GITHUB ORG MIGRATION: SELF-HOSTED TO CLOUD (AUTOMATION)

SEWP ZG628T DISSERTATION

by

Akhil Gupta
2018HW70294

Dissertation Work carried out at

Wipro Technologies, Bangalore

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
Pilani (Rajasthan) India

September 2022

SEWP ZG628T DISSERTATION

GITHUB ORG MIGRATION: SELF-HOSTED TO CLOUD (AUTOMATION)

Submitted in partial fulfillment of the requirements of

M.Tech Software Engineering Degree Program

by

Akhil Gupta
ID No. 2018HW70294

Under the supervision of

Vikram Kumar
Administrator - ICORE- CIS, Wipro Technologies

Dissertation work carried out at

Wipro Technologies, Bangalore

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN)

(September 2022)

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

CERTIFICATE

This is to certify that the Dissertation entitled **GitHub Org Migration: Self-hosted to Cloud (Automation)**

and submitted by **Akhil Gupta** ID No. **2018HW70294**

in partial fulfillment of the requirements of SEWP ZG628T Dissertation, embodies the work done by him/her under my supervision.

Signature of the Supervisor
Name: Vikram Kumar
Designation: Administrator - ICORE- CIS

Date: 11-09-2022

ACKNOWLEDGEMENT

I would like to express my deepest appreciation to my supervisor Er. Vikram Kumar, whose valuable guidance has helped me in the completion of this dissertation. His suggestions and his instructions have served as the major contributor to the completion of the dissertation.

I would like to extend my gratitude and thanks to my examiners Er. Moni Kumari and Er. Sanapal Alekhya who has helped me with their valuable suggestions and guidance has been very helpful in various phases of the completion of the dissertation.

I would like to express my special gratitude and thanks to BITs Pilani and the Training department for all their efforts in organizing this course.

Last, but not least, my parents are also an important inspiration for me. So, with due regard, I express my gratitude to them.

Abstract

Version control tools like Git, Concurrent Version System (CVS), Mercurial, and many others are widely used in the IT development area. Such tools offer to streamline the development process and keep a history of all changes within a code. So, if a developer makes a mistake, then it can be undone and fixed by comparing the new code with a previous version. Version Control Software (VCS) also known as Source Code Management (SCM) tools or Revision Control System (RCS) is a great fit for any web development company around the globe.

With the increase in demand for software, codebase data is gradually increasing on servers. To manages such data, the IT infrastructure needs to expand the existing data center, which involves large CapEx and OpEx. To overcome this challenge cloud is the ultimate solution. Still migrating the data from self-hosted GitHub to the cloud is another big challenge as this involves various categories of data for migration like code history, pull requests, releases, issues, etc.
Using the planned automation "GitHub Org Migration: Self-hosted to Cloud" the admin team can easily migrate the GitHub Org to the cloud with reduced overhead.

Migrating any GitHub organization from self-hosted to enterprise cloud requires various manual activities to be performed by the admin team, few are mentioned below:

- Create an empty org on GitHub enterprise cloud.
- Provide enterprise license to users (Providing access).
- Adding users with the specific role (owner/member) in the new org.
- Create a migrator file for repositories from self-hosted GitHub and import it into GitHub enterprise.

To reduce the overhead of a series of manual activities can be automated, as suggested below:

- Input the name of the org that exists in the self-hosted GitHub. A Python script can be used for gathering the active owner and members of entered organizations.
- Microsoft GraphQL (https://docs.microsoft.com/en-us/graph/api/overview?view=graph-rest-1.0) can be used to validate users' active status and provide the license to access the GitHub enterprise cloud using an automation script.
- Using APIs provided by GitHub (https://docs.github.com/en/rest/guides/getting-started-with-the-rest-api) new organization can be created on the GitHub enterprise cloud. Members can be assigned to the created org with respective roles using the same.
- A prepared migrator file can be imported to https://eci.github.com/ for successful migration or repositories. Documentation can be found at http://git.io/vtLRM.

