.

```yaml
contexts-demo-2:
  needs: [contexts-demo]
  runs-on: ubuntu-latest
  steps:
    - run: echo 'contexts-demo-2 running after contexts-demo'
```

1. This job **'context-demo-2'** is dependent on **'context-demo'** job
2. These jobs will not run in parallel

2. Execute the Action
   a. https://github.com/rajatt95/SeleniumAppiumTestsInGithubRunners/actions/workflows/contexts.yml
      i. Click on **Run workflow**

1.



   a. This Workflow will wait for the 1st job to complete
      i. Once 1st job is completed, it will start the 2nd job

---

========5_Part 5 | Jobs in GitHub Actions | Containers in Github Actions | Running Jobs in Parallel |==========

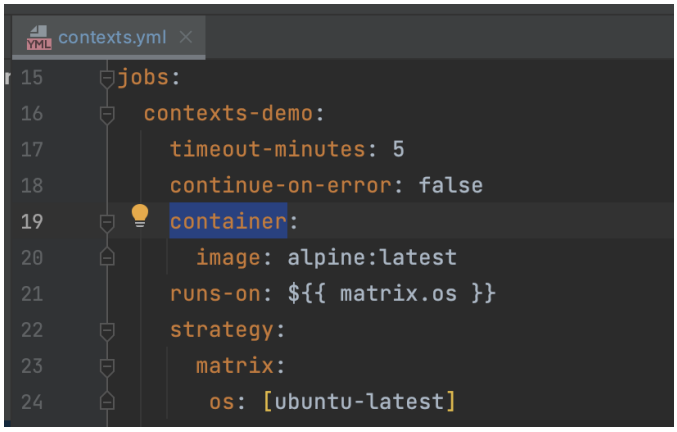
---

1. A Workflow run is made up of one or more jobs, which run in parallel (Default)
    a. To run jobs **sequentially**, you can define dependencies on other jobs using the **needs** keyword.
2. We have the Liberty to run each job in a different host.

—----------

1. Jobs are the building blocks of Workflow
2. **Workflow** is composed of **Jobs**
    a. **Jobs** are composed of multiple **Steps**

—----------

1. 

    a. **timeout-minutes -** Max. allowed time for the job execution
        i. If any job is taking more than the time mentioned here, then, the job will get automatically canceled.
    b. **container -** We can provide the specific version for execution
        i. Example -
            1. I do not want to execute the automated tests on the Chrome browser which comes with ubuntu-latest machine
            2. Then, I can provide the version of the Browser
                a. Also, we do not need to install Docker explicitly
                    i. Docker will be installed by default in the “ubuntu-latest” machine.
                b. We only need to provide the **Image name**
                c. We can define the **Ports** and **Volumes** as well for the Image

—----------

1. The job should **always run**
    a. **Irrespective of the execution result** of the job (contexts-demo)
    b.

```yaml
job3:
  runs-on: ubuntu-latest
  if: ${{ always() }}
  needs: [ contexts-demo ]
  steps:
    - run: echo 'job3'
```

---

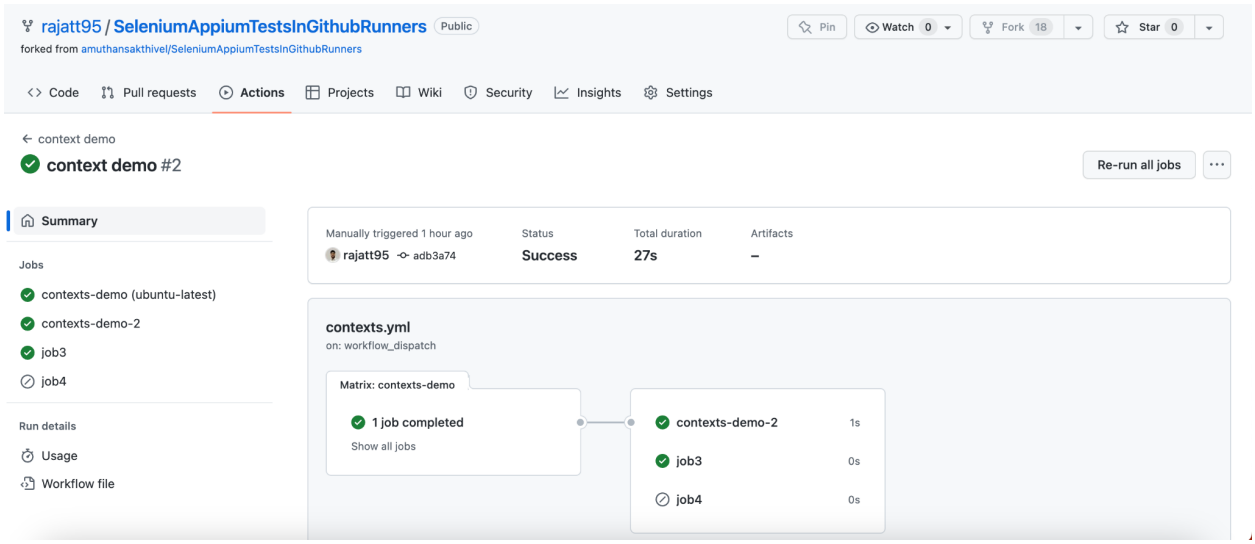
1. Job should **run Only-On-Failure** of job (contexts-demo)
    a.

```yaml
job4:
  runs-on: ubuntu-latest
  if: ${{ failure() }}
  needs: [ contexts-demo ]
  steps:
    - run: echo 'job4'
```

