# DevSecOps Internship at AMD

*Internship report submitted in partial fulfilment of the requirements for the degree of B.Tech. (and M.Tech (for DD))*

*by*

Sreepathy Jayanand
(Roll No: CED17I038)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY,
DESIGN AND MANUFACTURING, KANCHEEPURAM

October 2021

# Certificate

I, **Sreepathy Jayanand**, with Roll No: **CED17I038** hereby declare that the material presented in the Internship Report titled **DevSecOps Internship at AMD** represents original work carried out by me in the **Department of Computer Science and Engineering** at the **Indian Institute of Information Technology, Design and Manufacturing, Kancheepuram** during the year **2021**. With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: October 27 2021 Student's Signature: Sreepathy Jayanand

In my capacity as supervisor of the above-mentioned work, I certify that the work presented in this Report is carried out under my supervision, and is worthy of consideration for the requirements of internship work during the period May 2021 to October 2021.

Advisor's Name: C. Y. Hiew Advisor's Signature: C. Y. Hiew

# *Abstract*

This report documents the internship work I have carried out at AMD India Ltd. under the DevSecOps department. This report describes the following tasks completed during my internship at AMD India Ltd.

1. Creation of a Github API server, an internal API server that serves an internal service portal for obtaining and modifying various data from Github.

2. Creation of a GitHub runner agent, helping other organizations run work-flows and custom jobs in the cloud.

This report documents the work done to develop an API for the AMD - IT organization that queries and modifies data for the GitHub enterprise solution set up within AMD. The final output is a an API server ready to be run in the cloud as a container.

This report also presents the work done in developing a GitHub runner agent that enables other organizations within AMD to run workflows such as running Github actions, running custom workflows, custom jobs and other operations on the source code stored in Github all in the cloud.

# *Acknowledgements*

I would like to take this opportunity to express my gratitude to everyone who helped me during my internship. I am grateful for their constant support, constructive criticism and friendly advice. I express my heartfelt gratitude to AMD India Ltd. for giving me the opportunity to be a part of such an amazing organization and contributing to their growth at the same time.

I express my sincere thanks to Mr. Sharafudden Hassan, Principal Member of Technical Staff AMD, who despite being extremely busy with his responsibilities took the time to listen, direct and keep me on the right track during my internship.

I would also like to express my gratitude to Mr. Praveen Kumar Pagudala, Senior Manager AMD for his constant guidance and support during these tough times, helping me complete the internship without any hassle.

Last but certainly not least, I would like to thank my mentor, Mr. C. Y. Hiew, Senior Member of Technical staff AMD, for taking his time to give me valuable insights, suggesting possible improvements and for being always available for any doubts.

# Contents

# List of Figures

# Abbreviations

| | |
|---|---|
| **API** | **A**pplication **P**rogramming **I**nterface |
| **PAT** | **P**ersonal **A**ccess **T**oken |
| **AKS** | **A**zure **K**ubernetes **S**ervices |
| **ACR** | **A**zure **K**ubernetes **R**egistry |

# Chapter 1

# Introduction

## 1.1 Background

### 1.1.1 API Server

#### 1.1.1.1 What is an API server?

An Application Programming Interface or API is a software intermediary that allows for the communication between 2 applications or endpoints. APIs can be categorized into 4, namely

- Public APIs
- Internal APIs
- Partner APIs
- Composite APIs

An API server is an application that enables another application to exchange information with the server using the API through the internet.

#### 1.1.1.2 How does an API sever work?

Almost always an API sever is up and running either in an on - premise system or in the cloud, listening for requests. When it receives a request it performs the corresponding action, whether it is fetching data, modifying data or even deleting data and sends back a response according to the request.

### 1.1.2 Runner

#### 1.1.2.1 What is a Runner?

A runner is a virtual machine with a set of custom software, hosted either by a service provider or on our own machines to help execute various custom work-flows which we desire. The main use of a runner is that it can be tied to a pipeline, so that work-flows can be automatically triggered without manual interference. GitHub provides runner software which can be loaded on a machine, and can be used to run GitHub actions, custom work-flows and custom jobs.

#### 1.1.2.2 How does a Runner work?

A runner is typically tied with a pipeline, and requests are given to the runner based on the current action to be performed within the pipeline. The linking of the pipeline with a runner is usually performed using access tokens.

## 1.2 Motivation

### 1.2.1 Why use an API server?

An internal API can be always use to abstract away unnecessary details. Within a large enterprise, it is important that an organization only gets the minimum information needed to function properly. Using APIs can help us achieve this. Public APIs such as Google maps API, Covid API etc have helped us gain access useful information.

### 1.2.2 Why use a Runner?

A runner is important to run tasks without human intervention. Automation is a vital part in improving the reliability and efficiency in the computing world and runners can help achieve that.

## 1.3 Objectives of the work

### 1.3.1 GitHub API Server

Create an internal API server that can serve requests from ServiceNow(An internal ticketing application), with capabilites to retrieve information from the Github database, change information in the Github database and also delete information in the Github database for the AMD enterprise account and send back appropriate responses back to ServiceNow.

### 1.3.2 GitHub runner

Create a self - hosted Github runner with capabilities as requested by various other internal organizations and host it in the cloud, which can help them run custom work-flows, jobs and also Github actions.

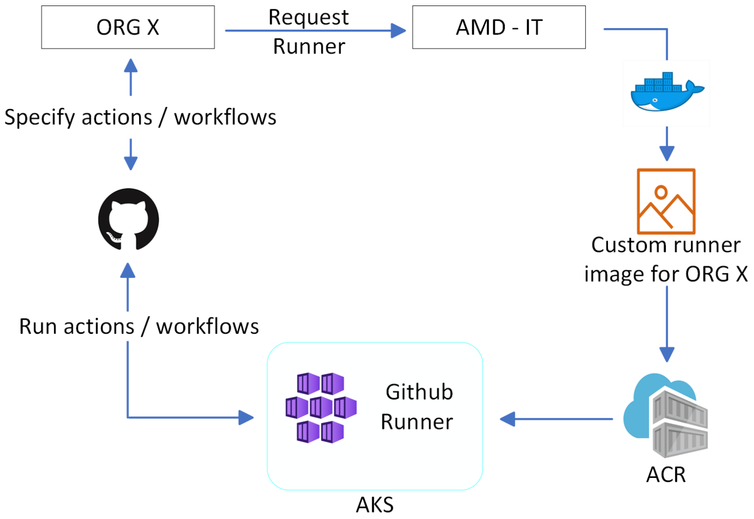


FIGURE 1.1: Github Runner Overview

# Chapter 2

# Methodology

## 2.1 Github API Server

In order to make an API, it is very important to have a clear idea on what the API has to do. What the inputs are, what the outputs are, what are the formats of the output, what is the maximum expected latency possible, required security measures etc. The Github API server required a total of 19 functions, each of which had a different kinds of inputs and outputs. Some of them required a JSON array output while some others required a boolean. In our case these were the objectives to be accomplished.

