Lien du dossier google drive contenant touts les fichiers extraits et utilisés dans le projet :

https://drive.google.com/drive/folders/1MVkSx81zCHLOEI4EFHq4PfyJRqQp2UIv?usp=sharing

Lien du Github repo :

aaa

1. **Extraction des 48 projets** 
   1. **Script utilisé :**

#!/usr/bin/env python

import github  # Importing the GitHub library to interact with GitHub API

import logging  # For logging messages and errors

import urllib.parse  # To parse and create URLs

import requests  # For making HTTP requests

import time  # For handling time-related tasks, like sleeping

import csv  # For CSV file operations

import datetime  # For handling date and time data

from github import Github  # Importing specifically the Github class from github library

# Constants for the file paths

LOG\_FILE = "./github\_collect.log"

OUTPUT\_FILE = './saver/output.csv'

# GitHub API tokens used for authentication to avoid rate limits

GITHUB\_API\_TOKENS = [

    # Replace these with actual tokens

]

# Constants for setting up the search query parameters

ITEM\_PER\_PAGE = 100

SEARCH\_QUERY = "extension:tf"

PARAMS = {"q": SEARCH\_QUERY, "sort": "stars", "order": "desc", "per\_page": str(ITEM\_PER\_PAGE)}

BASE\_SEARCH\_API\_URL = "https://api.github.com/search/code"

BASE\_SEARCH\_URL = BASE\_SEARCH\_API\_URL + "?" + urllib.parse.urlencode(PARAMS)

# Headers for the columns in the output CSV file

headers = ["repo\_full\_name", "created\_at", "updated\_at", "description", "num\_stars",

           "num\_forks", "base\_language", "has\_issues", "is\_archived", "topics",

           "repo\_languages", "size", "has\_license", "num\_contributors"]

def get\_contributors(contributors\_url, headers):

    # Function to get the number of contributors by paginating through the contributor's endpoint

    contributors\_count = 0

    page = 1

    while True:

        paginated\_url = f"{contributors\_url}?per\_page=100&page={page}"

        response = requests.get(url=paginated\_url, headers=headers)

        if response.status\_code == 200:

            contributors = response.json()

            contributors\_count += len(contributors)

            if len(contributors) < 100:

                break

            page += 1

        else:

            handle\_rate\_limit(response)

    return contributors\_count

def load\_visited\_repos(output\_file):

    # Loads the set of repositories already visited to avoid duplication

    visited = set()

    try:

        with open(output\_file, 'r', newline='', encoding='utf-8') as infile:

            csv\_reader = csv.reader(infile)

            next(csv\_reader, None)  # Skip the header

            for row in csv\_reader:

                if row:

                    visited.add(row[0])

    except FileNotFoundError:

        pass

    return visited

def handle\_rate\_limit(response):

    # Handles API rate limiting by waiting before making new requests

    print("reponse:: ", response.headers)

    duration = int(response.headers.get("Retry-After", 60))

    logging.warning(f"Limit exhausted. Sleep for {duration} secs\n{response.text}")

    time.sleep(duration)

def get\_secondary\_used\_languages(language\_url, headers):

    # Retrieves and calculates the percentage usage of languages in the repository

    response = requests.get(url=language\_url, headers=headers)

    if response.status\_code == 200:

        languages = response.json()

        total\_count = sum(languages.values())

        return {lang: round((count / total\_count) \* 100, 2) for lang, count in languages.items()}

    elif response.status\_code == 403:

        handle\_rate\_limit(response)

def write\_repo\_info\_to\_csv(repo\_info, output\_file):

    # Writes the gathered repository information into a CSV file

    with open(output\_file, "a", encoding='utf-8', newline='') as outfile:

        csv\_writer = csv.writer(outfile, delimiter=',')

        csv\_writer.writerow(repo\_info)

def setup\_logging():