        i. We have one parameter added in context-demo job
            1. **continue-on-error**
            a.

```yaml
jobs:
  contexts-demo:
    timeout-minutes: 5
    continue-on-error: false
    container:
      image: alpine:latest
    runs-on: ${{ matrix.os }}
```

                i. If this is marked as true, then, the result for this job will always be **PASSED**
                    1. It will never be marked as **FAIL**
                        a. That means Job4 will never execute.

2. 

   a. job4 is skipped
      i. Why?
         1. Because the result of the **job** (contexts-demo) is **passed.**
      ii. **job4 will execute only** when the result of the job (contexts-demo) is **failed**.

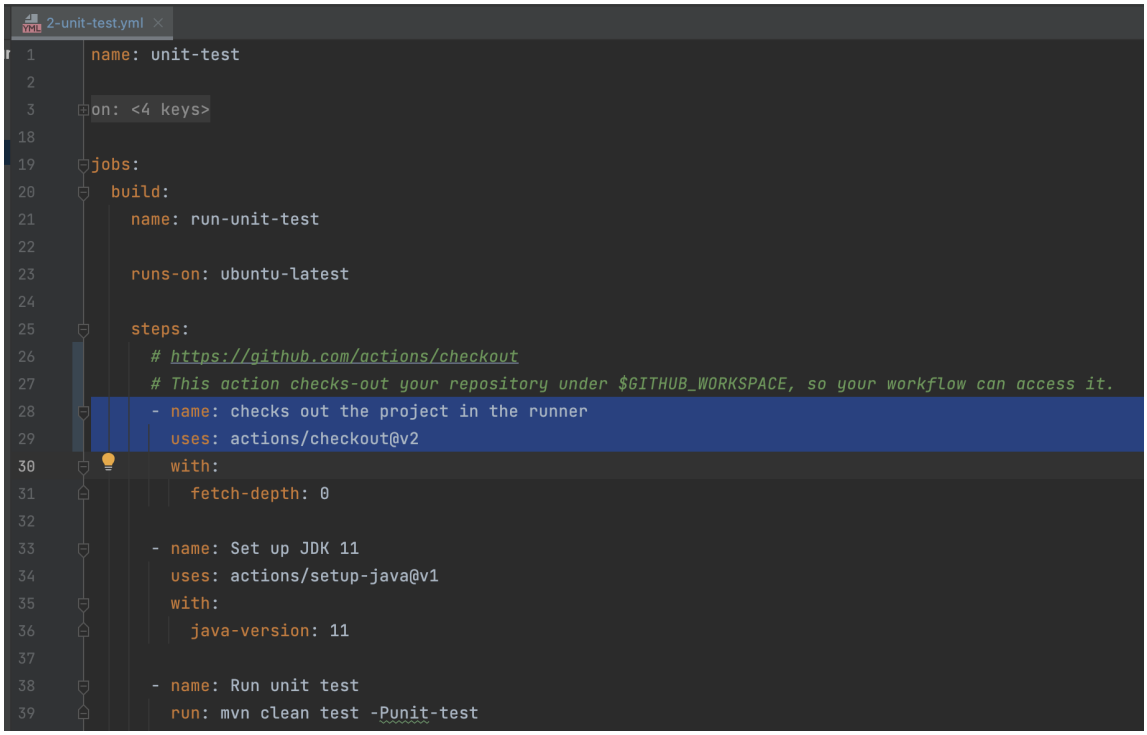
—----------

========6_Part 6 - Steps in Github Actions 🔥 | Step Failure | Timeout for a Step | Github Actions Tutorial |==========

---

1. **Steps**
    a. Jobs are composed of one or more Steps
    b. We can use the already existing Actions
        i. From Github Action Marketplace
    c. Or
        i. We can create customized Actions

—----------

1. 

    a. **Steps**
        i. checks out the project in the runner
        ii. Set up JDK 11
        iii. Run unit test
    b. If the 1st step gets failed due to some issue
        i. Then, 2nd and 3rd steps will get skipped.
        ii. To change this behavior (If you think, this step is not very important)
            1. Add one property with the step
                a. **continue-on-error: true**
    c. Other parameters may be added
        i. timeout-minutes

ii.

```yaml
- name: Run unit test
  # If pom.xml file is located inside runner directory/folder
  working-directory: /runner
  # If you want to execute the command in Powershell
  shell: powershell
  # Execute multiple Maven commands
  run: |
    mvn -v
    mvn clean test -Punit-test
```