Using the planned automation, manual activities such as creating org, verifying users' active status, and providing licenses for user access get reduced for the admin team. A final report containing logs from all activities generated by the script can be reviewed by the admin team. This will come out as the ultimate solution while performing bulk migration of organizations.

Contents

List of Symbols & Abbreviations used .....[07]

List of Tables / Figures .....[07]

1. Introduction .....[08]
   1.1. What is Version Control System (VCS)? .....[08]
   1.2. Type of VCS .....[08]
      1.2.1. Local Version Control System .....[08]
      1.2.2. Centralized Version Control System .....[08]
      1.2.3. Distributed Version Control System .....[09]
   1.3. Difference between Git and GitHub .....[09]
      1.3.1. How GitHub works .....[09]
   1.4. Why this automation is needed .....[09]
2. Project Modules …….....[10]
   2.1. Module 1 .....[10]
   2.2. Module 2 .....[10]
   2.3. Module 3 .....[10]
3. Methodology .....[10]
4. Requirements .....[10]
5. Design .....[11]
6. Coding .....[12]
   6.1. Code .....[12]
   6.2. Explanation of used functions .....[15]
7. Execution .....[16]
   7.1. Automated Script Execution .....[16]

Appendix .....[17]
   1. Data Flow Diagram (DFD – 0 Level) .....[17]

References .....[18]

List of Symbols & Abbreviations used

Keywords	Description
AD	Active Directory
API	Application Programming Interface
AZ	Azure
CSV	Comma-Separated Values
CapEx	Capital Expenditures
GHEC	GitHub Enterprise Cloud
GHES	GitHub Enterprise Server
IT	Information Technology
OS	Operating System
OpEx	Operating Expenditures
Org	GitHub Organization
PAT	Personal Access Token
Repo	GitHub Repository
URL	Uniform Resource Locator
VCS	Version Control System

List of Tables / Figures

S. No.	Figure Description
Fig 1.1	Version Control System in a local computer.
Fig 1.2	Centralized Version Control System.
Fig 1.3	Distributed Version Control System.
Fig 7.1	GHES Source org.
Fig 7.2	Initial prompt during script execution, input org name.
Fig 7.3	GHES org's Active/Inactive users' detail.

1. Chapter One: Introduction

1.1 What is Version Control System (VCS)? Version Control Systems are the software tools used for tracking and managing changes made to the source code during software development. It keeps a record of every single change made to the code. It also allows us to revert to the previous version of the code if any mistake is made in the current version. Without a VCS in place, it would not be possible to monitor the development of the project.

1.2 Type of VCS: There are three types of VCS:

- Local Version Control System
- Centralized Version Control System
- Distributed Version Control System

1.2.1 Local Version Control System: Local Version Control System is in your local machine. If the local machine crashes, it would not be possible to retrieve the files, and all the information will be lost. If anything happens to a single version, all the versions made after that will be lost. Also, with the Local Version Control System, it is not possible to collaborate with other collaborators.