    # Setup logging configuration

    logging.basicConfig(filename=LOG\_FILE, level=logging.INFO, format='%(asctime)s | %(levelname)s | %(message)s')

def init\_file(output\_file, headers):

    # Initializes the CSV file and writes the header

    with open(output\_file, 'w', newline='') as outfile:

        csv\_writer = csv.writer(outfile, delimiter=',')

        csv\_writer.writerow(headers)

def main(OUTPUT\_FILE):

    # The main function to orchestrate the data collection

    setup\_logging()

    visited\_repos = load\_visited\_repos(OUTPUT\_FILE)

    page\_index = 1

    state = True

    repos\_counter = 0

    while state:

        search\_url = f"{BASE\_SEARCH\_URL}&page={page\_index}"

        headers = {"Authorization": f"token {GITHUB\_API\_TOKENS[page\_index % len(GITHUB\_API\_TOKENS)]}",

                   "Accept": "application/vnd.github.v3+json"}

        g = Github(GITHUB\_API\_TOKENS[page\_index % len(GITHUB\_API\_TOKENS)])

        print(g.get\_rate\_limit())

        response = requests.get(search\_url, headers=headers)

        if response.status\_code == 200:

            files = response.json().get("items", [])

            print('visited\_repos :', visited\_repos)

            for file in files:

                if file["path"].endswith(".tf"):

                    repo\_full\_name = file.get("repository", {}).get("full\_name", None)

                    is\_not\_fork\_repo = file.get("repository", {}).get("fork", None)

                    if is\_not\_fork\_repo == False and repo\_full\_name and repo\_full\_name not in visited\_repos:

                        repo\_info = fetch\_repo\_info(file["repository"]["url"], headers)

                        print(repo\_info)

                        if repo\_info:

                            write\_repo\_info\_to\_csv(repo\_info, OUTPUT\_FILE)

                            visited\_repos.add(repo\_full\_name)

                            repos\_counter += 1

            page\_index += 1

        elif response.status\_code == 403:

            print("Please I'm waiting !!!!!!!!!!!!!!!!!!!!!!! ")  # This is the line you mentioned

            handle\_rate\_limit(response)

        elif response.status\_code in [422, 404]:

            state = False

            logging.info(response.json().get("message", "422 or 404 Error"))

        else:

            state = False

            logging.error(f"Query failed with code {response.status\_code}.\n{response.text}")

            print("!!!!! I'm Sleeping, Sorry You should wait little bit !!!!!")

            raise Exception(f"Query failed with code {response.status\_code}.\n{response.text}")

            time.sleep(1800)

    logging.info(f"{repos\_counter} repos added")

    print(f"{repos\_counter} repos added")

def fetch\_repo\_info(repo\_url, headers):

    # Fetches detailed information about a specific repository

    response = requests.get(url=repo\_url, headers=headers)

    if response.status\_code == 200:

        repo = response.json()

        repo\_info = [

            repo.get("full\_name"),

            repo.get("created\_at"),

            repo.get("updated\_at"),

            repo.get("description", ""),

            repo.get("stargazers\_count", 0),

            repo.get("forks\_count", 0),

            repo.get("language"),

            repo.get("has\_issues", False),

            repo.get("archived", False),

            repo.get("topics", []),

            get\_secondary\_used\_languages(f'{repo["url"]}/languages', headers),

            repo.get("size", 0),

            bool(repo.get("license")),

        ]

        return repo\_info

if \_\_name\_\_ == '\_\_main\_\_':

    main(OUTPUT\_FILE)