—----------

========7_Part 7 | Running web test in Github Actions | Setup Selenoid in Github Actions | Passing secrets |==========

---

1. **Running Web Tests**
    a. Choose the Trigger
    b. Choose the Runner
    c. Setup the Project
    d. Setup Java
    e. Setup Selenoid Container
    f. Run the tests by passing username and password

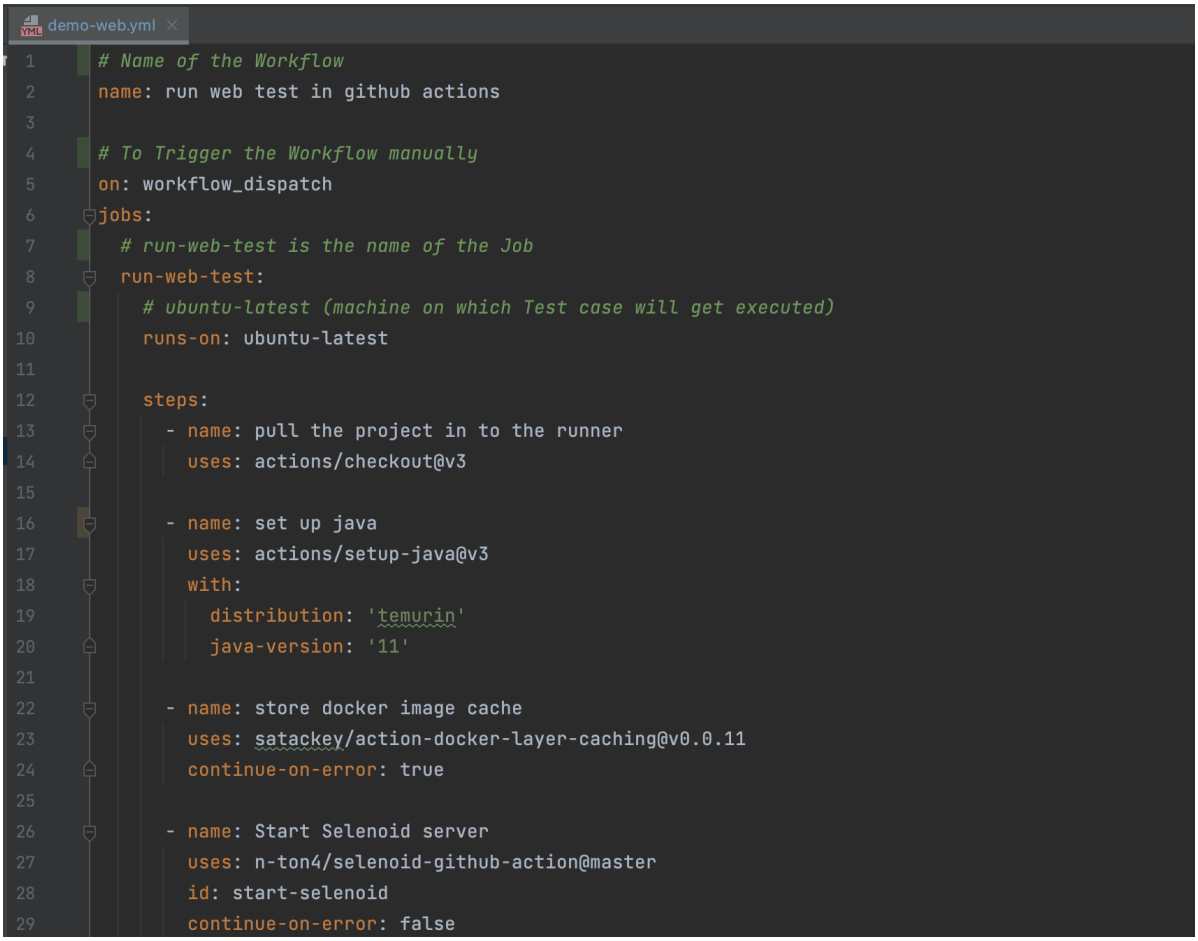
—----------

1.