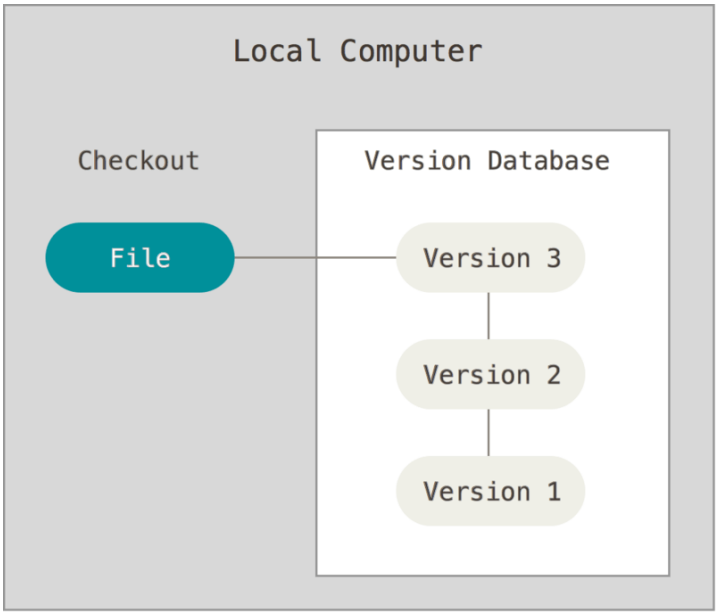


Fig 1.1

To collaborate with other developers on other systems, Centralized Version Control Systems are developed.

1.2.2 Centralized Version Control System: In the Centralized Version Control system, there will be a single central server that contains all the files related to the project, and many collaborators checkout files from this single server (you will only have a working copy). The problem with the Centralized Version Control Systems is if the central server crashes, almost everything related to the project will be lost.

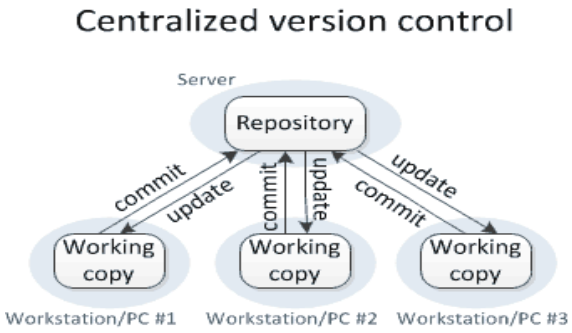


Fig 1.2

1.2.3 Distributed Version Control System: In a distributed version control system, there will be one or more servers and many collaborators like in the centralized system. But the difference is that not only do they check out the latest version, but each collaborator will have an exact copy (mirroring) of the main repository (including its entire history) on their local machines.

Each user has their own repository and a working copy. This is very useful because even if the server crashes we would not lose everything as several copies are residing in several other computers.

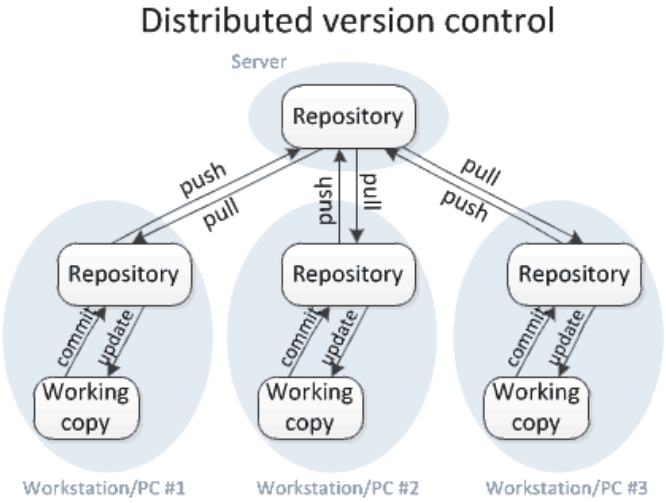


Fig 1.3

1.3 Difference between Git and GitHub: Git is a version control system of distributed nature that is used to track changes in source code during software development. It aids in coordinating work among programmers, but it can be used to track changes in any set of files. The main objectives of Git are speed, data integrity, and support for distributed, nonlinear workflows.

GitHub is a Git repository hosting service that adds many of its features. GitHub provides a Web-based graphical interface. It also provides access control, collaboration features, and basic task management tools for every project.

1.3.1 How GitHub works: GitHub hosts Git repositories and provides developers with tools to ship better code through command line features, issues (threaded discussions), pull requests, code reviews, or the use of a collection of free and for-purchase apps in the GitHub Marketplace. With collaboration layers like the GitHub flow, a community of 15 million developers, and an ecosystem with hundreds of integrations, GitHub changes the way software is built.