1. **Liste des 48 projets extraits**

* Azure\_\_Avere
* pluralsh\_\_plural-artifacts
* SUSE\_\_ha-sap-terraform-deployments
* GiganticMinecraft\_\_seichi\_infra
* alphagov\_\_govuk-aws
* kubernetes-sigs\_\_kubespray
* Azure\_\_az-hop
* CDCgov\_\_prime-simplereport
* clong\_\_DetectionLab
* 2i2c-org\_\_infrastructure
* cloudfoundry\_\_bosh-bootloader
* deckhouse\_\_deckhouse
* zenml-io\_\_mlstacks
* bridgecrewio\_\_terragoat
* apache\_\_beam
* compiler-explorer\_\_infra
* cattle-ops\_\_terraform-aws-gitlab-runner
* aws-observability\_\_terraform-aws-observability-accelerator
* cookpad\_\_terraform-aws-eks\_\_versioning
* rust-lang\_\_simpleinfra
* ministryofjustice\_\_modernisation-platform
* ManagedKube\_\_kubernetes-ops
* wireapp\_\_wire-server-deploy
* google\_\_go-cloud
* kube-hetzner\_\_terraform-hcloud-kube-hetzner
* Optum\_\_dce
* nasa\_\_cumulus
* iits-consulting\_\_terraform-opentelekomcloud-project-factory
* nebari-dev\_\_nebari
* awslabs\_\_data-on-eks
* uyuni-project\_\_sumaform
* kbst\_\_terraform-kubestack
* splunk\_\_attack\_range
* GoogleCloudPlatform\_\_hpc-toolkit
* Worklytics\_\_psoxy
* chanzuckerberg\_\_cztack
* Azure\_\_sap-automation
* alphagov\_\_govuk-infrastructure
* PaloAltoNetworks\_\_terraform-azurerm-vmseries-modules
* kubernetes\_\_k8s.io
* rancherfederal\_\_rke2-aws-tf
* magma\_\_magma
* RhinoSecurityLabs\_\_cloudgoat
* microsoft\_\_azure\_arc
* oracle-terraform-modules\_\_terraform-oci-oke
* ministryofjustice\_\_cloud-platform-infrastructure
* pingcap\_\_tidb-operator
* camptocamp\_\_devops-stack

1. **Caractéristiques extraites :**

Filepath, Workingdirectory, Entityname, Blocktype, Sourcename, Isterraforminit, Hash, Author, Date, Message, Operator, Major, Minor, Patch.

1. **L’Extraction des Versions des Fournisseurs depuis le Registre Terraform**

import pandas as pd

import requests

import json

# Load the CSV file

file\_path = '/content/Azure\_\_az-hop\_\_versioning\_\_data\_\_1.2.2\_TEST\_TEST\_TEST\_file.csv'

data = pd.read\_csv(file\_path)

# Get unique providers from the "SourceName" column

unique\_providers = data['sourceName'].unique()

# Function to fetch versions and their release dates from the Terraform Registry

def fetch\_versions\_and\_dates(provider):

    url = f"https://registry.terraform.io/v1/providers/{provider}"

    try:

        response = requests.get(url)

        if response.status\_code == 200:

            versions = response.json()["versions"]

            details = []

            for version in versions:

                version\_url = f"{url}/{version}"

                version\_response = requests.get(version\_url)

                if version\_response.status\_code == 200:

                    version\_data = version\_response.json()

                    details.append({

                        "version": version\_data["version"],

                        "published\_at": version\_data["published\_at"]

                    })

            return details

        else:

            print(f"Failed to fetch data for {provider}. Status code: {response.status\_code}")

    except Exception as e:

        print(f"Error fetching data for {provider}: {e}")

# Loop through each provider and print their versions and release dates

for provider in unique\_providers:

    print(f"\nFetching versions for {provider}")

    version\_details = fetch\_versions\_and\_dates(provider)

    if version\_details:

        for detail in version\_details:

            print(f"Version: {detail['version']}, Released: {detail['published\_at']}")

    else:

        print(f"No details found for {provider}")

Le script précèdent va dans le projet ( fichier csv d’un certain projet donné) a la colonne sourcename et cherche le nom du provider dans terraform registry et quand il le trouve il ramène toutes les versions existantes de ce fournisseur voici **un exemple** :