WebTest.java

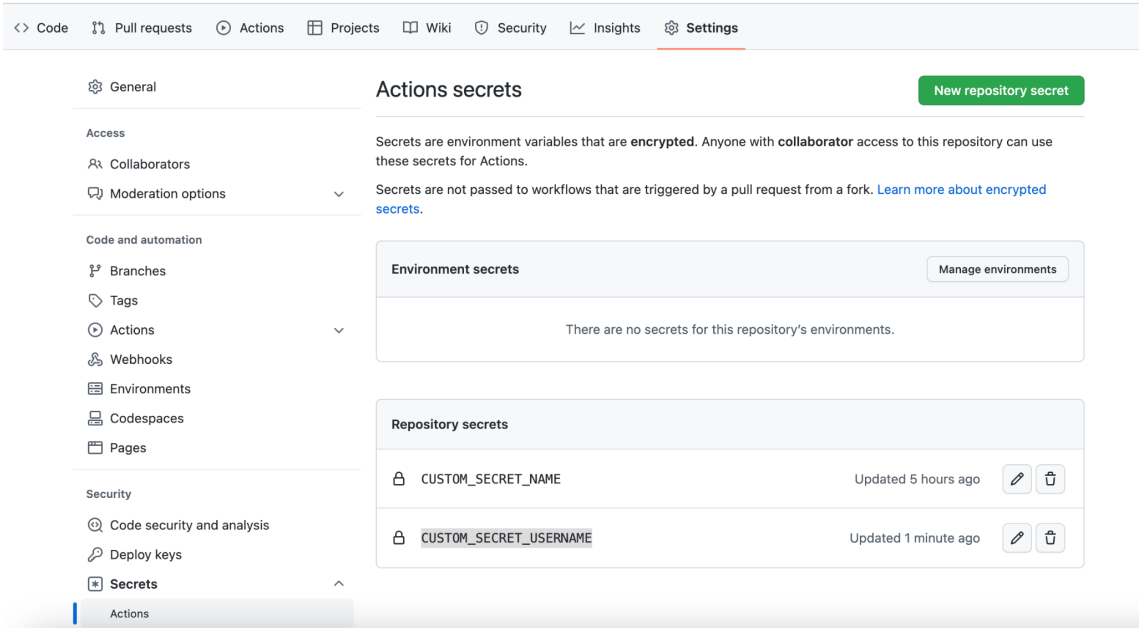
```java
package com.web;

import ...

2 usages  amuthan-sakthivel
public class WebTest {

    amuthan-sakthivel
    @SneakyThrows
    @Test(groups = "web")
    public void testGoogleSearchUsingSelenoid() {
        System.out.println("username :" + System.getProperty("username"));
        System.out.println("password :" + System.getProperty("password"));

        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability( capabilityName: "browserName", value: "chrome");
        capabilities.setCapability( capabilityName: "enableVNC", value: true);
        capabilities.setCapability( capabilityName: "enableVideo", value: false);
        WebDriver driver = new RemoteWebDriver(new URL( spec: "http://localhost:4444/wd/hub"),capabilities);
        driver.get("https://google.co.in");
        Assert.assertEquals(driver.getTitle(), expected: "Google");
        driver.quit();
    }
}
```

—----------

1. **demo-web.yml**
   a. 

2. Add a Secret
   a. Go to Code repository
   b. Go to Settings
   c. Click on Secrets (Under Security)
   d. Click on Actions
   e. New Repository Secret

f.



3.

```
    - name: run the web test
      run: |
#       mvn clean test -Pweb -Dusername=${{secrets.USERNAME}}
        mvn clean test -Pweb -Dusername=${{secrets.CUSTOM_SECRET_USERNAME}}
```

1.

demo-web.yml

```yaml
# Name of the Workflow
name: run web test in github actions

# To Trigger the Workflow manually
on: workflow_dispatch
jobs:
  # run-web-test is the name of the Job
  run-web-test:
    # ubuntu-latest (machine on which Test case will get executed)
    runs-on: ubuntu-latest

    steps:
      - name: pull the project in to the runner
        uses: actions/checkout@v3

      # This is an Optional Step
        # Why? - Because the machine (ubuntu-latest) has many softwares installed already
          # https://github.com/actions/runner-images
      - name: set up java
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '11'
```

a. https://github.com/actions/runner-images
b. Ubuntu-Latest
    i. https://github.com/actions/runner-images/blob/main/images/linux/Ubuntu2204-Readme.md
    ii. **Java**

Version	Vendor	Environment Variable
8.0.352+8	Eclipse Temurin	JAVA_HOME_8_X64
11.0.17+8 (default)	Eclipse Temurin	JAVA_HOME_11_X64
17.0.5+8	Eclipse Temurin	JAVA_HOME_17_X64

========8_Part 8 | Reduce workflow time | Caching Maven | Caching Docker Image Layers | Github Actions==========

---

1. So far,
    a. All the dependency is getting downloaded every time when we are triggering the Workflow (Manual/Automated)
        i. Start Selenoid Server - **32 seconds**
            1. **Why is it taking so long?**
                a. Because every time, we are pulling the latest Docker image for the Selenoid server.
    b. Job time can be drastically reduced by caching the dependencies or docker image layers.