- Inputs - Depending on the function, the inputs belonged to one / many from the list of Github organization name, repository name, permission type, collaborator name etc
- Outputs - The outputs were mainly of 2 types, JSON array for GET queries obtained back from Github or boolean indicating success / failure of update / delete operations.
- Maximum latency possible - We were given a barrier of a maximum of 3 seconds.
- Secure storage for credentials - Since this is an API which has to communicate with Github, we had to use personal access tokens. This has to be stored in a secure location so that it can be obtained whenever this API server starts, i.e during run time.

Once we have obtained all these information we can start to decide the implementation details of our API server.

### 2.1.1 API Framework

The most suitable framework, which suits our needs was the spring boot framework in JAVA. We decided to go with a Maven build.

### 2.1.2 Query framework

Some of the functionalities of the API involved getting information from Github. Github provides support for GraphQL API queries for obtaining information from Github. We decided to go with this instead of tradition HTTP Get query as it is the new norm supported by Github for GET requests and in a stretch we can get upto 100 items for a query.

### 2.1.3 Security

The connection to be established with Github was authenticated using Personal Access Tokens or PATs. It is not advisable to store PATs in the source code as it is a security risk. We decided to go with Azure key vaults to obtain the secrets during run time from Azure which is a safer practice.

### 2.1.4 Logging

We decided to go with log4j v2, as this is a simple logger for Java builds and can also support rolling log updates with a size cap on each of the log files and can automatically compress the older files to create archives.

### 2.1.5 Update framework

For all write related operations such as updates, deletes etc we decided to go with traditional HTTP PUT, PATCH, POST, DELETE methods.

## 2.2 Github Runner

Since the runner is made for another internal organization, they provided us information on the packages that they require on their runner. Now we had to create an image in which the github runner resides and host it in the cloud so that the organization can access it to provide it with workflows which it has to run.

### 2.2.1 Dockerfile

First we have to create the Dockerfile which contains all the commands needed to build the final image, including the base image, package installation, environment setting's modification etc.

### 2.2.2 Building Image

We decided to use Docker to build the image as it is the default norm in this domain.

### 2.2.3 Storing Image

We decided to use Azure container registry to store the images built so that they are always available for deployments to AKS.

### 2.2.4 Deployment

We decided to go with Azure Kubernetes Services or AKS, for our deployments.

# Chapter 3

# Work Done

## 3.1 Github API Server

The entire API server is build in JAVA. In this section we'll go through all the work done categorized into various steps.

### 3.1.1 Maven build and dependencies

As stated before, we decided to go with a Maven build for the Github API server. In a Maven build all dependencies / packages are defined within the pom.xml file. Here are a few of the major dependencies.

- org.springframework.boot - Java Spring Framework is a popular, open source framework for creating standalone applications that run on the Java Virtual Machine [1]. It also has support log4j v2, a logging tool as well as support for other useful plugins.
- com.graphql-java - GraphQL is a query language for APIs that allows clients to request limited data they need, making it possible for clients to gather data in a limited number of requests [2].

- org.apache.httpcomponents - HTTP components package gives us capabilities of using HTTP requests such as PUT, PATCH, DELETE and POST. Instead of GET we use GraphQL.
- com.azure - The Azure secrets package is used here and it has provision for storing secrets and obtaining them whenever needed from Azure cloud, instead of storing it in the source code itself.

These were some of the basic build decisions that we made. Here is a snippet of a dependency from the pom.xml file which contains all of the dependencies.

```
<dependency>
      <groupId>com.graphql-java</groupId>
      <artifactId>graphql-java</artifactId>
      <version>16.2</version>
</dependency>
```

### 3.1.2 Utility functions

- getPAT() - This is a utility function that fetches the secret PAT used to get information from Github. The secret is stored in Azure secrets. We use the SecretClientBuilder() function from the Azure keyvault class and parse in the Azure url where the key is stored as well as the client secret credentials stored locally to obtain the PAT from Azure so that we can now interact with Github.
- getSAML() - Another utility function that fetches the mapping from Github between the NTID (Unique within AMD) and the name of a person as well as the mapping between the Github unique login id with the NTID and name. This function is implemented by interacting with Github using the GraphQL API provided by Github. The GraphQL query to get the SAML is created and sent to api.github.com, and a JSON object containing all the details is sent back as the response.

### 3.1.3 Important variables

- PAT - The personal access token returned from getPAT() function

- LOG - The instance of logger class which contains member functions to help in the logging process.
- MAP-NTID-TO-LOGIN - A Hash Map, mapping NTID to Github login id. This is obtained from the getSAML() function.

These are a few of the major variables.

### 3.1.4 Graphql execution function - exeGraphql()

This is a utility function, which takes the graphql query as input and constructs an HTTP post request and sends it using the Github's API. The response contains the output for the query. The function then takes the output and formats in JSON and returns it.

Pseudo code -

```
JSONObject exeGraphql(JSONObject jsonObj) {
        request = CreateHTTPPostRequest(jsonObj);
        response = execute(request);
        json_formatted_response = format_into_json (response);
        return json_formatted_response;

}
```

### 3.1.5 GET data from Github

Using GraphQL we can build queries which we wish to address and send it to the Github API, and we get a response containing the required result. In one response a maximum of 100 items are retrieved. In case our response is bigger we have to create multiple requests and obtain 100 at a time.

Pseudo Code:

```
ans = {}
while (next_page_exists) {
        page = fetch_page();
        ans = ans + items_of_page(page);
        next_page_exists = page.does_next_page_exists();
}
```

Let's look at a mapping "/SNOW/org/ORG/repo/name/getadmin". Here we are supposed to fetch the admins of a repo with name - "name" and inside the organization "ORG". Here "ORG" and "name" are PathVariables and is sent by SNOW, the organization for which the API is being built.

Let's go through this function section by section.

```
@GetMapping("/SNOW/org/{ORG}/repo/{name}/getadmin")
    public JSONArray list_admins(@PathVariable String ORG, @PathVariable String
    name) {
        Instant start = Instant.now();
        LOG.info("Mapping - http://localhost:8080/SNOW/org/" + ORG + "/repo/" +
    name +
        "/getadmin" );
        JSONObject graphqlQueryObj = new JSONObject();
        JSONArray array = new JSONArray();
        String next_page_exists = "true";
        String after_suffix = "";

        while (next_page_exists.equals("true")) {
            graphqlQueryObj.put("query", "{\n" +
            "  organization(login: \"" + ORG + "\") {\n" +
            "    repository(name: \"" + name + "\") {\n" +
            "      collaborators(first: 100" + after_suffix + ") {\n" +
            "        totalCount\n" +
            "        edges {\n" +
            "          permission\n" +
            "          node {\n" +
            "            login\n" +
            "            name\n" +
            "          }\n" +
            "        }\n" +
            "        pageInfo {\n" +
            "          endCursor\n" +
            "          hasNextPage\n" +
            "        }\n" +
            "      }\n" +
            "    }\n" +
            "  }\n" +
            "}");
            JSONObject obj3 = new JSONObject();
            JSONArray temp_array = new JSONArray();
            obj3 = exeGrapql(graphqlQueryObj);
            LOG.debug("Return from graphql - " + obj3);
            obj3 = (JSONObject) obj3.get("data");
            obj3 = (JSONObject) obj3.get("organization");
            obj3 = (JSONObject) obj3.get("repository");
            obj3 = (JSONObject) obj3.get("collaborators");

            temp_array = (JSONArray) obj3.get("edges");
            if (temp_array.size() == 0)
                break;
            for (int i = 0; i < temp_array.size(); i++) {
                JSONObject temp_obj = new JSONObject();
                temp_obj = (JSONObject) temp_array.get(i);
                String permission_status = temp_obj.get("permission").toString();
                if (permission_status.equals("ADMIN")) {
                    temp_obj = (JSONObject) temp_obj.get("node");
                    String temp_string = temp_obj.get("login").toString();
                    array.add(temp_string);
                }
            }
            obj3 = (JSONObject) obj3.get("pageInfo");
            next_page_exists = obj3.get("hasNextPage").toString();
            String end_pointer = obj3.get("endCursor").toString();
            after_suffix = ", after: \"" + end_pointer + "\"";

        }
        Instant end = Instant.now();
        LOG.info("Time taken for " + "Mapping - http://localhost:8080/SNOW/org/" +
    ORG +
         "/repo/" + name + "/getadmin = " + Duration.between(start, end));
        return array;
    }
```