Fetching versions for hashicorp/azurerm

Version: 0.1.0, Released: 2017-07-12T17:43:53Z

Version: 0.1.1, Released: 2017-07-12T17:43:57Z

Version: 0.1.2, Released: 2017-07-12T17:44:01Z

Version: 0.1.3, Released: 2017-07-21T15:27:00Z

Version: 0.1.4, Released: 2017-07-26T21:34:29Z

Version: 0.1.5, Released: 2017-08-09T13:06:29Z

Version: 0.1.6, Released: 2017-08-31T13:43:12Z

Version: 0.1.7, Released: 2017-09-11T08:10:42Z

Version: 0.2.0, Released: 2017-09-15T19:37:20Z

Version: 0.2.1, Released: 2017-09-25T10:15:24Z

Version: 0.2.2, Released: 2017-09-28T14:08:14Z

Version: 0.3.0, Released: 2017-10-17T18:15:00Z

Version: 0.3.1, Released: 2017-10-21T01:17:47Z

…

Fetching versions for hashicorp/random

Version: 0.1.0, Released: 2017-07-12T19:48:48Z

Version: 1.0.0, Released: 2017-09-15T20:54:27Z

Version: 1.1.0, Released: 2017-12-01T01:29:27Z

Version: 1.2.0, Released: 2018-04-03T22:58:06Z

Version: 1.3.0, Released: 2018-05-21T22:07:34Z

Version: 1.3.1, Released: 2018-05-22T19:20:03Z

Version: 2.0.0, Released: 2018-08-15T23:56:09Z

Version: 2.1.0, Released: 2019-03-20T02:38:44Z

Version: 2.1.1, Released: 2019-04-12T00:16:45Z

Version: 2.1.2, Released: 2019-04-30T21:54:26Z

Version: 2.2.0, Released: 2019-08-08T17:25:57Z

…

1. **Classement des fournisseurs par utilisation :**

Pour savoir quel sont les fournisseurs les plus utilisés dans notre étude on a développé un script qui extrait les données a partir d’un autres fichier Excel qu’on va en parler juste après.

import pandas as pd

# Load the source Excel file

source\_file\_path = 'project\_provider\_upgrades\_downgrades results per provider.xlsx'

source\_data = pd.read\_excel(source\_file\_path)

# Aggregate data to count the number of projects each provider is used in

provider\_aggregation = source\_data.groupby('provider').agg({

    'project': pd.Series.nunique  # Counts unique projects per provider

}).rename(columns={'project': 'Projects'})

# Calculate the total number of projects in the dataset

total\_projects = source\_data['project'].nunique()

# Calculate the percentage of total projects for each provider

provider\_aggregation['Percentage'] = (provider\_aggregation['Projects'] / total\_projects) \* 100

# Reset the index to turn the 'provider' index into a column

provider\_aggregation.reset\_index(inplace=True)

# Save the processed data back into an Excel file

output\_file\_path = ' Provider \_ project \_ percentage excel.xlsx'

provider\_aggregation.to\_excel(output\_file\_path, index=False)

" https://docs.google.com/spreadsheets/d/1iYwQvXbnBD-K9-J8ohjZuz8hnFLn3t3C/edit?usp=sharing&ouid=105140662728749325776&rtpof=true&sd=true "

1. **Et maintenant voici le script pour extraire les données suivantes :**

Project, provider, upgrades, downgrades, total upgrades in the project, total downgrades in the project, total changes in the project, total upgrades of the provider across all projects, total downgrades of the provider across all projects, total changes of the provider across all projects.