—----------

—----------

1. **Maven Cache Github Action**
    a. https://github.com/marketplace/actions/maven-cache

Usage

The `skjolber/maven-cache-github-action` action must be present **twice** in your build job, with `step: restore` and `step: save` parameters:

```yaml
jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    name: Maven build with caching
    steps:
      - name: Checkout
        uses: actions/checkout@v2
      - name: Set up JDK 1.8
        uses: actions/setup-java@v1
        with:
          java-version: 1.8
      - name: Restore Maven cache
        uses: skjolber/maven-cache-github-action@v1
        with:
          step: restore
      - name: Build hello-world application with Maven
        run: mvn --batch-mode --update-snapshots verify
      - name: Save Maven cache
        uses: skjolber/maven-cache-github-action@v1
        with:
          step: save
```

i.

b.

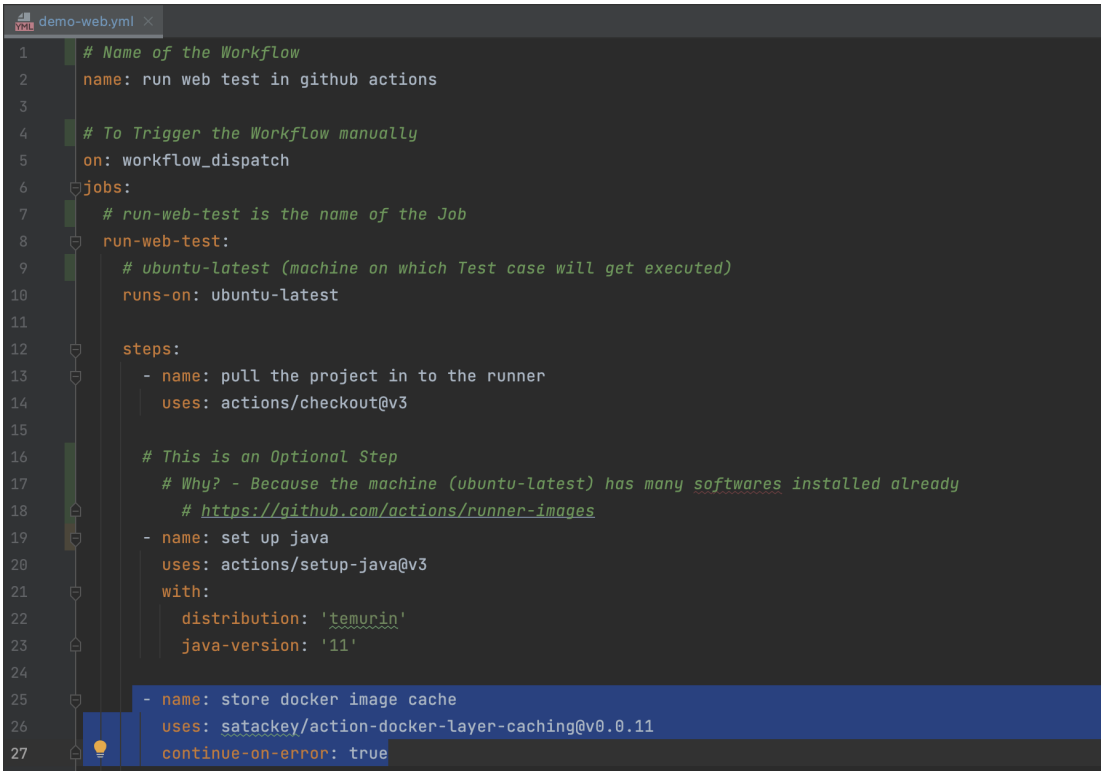
demo-web.yml

```yaml
        # Allowing 10 instances of Chrome Browser
        args: -limit 10
        browsers: chrome
        last-versions: 1

      # Maven Cache Github Action
        # Restoring the Maven Cache
    - name: Restore Maven cache
      uses: skjolber/maven-cache-github-action@v1
      continue-on-error: true
      with:
        step: restore

    - name: run the web test
      run: |
#       mvn clean test -Pweb -Dusername=${{secrets.USERNAME}}
        mvn clean test -Pweb -Dusername=${{secrets.CUSTOM_SECRET_USERNAME}}

      # Save Maven Cache after execution
    - name: Save Maven cache
      uses: skjolber/maven-cache-github-action@v1
      continue-on-error: true
      with:
        step: save
```

i. For the 1st time, it will take time to download and cache it

1. And from 2nd time, it will use the dependencies downloaded in 1st-time

c.