GitHub builds collaboration directly into the development process. Work is organized into repositories where developers can outline requirements or direction and set expectations for team members. Then, using the GitHub flow, developers simply create a branch to work on updates, commit changes to save them, open a pull request to propose and discuss changes, and merge pull requests once everyone is on the same page.

1.4 Why this automation is needed: With the increase in demand for software, codebase data is gradually increasing on servers. To manages such data, the IT infrastructure needs to expand the existing data center, which involves large CapEx and OpEx. To overcome this challenge cloud is the ultimate solution. Still migrating the data from self-hosted GitHub to the cloud is another big challenge as this involves various categories of data for migration like code history, pull requests, releases, issues, etc.

Using the planned automation “GitHub Org Migration: Self-hosted to Cloud” the admin team can easily migrate the GitHub Org to the cloud with reduced overhead.

2. Chapter Two: Project Modules

GitHub Org Migration: Self-hosted to Cloud includes communication between various applications i.e., GitHub Internal, Azure Active Directory, and GitHub Cloud. These applications can be categorized as Module 1, Module 2, and Module 3 respectively.

2.1 Module 1: This module involves an internally hosted GitHub application (Source), where orgs and repos already exist with all the production data. Information such as org detail, org members' and owners' IDs will be extracted from this module for the target org.

2.2 Module 2: This module involves Azure Active Directory (AZ AD). All the extracted IDs will be checked if active. Also, user IDs will get added to the AZ Group to provide user access to the GitHub Enterprise Cloud.

2.3 Module 3: This module involves the GitHub Enterprise Cloud application (Destination). Extracted data from the source module like org detail, members' and owners' IDs will be utilized here for migration.

3. Chapter Three: Methodology

Why GitHub Org Migration: Self-hosted to Cloud is needed? Migrating any GitHub organization from self-hosted to enterprise cloud requires various manual activities to be performed by the admin team, few are mentioned below:

- Create an empty org on GitHub enterprise cloud.
- Provide enterprise license to users (Providing access).
- Adding users with the specific role (owner/member) in the new org.
- Create a migrator file for repositories from self-hosted GitHub and import it into GitHub enterprise.

To reduce the overhead of a series of manual activities can be automated, as suggested below:

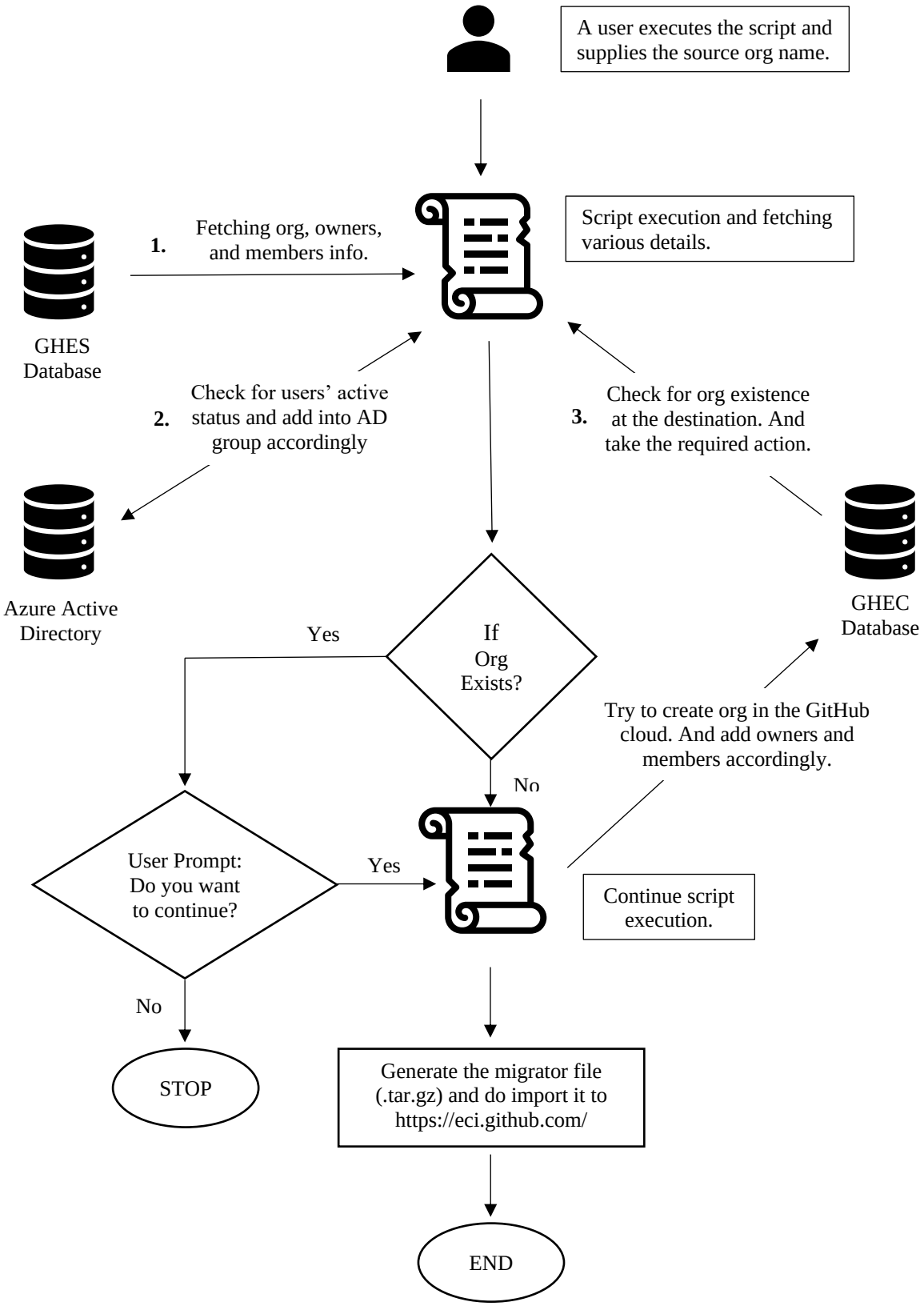
- Input the name of the org that exists in the self-hosted GitHub. A Python script can be used for gathering the active owner and members of entered organizations.
- Microsoft GraphQL (https://docs.microsoft.com/en-us/graph/api/overview?view=graph-rest-1.0) can be used to validate users' active status and provide the license to access the GitHub enterprise cloud using an automation script.
- Using APIs provided by GitHub (https://docs.github.com/en/rest/guides/getting-started-with-the-rest-api) new organization can be created on the GitHub enterprise cloud. Members can be assigned to the created org with respective roles using the same.
- A prepared migrator file can be imported to https://eci.github.com/ for successful migration or repositories. Documentation can be found at http://git.io/vtLRM.

4. Chapter Four: Requirements

Following are the requirements for the creation of the mentioned automation.

- Runtime Python 3.x is needed with the 'requests' package.
- Personal Access Token (PAT) from GHEC to access GitHub cloud.
- Personal Access Token (PAT) from GHES to access self-hosted GitHub.
- Client id, Client secret, Grant type, and URL from AZ AD OAuth for generating token to access AZ AD.
- AZ AD group id to add users for assigning licenses.