import pandas as pd

import os

import glob

def calculate\_changes(df):

    df = df.sort\_values(by='date')

    df[['prev\_major', 'prev\_minor', 'prev\_patch']] = df[['major', 'minor', 'patch']].shift(1)

    def compare\_versions(row):

        if row['major'] > row['prev\_major']:

            return 'upgrade'

        elif row['major'] < row['prev\_major']:

            return 'downgrade'

        else:

            if row['minor'] > row['prev\_minor']:

                return 'upgrade'

            elif row['minor'] < row['prev\_minor']:

                return 'downgrade'

            else:

                if row['patch'] > row['prev\_patch']:

                    return 'upgrade'

                elif row['patch'] < row['prev\_patch']:

                    return 'downgrade'

                else:

                    return 'no change'

    df['change\_type'] = df.apply(compare\_versions, axis=1)

    return df

def analyze\_csv(file\_path):

    data = pd.read\_csv(file\_path)

    data = data[data['blockType'] == 'provider']

    data = data[data['sourceName'] != '\_']

    data = data[~((data['major'] == -1) & (data['minor'] == -1) & (data['patch'] == -1))]

    data.loc[:, 'patch'] = data['patch'].replace(-1, 0)

    data.loc[:, ['major', 'minor', 'patch']] = data[['major', 'minor', 'patch']].fillna(0).astype(int)

    if data.empty:

        return pd.DataFrame(), os.path.basename(file\_path).split('versioning')[0].strip('\_')

    results = data.groupby(['workingDirectory', 'sourceName']).apply(calculate\_changes).reset\_index(drop=True)

    if 'change\_type' not in results.columns:

        results['change\_type'] = pd.Series(['no change'] \* len(results))

    summary = results.groupby(['sourceName', 'change\_type']).size().unstack(fill\_value=0)

    summary = summary.reindex(columns=['upgrade', 'downgrade'], fill\_value=0)

    project\_name = os.path.basename(file\_path).split('versioning')[0].strip('\_')

    return summary, project\_name

directory\_path = '/content'

file\_paths = glob.glob(os.path.join(directory\_path, '\*.csv'))

all\_projects\_summary = {}

for file\_path in file\_paths:

    summary, project\_name = analyze\_csv(file\_path)

    if project\_name in all\_projects\_summary:

        all\_projects\_summary[project\_name] = all\_projects\_summary[project\_name].add(summary, fill\_value=0)

    else:

        all\_projects\_summary[project\_name] = summary

consolidated\_results = []

total\_per\_provider = pd.DataFrame()

for project, summary in all\_projects\_summary.items():

    total\_per\_provider = total\_per\_provider.add(summary, fill\_value=0)

total\_per\_provider['total\_changes'] = total\_per\_provider['upgrade'] + total\_per\_provider['downgrade']

for project, summary in all\_projects\_summary.items():

    for provider, row in summary.iterrows():

        consolidated\_results.append({

            'project': project,

            'provider': provider,

            'upgrades': int(row['upgrade']),

            'downgrades': int(row['downgrade']),

            'total upgrades in the project': summary['upgrade'].sum(),

            'total downgrades in the project': summary['downgrade'].sum(),

            'total changes in the project': summary['upgrade'].sum() + summary['downgrade'].sum(),

            'total upgrades of the provider across all projects': total\_per\_provider.at[provider, 'upgrade'],

            'total downgrades of the provider across all projects': total\_per\_provider.at[provider, 'downgrade'],

            'total changes of the provider across all projects': total\_per\_provider.at[provider, 'total\_changes']

        })

df\_results = pd.DataFrame(consolidated\_results)

output\_path = '/content/project\_provider\_upgrades\_downgrades.csv'

df\_results.to\_csv(output\_path, index=False)

print(f"Data written to {output\_path}")

" https://docs.google.com/spreadsheets/d/1-o-08g8sk-17w3aac9H\_5V44rGq6MJfL/edit?usp=sharing&ouid=105140662728749325776&rtpof=true&sd=true "

1. **Total des changements par projet**

Une image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquement

Une image contenant texte, diagramme, ligne, Police

Description générée automatiquement

Une image contenant texte, diagramme, Tracé, Police

Description générée automatiquement

Ces graphes ont été extraits à partir du fichier Excel qui contient le nombre de changements, upgrades et downgrades par projets