—----------

1. Earlier - Start Selenoid Server - **32 seconds**
2. Now - Start Selenoid Server - **5 seconds**

—----------

========9_Part 9 | No Cloud Providers Needed 🔥 | Run Android Appium Tests inside Github Runner 🔥==========

---

1. We can run the Android tests in Github Runner itself
    a. Setup project
    b. Setup Java
    c. Setup Node
    d. Download Appium
    e. Run Appium
    f. Start Emulator
    g. Run the test

—----------

1.

```java
package com.mobile;

import ...

2 usages  amuthan-sakthivel
public class AndroidTest {

    amuthan-sakthivel
    @SneakyThrows
    @Test
    public void testAppLaunchAndroid() {
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability(MobileCapabilityType.PLATFORM_NAME, Platform.ANDROID);
        capabilities.setCapability(MobileCapabilityType.AUTOMATION_NAME, value: "UIAutomator2");
        capabilities.setCapability("uiautomator2ServerInstallTimeout", 60000);
        capabilities.setCapability(MobileCapabilityType.APP, value: System.getProperty("user.dir")+"/ApiDemos-debug.apk");
        AndroidDriver<AndroidElement> driver = new AndroidDriver<>(new URL( spec: "http://127.0.0.1:4723/wd/hub"),
                capabilities);
        driver.findElementByAccessibilityId( using: "Animation").click();
    }
}
```

—----------

1. https://github.com/actions/runner-images/blob/main/images/macos/macos-12-Readme.md

—----------

1.

demo-android.yml

```yaml
# Name of the Workflow
name: run android tests in github runner

on:
  push:

  # To Trigger the Workflow manually
  workflow_dispatch:

jobs:
  # run-mobile-tests is the name of the Job
  run-mobile-tests:
    # macos-latest (machine on which Test case will get executed)
    runs-on: macos-latest

    steps:
      # This is an Optional Step
       # Why? - Because the machine (macos-latest) has many softwares installed already
        # Java 8 is the default version
        # https://github.com/actions/runner-images
      - name: setup java
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '11'

      - name: pull the project in to the runner
        uses: actions/checkout@v3
```

2.

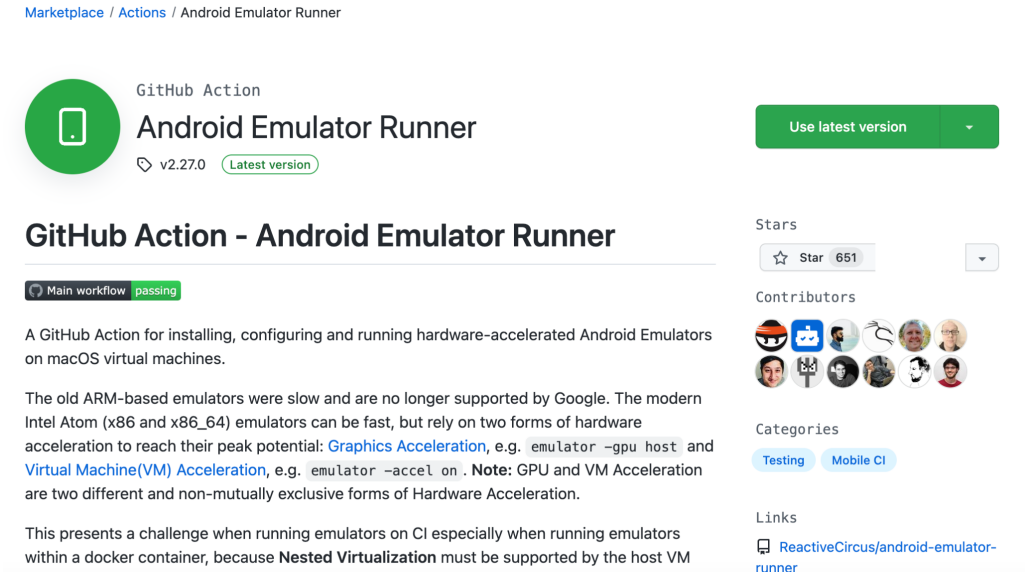
demo-android.yml

```yaml
      - name: pull the project in to the runner
        uses: actions/checkout@v3

      - name: setup appium
        run: |
          npm install -g appium@v1.22
          appium -v
        # appium &>/dev/null & - This will keep Appium server in listening mode
          appium &>/dev/null &

      - name: run appium android tests
        uses: reactivecircus/android-emulator-runner@v2
        with:
          api-level: 29
          script: mvn clean test -Pandroid
```