We start this mapping by logging all the details as sent by SNOW. We declare the variables, to store the final JSON array, the loop iterator checker and the ”after-suffix” which takes care of the storing the pointer of the last element that we accessed so that we can start querying from the next element onward.

Now inside the while loop we construct the graphQL query, depending on what we want to fetch from Github. In this case it is the list of admins, within a repo, within an organization. We use the ”after suffix” variable to store the last accessed element's pointer. Then we use the exeGraphql command to obtain the results of the Graphql query contructed, which returns the JSON object sent back from Github.

Now we go through the JSON object to get the list of admins and push it into ”array” which is what we return from this mapping.

There are 13 such functions, each reponsible for fetching various different information from Github all using the same principle.

### 3.1.6 Update Information functions

We saw how we get information using Graphql queries. Now we see how we update information. This is done in the traditional HTTP POST, PUT, DELETE, PATCH request. Let’s take a look at one of the functions used to change the permission status of an outside collaborator for a repo within an organization.

Pseudo code:

```
boolean change_permission(ORG, name, collaborator-name, permission-status) {
        api_url = "https://api.github.com/repos/" + ORG + "/" + name + "/
    collaborators/" +                                     collaborator_name;
        response = execute_HTTP_Put(api_url, permissions);
        return response.status;

}
```

Here the execute-HTTP-PUT function creates an HTTP PUT request with a JSON body which includes the permission-status and the api-url so that Github can update the new permission status of the collaborator.

Now let’s go through this function section by section.

```
 @GetMapping("/SNOW/org/{ORG}/repo/{name}/oc/add/{collaborator_name}/{
    permission_status}")
    public boolean outside_collaborator_read(@PathVariable String ORG,
    @PathVariable String name, @PathVariable String collaborator_name,
    @PathVariable String permission_status) {
        Instant start = Instant.now();
        LOG.info("Mapping - http://localhost:8080/SNOW/org/" + ORG + "/repo/" +
    name + "/oc/add/" + collaborator_name + "/" + permission_status);
        boolean response_bool = false;
        String api_url = "https://api.github.com/repos/" + ORG + "/" + name + "/
    collaborators/" + collaborator_name;
        String permission_string;
        if (permission_status.equals("read"))
            permission_string = "pull";
        else
            permission_string = "push";
        String json_form = "{\"permission\":\"" + permission_string + "\"}";
        CloseableHttpResponse response = Execute_HTTP_PUT(api_url, json_form);
        if (response.getStatusLine().getStatusCode() == 201 || response.
    getStatusLine().getStatusCode() == 204)
                response_bool = true;
        Instant end = Instant.now();
        LOG.info("Time taken for " + "Mapping - http://localhost:8080/SNOW/org/" +
    ORG + "/repo/" + name + "/oc/add/" + collaborator_name + "/" +
    permission_status + " = " + Duration.between(start, end));
        return response_bool;

    }
```

A total of 6 update functions were created, each for executing a different kind of update in AMD's github database.

## 3.2 Github Runner

The creation of the Github runner can be broken down into 4 parts

- Dockerfile creation
- Image creation from Dockerfile
- Uploading the image created to ACR
- Deploying the image stored in ACR to AKS

### 3.2.1 Dockerfile Creation

After obtaining the packages that have to be included in the runner as well as the base image, whether Windows or Ubuntu, we can start building the dockerfile.

Here is the pseudo code for the dockerfile

```
FROM mcr.microsoft.com/windows/servercore:ltsc2019

Install packages and change environment variables as needed

WORKDIR /github
RUN mkdir actions-runner; cd actions-runner
RUN Invoke-WebRequest -Uri https://github.com/actions/runner/releases/download/v2
    .279.0/actions-runner-win-x64-2.279.0.zip -OutFile C:\github\actions-runner\
    actions-runner-win-x64-2.279.0.zip
RUN Add-Type -AssemblyName System.IO.Compression.FileSystem
ENV GHTEMP = actions-runner-win-x64-2.279.0.zip
#RUN [System.IO.Compression.ZipFile]::ExtractToDirectory("$PWD/$GHTEMP", "$PWD")
RUN echo $PWD
RUN cd C:\github\actions-runner
RUN dir
RUN powershell.exe Expand-Archive -LiteralPath C:\github\actions-runner\actions-
    runner-win-x64-2.279.0.zip C:\github\actions-runner
COPY start.ps1 .
CMD powershell .\start.ps1
```

We begin with the base OS, in this case it is windows-servercore. Then we install all the required packages for the image, update the environment variables if required and make sure the image is read for the given needs. Now we install the Github runner to this image. The Github runner is publicly available. We download this, unzip it to get all the libraries and executables, and add it to the image.

Now we define the entry point to the image as ”start.ps1” which is a powershell script which takes care of linking the runner to the github organization for which this runner was intended for, as well as start the runner execution.

Here is the psudo code

```
$registration_url = "https://api.github.com/orgs/${Env:GITHUB_OWNER}/actions/
    runners/registration-token"
```

```
$payload = curl.exe --request POST "https://api.github.com/orgs/${Env:GITHUB_OWNER
    }/actions/runners/registration-token" --header "Authorization: Bearer ${Env:
    GITHUB_PERSONAL_TOKEN}"
echo "Requesting registration URL ${payload}"
$final_token = $payload.token
try {
    C:\github\actions-runner\config.cmd --url https://github.com/${Env:GITHUB_OWNER
    } --token ${final_token} --runnergroup ${Env:GITHUB_RUNNER_GROUP} --labels ${
    Env:GITHUB_RUNNER_LABELS}
    C:\github\actions-runner\run.cmd
}
finally {
    C:\github\actions-runner\config.cmd remove --token ${final_token}
}
```

First we obtain the registration token for linking the runner to the organization. In order to obtain this token we have to authenticate using our PAT, which is stored as an environment variable where we deploy, i.e AKS. Once we establish the connection of the runner with the github organization and update the config.cmd in this process, we can execute the runner, which actively listens for jobs as given by the github organization.