5. Chapter Five: Design



6. Chapter Six: Coding

6.1 Code:

```
import requests
import json

##Generating GHEC Token.
def get_ghec_token():
    return "ghp_qIXqpEfczmjTFvYVFajxxxxxx"

##Generating token for AD.
def get_token():
  payload = {"client_id": "4596249c-6b94-4870-aeb1-xxxxxxx",
  "scope": "https://graph.microsoft.com/.default",
  "client_secret": "4cJ7Q~pWQ2DuURNoYWmjUnhDM-xxxxxxx",
  "grant_type": "client_credentials"}
  url= 'https://login.microsoftonline.com/906aefe9-76a7-4f65-xxxx-
xxxxxxx/oauth2/v2.0/token'

  headers = {
      'Content-Type' : 'application/x-www-form-urlencoded'
  }

  try:
      response = requests.request("POST", url, headers=headers,
data=payload)
      token_value = response.json().get('access_token')
      return str(token_value) #Returning token to the caller in the main
function.
  except:
      print("[Terminated] Unable to get access token.")
      exit() #Program exited if no token generated.

##Check if the NTID is active or inactive in AD
def get_active_status(headers,user_list):
  output_dict = {'active': [], 'inactive': [], 'active_id': []}
  for i in user_list:
    oid_url =
"https://graph.microsoft.com/beta/users?$filter=startswith(userPrincipalNam
e,'"+ i + "@')&select=userPrincipalName"
    try:
      response = requests.request("GET", oid_url, headers=headers)
      user_data = json.loads(response.text)
      if user_data["value"]:
        output_dict["active"] +=
[user_data["value"][0]["userPrincipalName"]]
        output_dict["active_id"] += [i]
      else:
        output_dict["inactive"] += [i]
    except:
      output_dict["inactive"] += [i]
  return output_dict

##Get list of all the members in org
def get_ghes_members(org, headers):
    loop = True
    members = []
    count = 1
```

```
    while loop:
        url =
"https://github.comcast.com/api/v3/orgs/"+org+"/members?per_page=100&page="
+ str(count)
        response = requests.request("GET", url, headers=headers)
        results = response.json()
        if not len(results) == 0:
            if ("message" in results) and (results['message'] == "Not
Found"):
                print("Unable to find Org or authenticated user is not part
of Org")
                exit()
            for result in results:
                if not type(result)=='str':
                    members.append(result['login'])
            count +=1
        else:
            loop =False
    owner = []
    non_owner = []

    for member in members:
        url1 =  "https://github.comcast.com/api/v3/orgs/"+ org +
"/memberships/"+ member
        response = requests.request("GET", url1, headers=headers)
        result = response.json()
        if 'role' in result.keys():
            if result['role'] == 'admin':
                owner.append(member)
            if result['role'] == 'member':
                non_owner.append(member)
    return {"member": non_owner, "admin": owner}


##Generate All possible combinations of - and _ for service accounts
def get_combination(user):
    result = []
    count = 0
    for ch in user:
        if ch == '-':
            count += 1
    if count >= 1:
        result.append(user.replace('-','_'))

    if count >=2:
        result.append(user.replace('-','_', 1))
        result.append(user.replace('-','_', 2).replace('_','-',1))
    if count >=3:
        result.append(user.replace('-','_', 2))
        result.append(user.replace('-','_', 3).replace('_','-',2))
        result.append(user.replace('-','_', 3).replace('_','-
',2).replace('-','_', 1))
    if count >=4:
        result.append(user.replace('-','_', 3))
        result.append(user.replace('-','_', 4).replace('_','-',1))
        result.append(user.replace('-','_', 4).replace('_','-',2))
        result.append(user.replace('-','_', 4).replace('_','-',3))

    return result
```

```
if __name__ == '__main__':

    ##Generating header for GHES API
    ENT_TOKEN = "f831100027ef4b6xxxxxx"
    ghes_header = {
    'Authorization': 'Bearer '+ ENT_TOKEN
    }

    ##Generating header for AD API
    AD_TOKEN = get_token()
    ad_header = {
    'Authorization': 'Bearer {}'.format(AD_TOKEN)
    }

    ##Generating header for GHEC API
    GIT_ADM_TOKEN = get_ghec_token()
    ghec_header = {
    'Authorization': 'Bearer ' + GIT_ADM_TOKEN,
    'Content-Type': 'application/json'
    }

    ##Input Org GHES Org name from user
    org = input("Enter Org name: ")

    ##Fetch all the members of Org
    output = get_ghes_members(org, ghes_header)
    if "entdetsa" in output['admin']:
        output['admin'].remove("entdetsa")
    if "entdetsa" in output['member']:
        output['member'].remove("entdetsa")

    ##Check Active and inactive users
    admin = get_active_status(ad_header,output['admin'])
    #Check service account marked as inactive and update
    if admin["inactive"]:
        for user in admin["inactive"]:
            if "-" in user:
                res = get_combination(user)
                svc_acc = get_active_status(ad_header,res)
                if svc_acc["active"]:
                    admin["active"] += svc_acc["active"]
                    admin["active_id"].append(user)
                    admin["inactive"].remove(user)

    print("\n\nActive Owners: ", admin['active'])
    print("\nInactive Owners: ", admin['inactive'])
    member = get_active_status(ad_header,output['member'])
    #Check service account marked as inactive and update
    if member["inactive"]:
        for user in member["inactive"]:
            if "-" in user:
                res = get_combination(user)
                svc_acc = get_active_status(ad_header,res)
                if svc_acc["active"]:
                    member["active"] += svc_acc["active"]
                    member["active_id"].append(user)
                    member["inactive"].remove(user)

    print("\n\nActive Members: ", member['active'])
    print("\nInactive Members: ", member['inactive'])
```