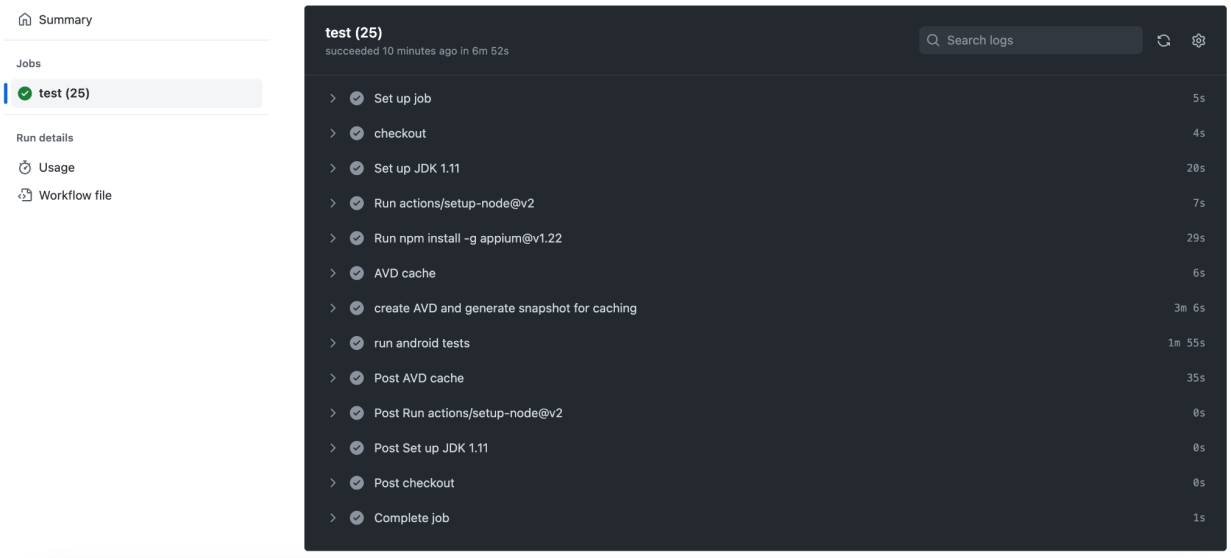
—----------

1. https://github.com/marketplace/actions/android-emulator-runner

a.



1.



========10_Part 10 | Run iOS Appium Tests in Github Runner 🔥 | No cloud providers needed |==========

---

1. We can run the iOS tests in the Github Runner itself
    a. Setup Java
    b. Setup Node
    c. Download Appium
    d. Run Appium
    e. Start Emulator
    f. Run the test

—----------

1.

```java
package com.mobile;

import ...

2 usages  amuthan-sakthivel
public class IosTest {

    amuthan-sakthivel
    @SneakyThrows
    @Test
    public void testAppLaunchIOS() {
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability(MobileCapabilityType.PLATFORM_NAME, Platform.IOS);
        capabilities.setCapability(MobileCapabilityType.AUTOMATION_NAME, value: "XCUITest");
        capabilities.setCapability( capabilityName: "isHeadless", value: true);
        capabilities.setCapability(MobileCapabilityType.DEVICE_NAME, value: "iPod touch (7th generation)");
        //capabilities.setCapability(MobileCapabilityType.APP, System.getProperty("user.dir")+"/DailyCheck.zip");
        capabilities.setCapability(MobileCapabilityType.APP, value: System.getProperty("user.dir")+"/TestApp-iphonesimulator.app");
        AndroidDriver<AndroidElement> driver = new AndroidDriver<>(new URL( spec: "http://127.0.0.1:4723/wd/hub"),
                capabilities);
    }
}
```

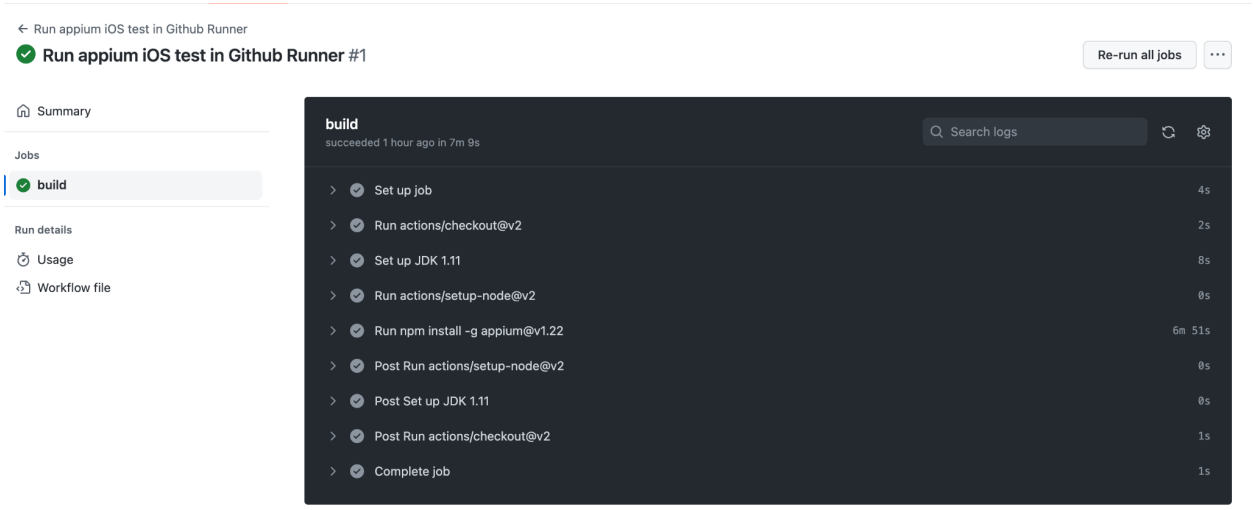
2.

demo-ios.yml

```yaml
name: run ios tests in github runner

on: push

jobs:
  run-ios-tests:
    runs-on: macos-latest

    steps:
      - name: setup java
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '11'

      - name: pull the project in to the runner
        uses: actions/checkout@v3

      - name: setup appium
        run: |
          npm install -g appium@v1.22
          appium -v
          appium &>/dev/null &

      - name: run appium ios tests
        run: mvn clean test -Pios
```

—----------

1.