### 3.2.2 Image Creation

Now that we have the Dockerfile we can use Docker to build this file to obtain an image which can be deployed to a container in AKS. This can be performed using the command

```
docker build -t <image-name:tag> <PATH_of_Dockerfile>
```

where we specify the image name and the tag for the image to be created.

### 3.2.3 Uploading image to ACR

Now that we have built the image, we have to upload it into ACR. We specify the ACR location to where the image has to be uploaded. This is done using the command

```
docker tag <image-name> <ACR-name/repo>
```

Now we can push the image to ACR using the command

```
docker push <ACR-name/repo>
```

### 3.2.4 Creating a deployment in AKS

In order to create a deployment in AKS, we have to declare different yaml files. There are 2 yaml files for this deployment. Secret.yaml and deployment.yaml. The secret.yaml file contains the PAT necessary as mentioned above. The deployment.yaml files contains information like the namespace in which we are deploying, the runner group name, runner owner i.e the github organization name, the spec of the cluster as well as the image name from ACR among other parameters. Here is a snippet from the secret.yaml file.

```
metadata:
  name: github-secret
  namespace: github-runner-test-sj
type: Opaque
stringData:
  GITHUB_PERSONAL_TOKEN: "PAT"
```

Here is a snippet from deployment.yaml file.

```
metadata:
        name: runner-windows
        namespace: github-runner-test-sj
spec:
        replicas: 1
        containers:
        - name: runner-windows
        image: evuedsoacr.azurecr.io/amd-it/github-runners-windows:17503
        env:
        - name: GITHUB_OWNER
          value: amd-trial
        - name: GITHUB_RUNNER_GROUP
          value: windows-runners-test-sj
          - name: GITHUB_PERSONAL_TOKEN
          valueFrom:
            secretKeyRef:
              name: github-secret
              key: GITHUB_PERSONAL_TOKEN
```

These are a few of the important fields in the deployment.yaml file. We specify the location of the image, the github organization for which the runner has to be created, the number of replicas etc. We also specify the deployment to use the secrets defined by the secret.yaml file.

# Chapter 4

# Results and Discussions

## 4.1 Github API Server

As per our objectives defined, we were able to deliver all the required functionalities within the maximum possible latency limit defined. Some of the various functionalities were:

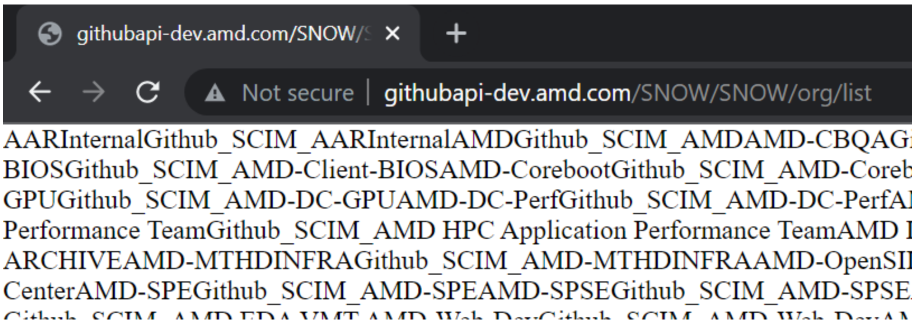
- List admins, given the repository name and the organization name.
- Create a repository given the organization.
- Change the visibility of a repository, given the organization and the repository.
- Update the permissions of an external collaborator, given the collaborator name, permission status, repository name and the organization name.
- Remove a collaborator from a repository, given the collaborator name, the repository name and the organization name.
- Delete a repository, given the repository, given the organization and the repository name.
- List all repositories, given the organization.
- List all the members, given the organization.
- List all outside collaborators, given the repository and the organization.

- Add external collaborator to a repository, given the collaborator name, repository name and the organization name.
- Get the Github login id, given the NTID of a user.
- List all organizations of the enterprise.

These are the list of utility functions.

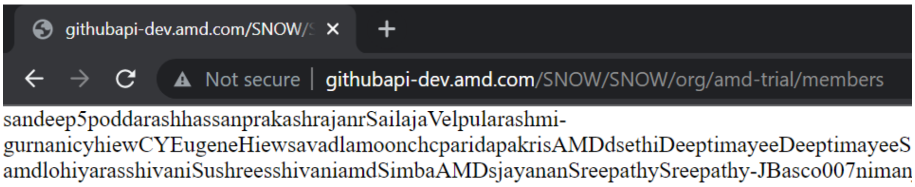
- getPAT() - Fetch the Personal access token from Azure key vault.
- getSAML() - Fetch the list of mapping from Github id to NTID.
- exeGraphql() - Execute a graphql query using HTTP Post request to Github API and retrieve the response.
- Execute-HTTP-POST() - Make a HTTP Post request by adding headers and required JSON body and send it to Github API.
- Execute-HTTP-PUT() - Make a HTTP Put request by adding headers and required JSON body and send it to Github API.
- Execute-HTTP-Patch() - Make a HTTP Patch request by adding headers and required JSON body and send it to Github API.
- Execute-HTTP-DELETE() - Make a HTTP Delete request by adding headers and required JSON body and send it to Github API.

Here are some of the output screenshots:


githubapi-dev.amd.com/SNOW/S
Not secure
githubapi-dev.amd.com/SNOW/SNOW/org/list

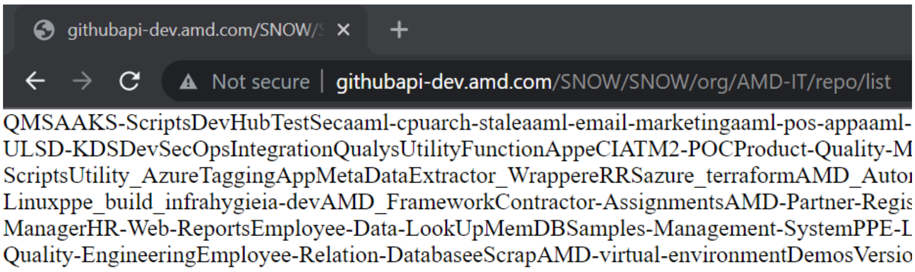
```
AARInternalGithub_SCIM_AARInternalAMDGithub_SCIM_AMDAMD-CBQAG
BIOSGithub_SCIM_AMD-Client-BIOSAMD-CorebootGithub_SCIM_AMD-Coreb
GPUGithub_SCIM_AMD-DC-GPUAMD-DC-PerfGithub_SCIM_AMD-DC-PerfAl
Performance TeamGithub_SCIM_AMD HPC Application Performance TeamAMD I
ARCHIVEAMD-MTHDINFRAGithub_SCIM_AMD-MTHDINFRAAMD-OpenSIl
CenterAMD-SPEGithub_SCIM_AMD-SPEAMD-SPSEGithub_SCIM_AMD-SPSE
Github_SCIM_AMD EDA VMT AMD Web DevGithub_SCIM_AMD Web DevAM
```