6.2 Explanation of used functions:

- get_ghec_token(): Function is responsible for fetching the GHEC PAT token to provide access to the GitHub cloud.

- get_token(): AZ AD token can be generated inside this function using OAuth credentials and supply the generated token to the main function.

- get_active_status(headers,user_list): Function accepts two arguments header (used in requests function) and user_list (list of user ids). This function verified the active status of user ids in the input list and return dictionary (`{'active': [], 'inactive': [], 'active_id': []}`) where dictionary key 'active' contains the list of active users' email, key 'inactive' contains the list of inactive users' id, and key 'active_id' contains the list of active users' id.

- get_ghes_members(org, headers): Function accepts two arguments org (Organization name input by the user) and header (used in requests function). This function returns `{"member": non_owner, "admin": owner}` where dictionary key 'member' contains the list of GHES org members' id and key 'admin' contains the list of GHES org owners' id.

- get_combination(user): Function accept user id as argument. This is used for a special case (service account user). Here in Comcast service account can contain '-'/'_' but in GitHub '_' automatically change into '-'. Using this function we try to generate various combinations by replacing '-' with '_' to get actual user ID available in AZ AD.
  Eg: User id in GitHub: svc-det-team, function will generate combinations are svc_det-team, svc-det_team, svc_det_team.

7. Chapter Seven: Execution

7.1 Automated Script Execution: As per the requirement, executing the prepared script on centos OS with python 3.6 with the 'requests' package installed and named the code file as “script.py”. Our target org which is required to migrate from GHES to GHEC is “XfinityDesignSystem” Fig 7.1 is the snip of Org from self-hosted GitHub (GHES).