—----------

========11_Part 11 | Trigger Automated Test workflow in one repo from Development Workflow in another repo==========

---

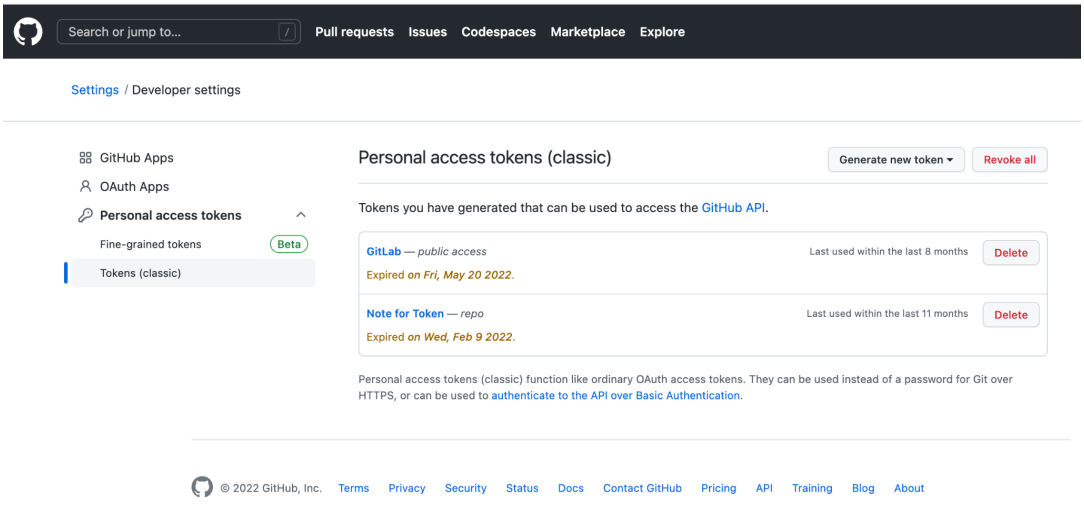
1. Triggering a Workflow in one repository from another repository using Repository dispatch
    a. Example
        i. We want to **run our automated tests** when there is a **successful merge in the development branch**

—----------

1. https://github.com/marketplace/actions/repository-dispatch

—----------

1. **To generate Personal Token**
    a. Go to Github Profile
    b. Go to Settings
    c. Go to Developer Settings
    d. Go to Personal Access Tokens
    e. Click on **Generate new Token**
        i. https://github.com/settings/tokens



ii.

—----------

1. trigger-test.yml

```yaml
#development repository
name: trigger tests in another repository
on: workflow_dispatch
jobs:
  trigger-test:
    runs-on: ubuntu-latest
    #build container #deploy container
    steps:
      - name: trigger the automated test in another repo
        uses: peter-evans/repository-dispatch@v2
        with:
          token: ${{ secrets.REPO_ACCESS_TOKEN }}
          repository: amuthansakthivel/SeleniumAppiumTestsInGithubRunners
          event-type: my-event
          client-payload: '{"github": ${{ toJson(github) }}}'
```

   a. **trigger-test.yml file**
      i. This is for the development code.
         1. Code push to develop branch -> Merge success will lead to the execution of automated tests (another repository)

2. repository-dispatch.yml

```yaml
name: run the workflow

on:
  repository_dispatch:
    types: [my-event]

jobs:
  build:
    name: run-workflow
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2
        with:
          fetch-depth: 0

      - name: print trigger details
        run: |
          echo ${{ github.event.client_payload.github.action }}
          echo ${{ github.event.client_payload.github.actor }}
```

   a. **repository-dispatch.yml**
      i. This file is used to execute the automated tests.

—----------

==========

1. **More documents:**
    a. https://github.com/rajatt95/Documents

2. **Learnings from Tutor (Code Repository):**
    a. **This course**
        i. https://github.com/stars/rajatt95/lists/youtube-tmb-github-actions
    b. **Other course(s):**
        i. https://github.com/stars/rajatt95/lists/testing-mini-bytes-amuthan
    c.
3. **To connect:**
    a. https://www.linkedin.com/in/rajat-v-3b0685128/
    b. https://github.com/rajatt95
    c. https://rajatt95.github.io/

==========