Figure 4.1: Output - List of Organizations


githubapi-dev.amd.com/SNOW/S
Not secure
githubapi-dev.amd.com/SNOW/SNOW/org/amd-trial/members

```
sandeep5poddarashhassanprakashrajanrSailajaVelpularashmi-
gurnanicyhiewCYEugeneHiewsavadlamoonchcparidapakrisAMDdsethiDeeptimayeeDeeptimayeeS
amdlohiyarasshivaniSushreesshivaniamdSimbaAMDsjayananSreepathySreepathy-JBasco007niman
```

FIGURE 4.2: Output - List of members in an organization


githubapi-dev.amd.com/SNOW/S
Not secure
githubapi-dev.amd.com/SNOW/SNOW/org/AMD-IT/repo/list

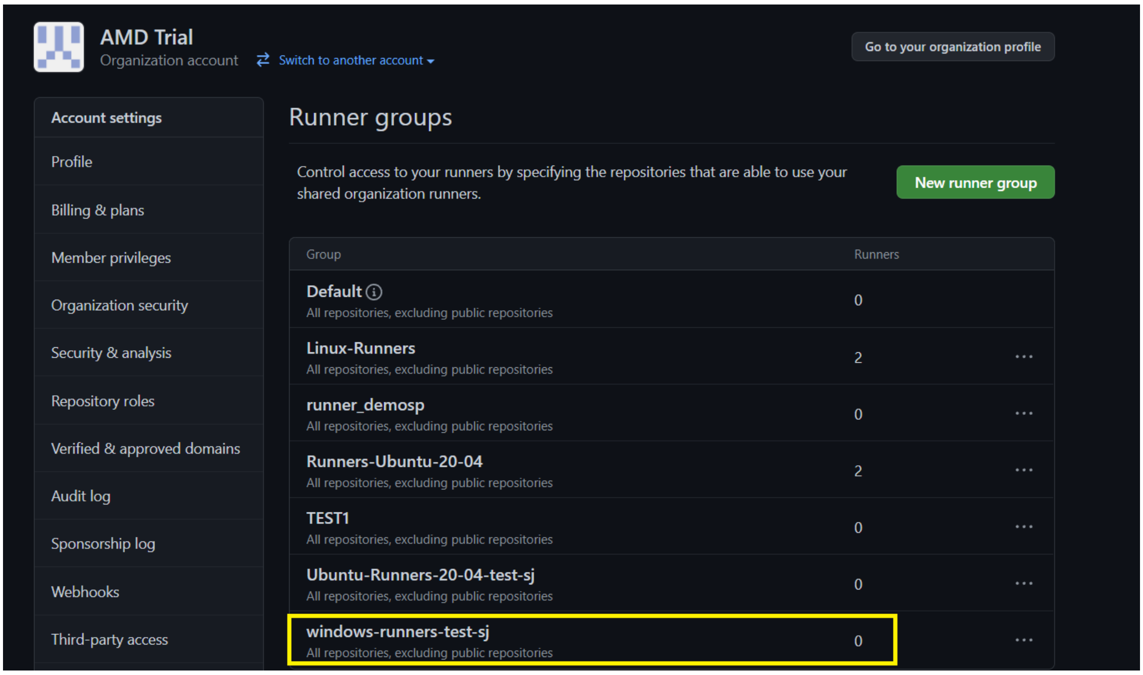
```
QMSAAKS-ScriptsDevHubTestSecaaml-cpuarch-staleaaml-email-marketingaaml-pos-appaaml-
ULSD-KDSDevSecOpsIntegrationQualysUtilityFunctionAppeCIATM2-POCProduct-Quality-M
ScriptsUtility_AzureTaggingAppMetaDataExtractor_WrappereRRSazure_terraformAMD_Auto
Linuxppe_build_infrahygieia-devAMD_FrameworkContractor-AssignmentsAMD-Partner-Regis
ManagerHR-Web-ReportsEmployee-Data-LookUpMemDBSamples-Management-SystemPPE-L
Quality-EngineeringEmployee-Relation-DatabaseeScrapAMD-virtual-environmentDemosVersio
```

FIGURE 4.3: Output - List of repositories of an organization

## 4.2 Github Runner

As per our objectives defined for the Github Runner, we were able to stand up the Github runner with the capabilities as requested by various organizations. It is able to successfully listen to jobs and communicate back the result.

Here are some of the output screenshots:

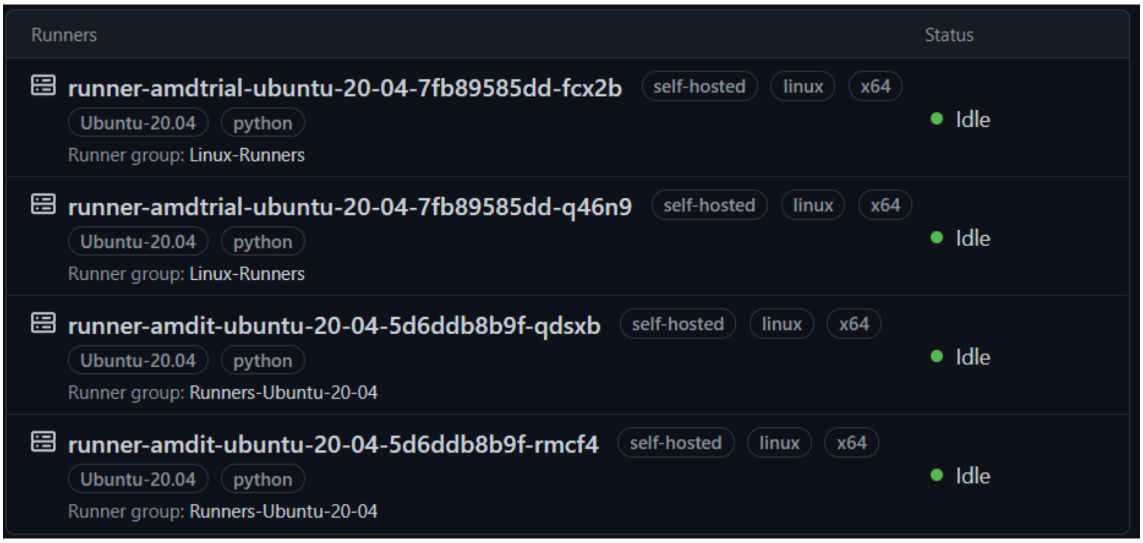


Runner groups

Control access to your runners by specifying the repositories that are able to use your shared organization runners.