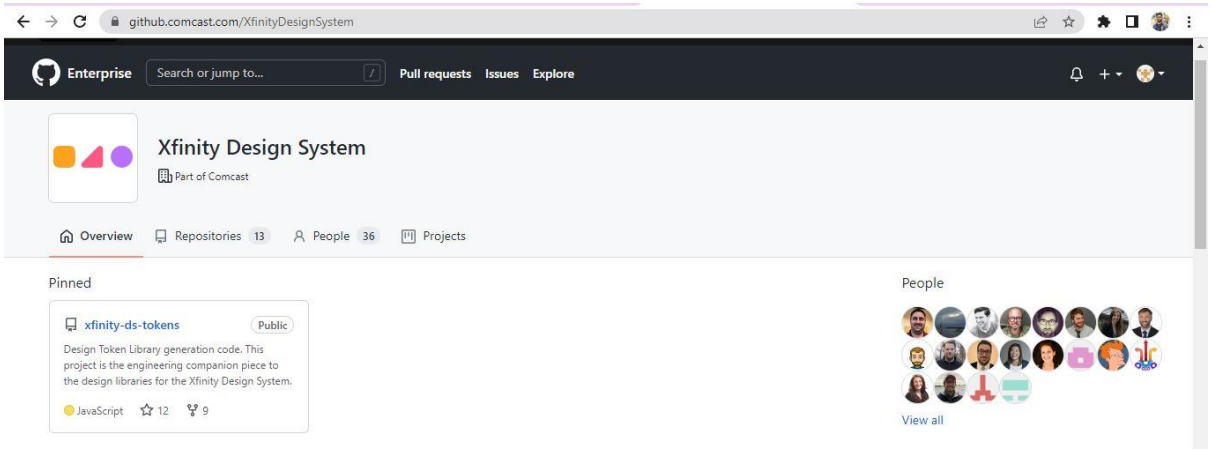


Fig 7.1

For executing the script, run “python3 script.py”. This will execute the script using python 3, prompt will appear “Enter Org name: “. Our target org is “XfinityDesignSystem”, provided input. As shown in Fig 7.2.

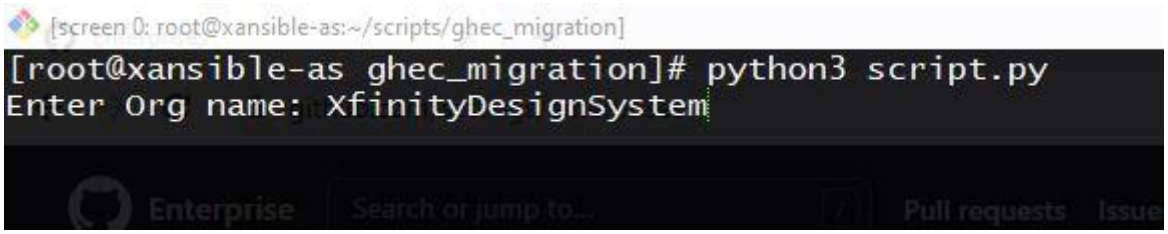


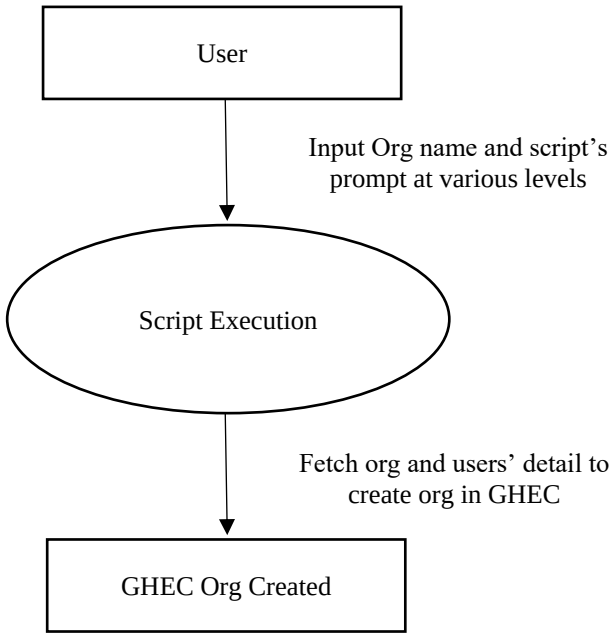
Fig 7.2

The script will accept the org name and pass the same to the defined function named “get_ghes_members(org, headers)”. This will check the org existence and fetch and returns owners and members of the org. In case it does not get the entered org in GHES, return “Unable to find Org or authenticated user is not part of Org” and terminate execution. Fig 7.3 shows script fetched detail of given org.

Fig 7.3

Appendix

1. **Data Flow Diagram (DFD – 0 Level):**



References

1. GitHub Cloud: https://github.com/
2. GitHub Documentation: https://docs.github.com/en/get-started/using-git/about-git
3. Python Documentation: https://docs.python.org/3/tutorial/
4. Python Learning: https://www.tutorialspoint.com/python/index.htm
5. VS code and documentation: https://code.visualstudio.com/docs