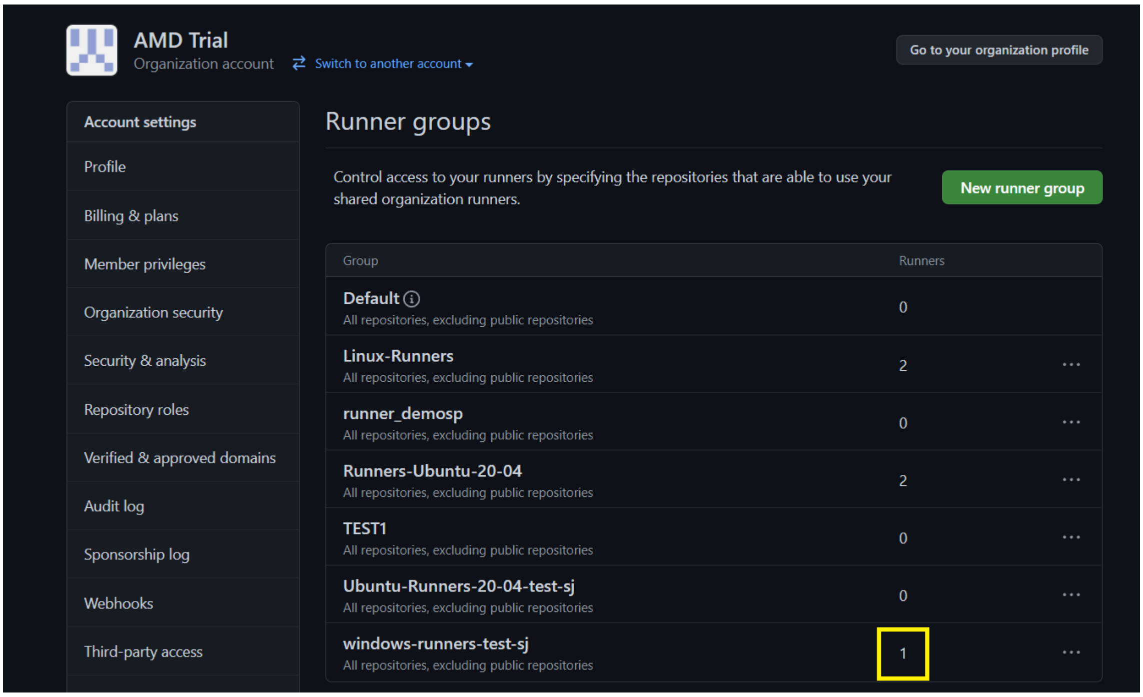
| Group | Runners |
|---|---|
| Default<br>All repositories, excluding public repositories | 0 |
| Linux-Runners<br>All repositories, excluding public repositories | 2 |
| runner_demosp<br>All repositories, excluding public repositories | 0 |
| Runners-Ubuntu-20-04<br>All repositories, excluding public repositories | 2 |
| TEST1<br>All repositories, excluding public repositories | 0 |
| Ubuntu-Runners-20-04-test-sj<br>All repositories, excluding public repositories | 0 |
| windows-runners-test-sj<br>All repositories, excluding public repositories | 0 |

FIGURE 4.4: Output - Count of runners in the runner group ”windows-runners-test-sj” before adding the runner



| Runners | Status |
|---|---|
| runner-amdtrial-ubuntu-20-04-7fb89585dd-fcx2b self-hosted linux x64<br>Ubuntu-20.04 python<br>Runner group: Linux-Runners | Idle |
| runner-amdtrial-ubuntu-20-04-7fb89585dd-q46n9 self-hosted linux x64<br>Ubuntu-20.04 python<br>Runner group: Linux-Runners | Idle |
| runner-amdit-ubuntu-20-04-5d6ddb8b9f-qdsxb self-hosted linux x64<br>Ubuntu-20.04 python<br>Runner group: Runners-Ubuntu-20-04 | Idle |
| runner-amdit-ubuntu-20-04-5d6ddb8b9f-rmcf4 self-hosted linux x64<br>Ubuntu-20.04 python<br>Runner group: Runners-Ubuntu-20-04 | Idle |

FIGURE 4.5: Output - List of runners before adding the runner

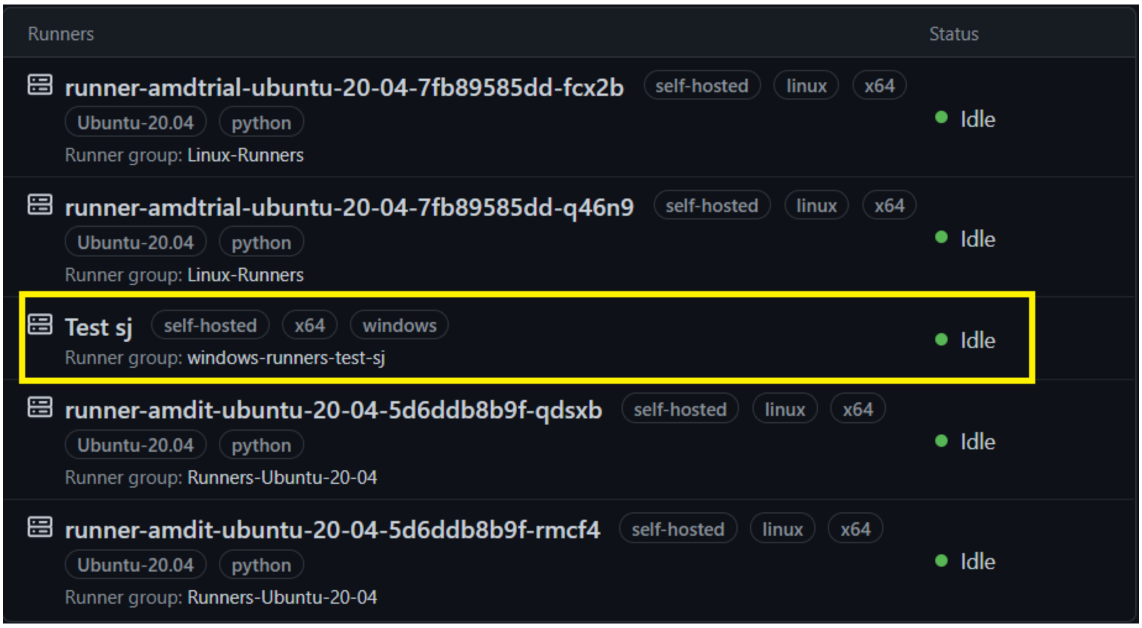


Runner groups

Control access to your runners by specifying the repositories that are able to use your shared organization runners.

| Group | Runners |
|---|---|
| Default<br>All repositories, excluding public repositories | 0 |
| Linux-Runners<br>All repositories, excluding public repositories | 2 |
| runner_demosp<br>All repositories, excluding public repositories | 0 |
| Runners-Ubuntu-20-04<br>All repositories, excluding public repositories | 2 |
| TEST1<br>All repositories, excluding public repositories | 0 |
| Ubuntu-Runners-20-04-test-sj<br>All repositories, excluding public repositories | 0 |
| windows-runners-test-sj<br>All repositories, excluding public repositories | 1 |

Figure 4.6: Output - Count of runners in the runner group "windows-runners-test-sj" after adding the runner



| Runners | Status |
|---|---|
| runner-amdtrial-ubuntu-20-04-7fb89585dd-fcx2b self-hosted linux x64 Ubuntu-20.04 python<br>Runner group: Linux-Runners | ● Idle |
| runner-amdtrial-ubuntu-20-04-7fb89585dd-q46n9 self-hosted linux x64 Ubuntu-20.04 python<br>Runner group: Linux-Runners | ● Idle |
| Test sj self-hosted x64 windows<br>Runner group: windows-runners-test-sj | ● Idle |
| runner-amdit-ubuntu-20-04-5d6ddb8b9f-qdsxb self-hosted linux x64 Ubuntu-20.04 python<br>Runner group: Runners-Ubuntu-20-04 | ● Idle |
| runner-amdit-ubuntu-20-04-5d6ddb8b9f-rmcf4 self-hosted linux x64 Ubuntu-20.04 python<br>Runner group: Runners-Ubuntu-20-04 | ● Idle |

Figure 4.7: Output - List of runners after adding the runner

You can see that the runner "Test sj", inside the runner group "windows-runners-test-sj" is in the idle state and is actively looking for the workloads to be given.

# Chapter 5

# Conclusion

The above mentioned 2 assignments given to me as a part of the DevSecOps internship, ”Create an internal GitHub API Server” and ”Create a Github Runner” were completed and all objectives and metrics set as a part of these assignments were met.

# Appendix A

# Code Snippets - Github API server

HTTP-Post() function

```
CloseableHttpResponse Execute_HTTP_POST(String api_url, String json_form) {
        CloseableHttpResponse response = null;
        CloseableHttpClient client = null;
        try {
            HttpPost httpPost = new HttpPost(api_url);
            httpPost.addHeader("Authorization", "token " + PAT);
            httpPost.addHeader("Content-Type", "application/json");
            StringEntity params = new StringEntity(json_form);
            httpPost.setEntity(params);
            client = HttpClients.createDefault();
            response = client.execute(httpPost);
            LOG.debug("Return from httpPost request - " + response);
            LOG.info("HTTP post finished successfully");
        }
        catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        catch (ClientProtocolException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        return response;
    }
```

## ExeGraphql() Function

```
public JSONObject exeGrapql(JSONObject jsonObj) {
        //https://www.tutorialspoint.com/how-to-read-parse-json-array-using-java
        Instant start = Instant.now();
        CloseableHttpClient client = null;
        CloseableHttpResponse response = null;

        client = HttpClients.createDefault();
        JSONObject rtnObj = new JSONObject();

        HttpPost httpPost = new HttpPost("https://api.github.com/graphql");
        httpPost.addHeader("Authorization", "Bearer " + PAT);
        httpPost.addHeader("Accept", "application/json");
        httpPost.addHeader("Content-Type", "application/json");

        try {
            StringEntity entity = new StringEntity(jsonObj.toString());
            httpPost.setEntity(entity);
            response = client.execute(httpPost);

            System.out.println(response.toString());
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(
    response.getEntity().getContent()));
            String line = null;
            StringBuilder builder = new StringBuilder();
            while ((line = reader.readLine()) != null) {
                builder.append(line);
            }

            JSONParser parser = new JSONParser();
            Object obj = parser.parse(builder.toString());
            rtnObj = (JSONObject) obj;
        } catch (Exception e) {
            e.printStackTrace();
        }
        Instant end = Instant.now();
        LOG.info("Time taken for exeGraphql = " + Duration.between(start, end));
        return rtnObj;
    }
```

# Appendix B

# Code Snippets - Github Runner

Dockerfile

```
FROM mcr.microsoft.com/windows/servercore:ltsc2019

ENV chocolateyUseWindowsCompression false
ENV GITHUB_PERSONAL_TOKEN ""
ENV GITHUB_OWNER ""
ENV GITHUB_RUNNER_GROUP = ""
ENV GITHUB_RUNNER_LABELS = ""
RUN powershell -Command \
        iex ((new-object net.webclient).DownloadString(’https://chocolatey.org/
    install.ps1’)); \#        choco feature disable --name showDownloadProgress

RUN choco install visualstudio2019buildtools -y
RUN choco install visualstudio2019community -y
RUN choco install openjdk -y
RUN choco install visualstudio2019-workload-xamarinbuildtools -y
RUN choco install microsoft-build-tools -y
RUN choco install visualstudio2019enterprise --norestart -y
RUN choco install windows-sdk-8.1 -y

RUN powershell (new-object System.Net.WebClient).Downloadfile(’http://javadl.oracle
    .com/webapps/download/AutoDL?BundleId=210185’, ’C:\jre-8u91-windows-x64.exe’)
RUN powershell start-process -filepath C:\jre-8u91-windows-x64.exe -passthru -wait
    -argumentlist "/s,INSTALLDIR=c:\Java\jre1.8.0_91,/L,install64.log"
RUN del C:\jre-8u91-windows-x64.exe

CMD [ "c:\\Java\\jre1.8.0_91\\bin\\java.exe", "-version"]
```

```
SHELL ["powershell", "-Command", "$ErrorActionPreference = ’Stop’;
    $ProgressPreference=’silentlyContinue’;"]
ADD https://aka.ms/vs/16/release/vs_buildtools.exe C:\TEMP\vs_buildtools.exe
RUN powershell -Command \
        Invoke-WebRequest ’https://aka.ms/vs/16/release/vs_community.exe’ -OutFile
    ’c\:TEMP\vs_community.exe’ -UseBasicParsing
#RUN & "C:TEMP\vs_community.exe" --add Microsoft.VisualStudio.Workload.NetWeb --
    quiet --wait --norestart --noUpdateInstaller | Out-Default

# Install .NET 4.8
RUN curl -fSLo dotnet-framework-installer.exe https://download.visualstudio.
    microsoft.com/download/pr/7afca223-55d2-470a-8edc-6a1739ae3252/
    abd170b4b0ec15ad0222a809b761a036/ndp48-x86-x64-allos-enu.exe ‘
    && .\dotnet-framework-installer.exe /q ‘
    && del .\dotnet-framework-installer.exe ‘
    && powershell Remove-Item -Force -Recurse ${Env:TEMP}\*

# Apply latest patch
RUN curl -fSLo patch.msu http://download.windowsupdate.com/c/msdownload/update/
    software/secu/2019/09/windows10.0-kb4514358-
    x64_b93fca9a74bb0e75ba9e878dd5f2fc537b92a32b.msu ‘
    && mkdir patch ‘
    && expand patch.msu patch -F:* ‘
    && del /F /Q patch.msu ‘
    && DISM /Online /Quiet /Add-Package /PackagePath:C:\patch\Windows10.0-kb4514358
    -x64.cab ‘
    && rmdir /S /Q patch

# ngen .NET Fx
ENV COMPLUS_NGenProtectedProcess_FeatureEnabled 0
RUN \Windows\Microsoft.NET\Framework64\v4.0.30319\ngen uninstall "Microsoft.Tpm.
    Commands, Version=10.0.0.0, Culture=Neutral, PublicKeyToken=31bf3856ad364e35,
    processorArchitecture=amd64" ‘
    && \Windows\Microsoft.NET\Framework64\v4.0.30319\ngen update ‘
    && \Windows\Microsoft.NET\Framework\v4.0.30319\ngen update

# Set PowerShell as default shell
SHELL [ "powershell", "-NoProfile", "-Command" ]
RUN $ProgressPreference = ’SilentlyContinue’

# Install PS Modules in C:\Modules
RUN New-Item -ItemType Directory -Name Modules -Path c:\ -Force | Out-Null ; ‘
    Install-PackageProvider -Name NuGet -Force | Out-Null; ‘
    Save-Module Az -Path c:\Modules -Confirm:$false | Out-Null; ‘
    Save-Module Pester -Path c:\Modules -Confirm:$false | Out-Null; ‘
    Save-Module PSScriptAnalyzer -Path c:\Modules -Confirm:$false | Out-Null
    # Add C:\Modules to PSModulePatch environment variable
RUN $ModulesPath = [environment]::GetEnvironmentVariable(’PSModulePath’, [System.
    EnvironmentVariableTarget]::Machine) ; ‘
    $NewModulesPath = ’C:\Modules;’ + $ModulesPath ; ‘
    [environment]::SetEnvironmentVariable(’PSModulePath’, $NewModulesPath, [System.
    EnvironmentVariableTarget]::Machine)

# Install Chocolately
ENV chocolateyUseWindowsCompression false
RUN Invoke-WebRequest -Uri ’https://chocolatey.org/install.ps1’ -OutFile ./choco-
    install.ps1 ; ‘
    .\\choco-install.ps1 | Out-Null ; ‘
    choco feature disable --name showDownloadProgress ; ‘
    Remove-Item ".\choco-install.ps1"

#RUN choco install openjdk -y
RUN choco install jdk8 -y
RUN choco install windows-sdk-8.1 -y
RUN choco install dotnetcore-sdk --version=2.1.300 -y
RUN choco install dotnetcore --version=2.1.0 -y
RUN choco install dotnetcore-runtime --version=2.1.0 -y

# Install Visual Studio Test Platform (vstest)
RUN Register-PackageSource -Name MyNuGet -Location https://www.nuget.org/api/v2 -
    ProviderName NuGet | Out-Null ; ‘
    Install-Package Microsoft.TestPlatform -Force | Out-Null ; ‘
    setx vstest ’c:\Program Files\PackageManagement\NuGet\Packages\Microsoft.
    TestPlatform.16.2.0\tools\net451\Common7\IDE\Extensions\TestPlatform’

# Install Visual Studio with dotNet workload
RUN Invoke-WebRequest "https://aka.ms/vs/16/release/vs_enterprise.exe" -OutFile ".\
    vs_enterprise.exe" -UseBasicParsing ; ‘
    Start-Process .\vs_enterprise.exe -ArgumentList ’--add Microsoft.VisualStudio.
    Workload.NetWeb --quiet --norestart’ -Wait ; ‘
    Remove-Item ".\vs_enterprise.exe" ; ‘
    setx visualstudio ’C:\Program Files (x86)\Microsoft Visual Studio\2019\
    Enterprise\MSBuild\Current\Bin’
    # Update PATH environment variable
RUN $machinePath = [environment]::GetEnvironmentVariable(’path’, [System.
    EnvironmentVariableTarget]::Machine) ; ‘
    $newMachinePath = ’C:\Program Files (x86)\Microsoft Visual Studio\2019\
    Enterprise\MSBuild\Current\Bin;’ + $machinePath ; ‘
    [environment]::SetEnvironmentVariable('path', $newMachinePath, [System.
    EnvironmentVariableTarget]::Machine)

RUN choco install webdeploy -y

# Install .NET Framework 4.6 Targeting Pack
RUN Invoke-WebRequest "https://go.microsoft.com/fwlink/?linkid=2099469" -OutFile
    ".\ndp46-targetingpack-kb3045566.exe" -UseBasicParsing ; `
    Start-Process .\ndp46-targetingpack-kb3045566.exe -ArgumentList '/quiet /
    norestart' -Wait ; `
    Remove-Item ".\ndp46-targetingpack-kb3045566.exe"

 RUN choco install python3 -y
 RUN choco install pip --allow-empty-checksums -y
 RUN choco install azure-cli -y
 RUN choco install maven -y
 RUN choco install googlechrome -y
 RUN choco install powershell -y
 RUN choco install reportgenerator.portable -y
 RUN choco install opencover.portable -y

# Clean up temp directory
# RUN Remove-Ie -Recurse ${Env:TEMP}\*

WORKDIR /github

RUN mkdir actions-runner; cd actions-runner
RUN Invoke-WebRequest -Uri https://github.com/actions/runner/releases/download/v2
    .279.0/actions-runner-win-x64-2.279.0.zip -OutFile C:\github\actions-runner\
    actions-runner-win-x64-2.279.0.zip
RUN Add-Type -AssemblyName System.IO.Compression.FileSystem
ENV GHTEMP = actions-runner-win-x64-2.279.0.zip
RUN [System.IO.Compression.ZipFile]::ExtractToDirectory("$PWD/$GHTEMP", "$PWD")
RUN echo $PWD
RUN cd C:\github\actions-runner
RUN dir
RUN powershell.exe Expand-Archive -LiteralPath C:\github\actions-runner\actions-
    runner-win-x64-2.279.0.zip C:\github\actions-runner
COPY start.ps1 .

#CMD java -version

#CMD dir 'C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\Team Tools
    \Dynamic Code Coverage Tools\'
```

```
CMD powershell .\start.ps1
```

Deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: runner-windows
  namespace: github-runner-test-sj
  labels:
    app: runner-windows
spec:
  replicas: 1
  selector:
    matchLabels:
      app: runner-windows
  template:
    metadata:
      labels:
        app: runner-windows
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": windows
      containers:
      - name: runner-windows
        image: evuedsoacr.azurecr.io/amd-it/github-runners-windows:17503
        env:
        - name: GITHUB_OWNER
          value: amd-trial
        - name: GITHUB_RUNNER_GROUP
          value: windows-runners-test-sj
        - name: GITHUB_RUNNER_LABELS
          value: Windows
        - name: RUNNER_WORKDIR
          value: _works
        - name: GITHUB_PERSONAL_TOKEN
          valueFrom:
            secretKeyRef:
              name: github-secret
              key: GITHUB_PERSONAL_TOKEN
        resources:
          limits:
            cpu: 8
            memory: 8192M
          requests:
            cpu: 4
            memory: 4096M



---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: runner-windows-hpa
  namespace: github-runner-test-sj
spec:
  maxReplicas: 10 # define max replica count
  minReplicas: 2  # define min replica count
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: runner-windows
  targetCPUUtilizationPercentage: 50 # target CPU utilization
```

Secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: github-secret
  namespace: github-runner-test-sj
type: Opaque
stringData:
  GITHUB_PERSONAL_TOKEN: "PAT"
```

# Bibliography

[1] "What is java spring boot?" [Online]. Available: https://www.ibm.com/cloud/learn/java-spring-boot

[2] "Graphql java basics." [Online]. Available: https://dzone.com/articles/a-beginners-guide-to-graphql-with-spring-boot