"https://docs.google.com/spreadsheets/d/1sCJ99WocGYTnxUsPQN2kjaD879bBI6Fx/edit?usp=sharing&ouid=105140662728749325776&rtpof=true&sd=true"

Qui lui-même été extrait à partir du fichier « project\_provider\_upgrades\_downgrades results per provider.xlsx » en utilisant ce script :

import pandas as pd

# Load the detailed data

file\_path = '/content/project\_provider\_upgrades\_downgrades results per provider.xlsx'

data = pd.read\_excel(file\_path)

# Aggregate the data by project

aggregated\_data = data.groupby('project').agg({

    'upgrades': 'sum',

    'downgrades': 'sum'

})

aggregated\_data['total changes'] = aggregated\_data['upgrades'] + aggregated\_data['downgrades']

# Sort the data by 'total changes' in descending order

sorted\_data = aggregated\_data.sort\_values('total changes', ascending=False).reset\_index()

# Save the sorted aggregated data to a new Excel file

output\_file\_path = '/content/upgrades, downgrades and changes per project (sorted).xlsx'

sorted\_data.to\_excel(output\_file\_path, index=False)

print("Data aggregated and saved successfully.")

1. **Et maintenant les statistiques des Changements de Versions**

À partir du fichier « upgrades, downgrades and changes per project (sorted).xlsx » on a extrait ce graphe ainsi que l tableau de statistique

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

**En utilisant ce script :**

import pandas as pd

# Define the data

data = [

    {"Project": "Azure\_\_Avere", "Upgrades": 646, "Downgrades": 1, "Changes": 647},

    {"Project": "pluralsh\_\_plural-artifacts", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "SUSE\_\_ha-sap-terraform-deployments", "Upgrades": 22, "Downgrades": 7, "Changes": 29},

    {"Project": "GiganticMinecraft\_\_seichi\_infra", "Upgrades": 50, "Downgrades": 11, "Changes": 61},

    {"Project": "alphagov\_\_govuk-aws", "Upgrades": 9, "Downgrades": 2, "Changes": 11},

    {"Project": "kubernetes-sigs\_\_kubespray", "Upgrades": 13, "Downgrades": 0, "Changes": 13},

    {"Project": "Azure\_\_az-hop", "Upgrades": 51, "Downgrades": 16, "Changes": 67},

    {"Project": "CDCgov\_\_prime-simplereport", "Upgrades": 156, "Downgrades": 0, "Changes": 156},

    {"Project": "clong\_\_DetectionLab", "Upgrades": 6, "Downgrades": 1, "Changes": 7},

    {"Project": "2i2c-org\_\_infrastructure", "Upgrades": 16, "Downgrades": 4, "Changes": 20},

    {"Project": "cloudfoundry\_\_bosh-bootloader", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "deckhouse\_\_deckhouse", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "zenml-io\_\_mlstacks", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "bridgecrewio\_\_terragoat", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "apache\_\_beam", "Upgrades": 2, "Downgrades": 0, "Changes": 2},

    {"Project": "compiler-explorer\_\_infra", "Upgrades": 3, "Downgrades": 0, "Changes": 3},

    {"Project": "cattle-ops\_\_terraform-aws-gitlab-runner", "Upgrades": 319, "Downgrades": 11, "Changes": 330},

    {"Project": "aws-observability\_\_terraform-aws-observability-accelerator", "Upgrades": 14, "Downgrades": 2, "Changes": 16},

    {"Project": "cookpad\_\_terraform-aws-eks\_\_versioning", "Upgrades": 50, "Downgrades": 47, "Changes": 97},

    {"Project": "rust-lang\_\_simpleinfra", "Upgrades": 59, "Downgrades": 13, "Changes": 72},

    {"Project": "ministryofjustice\_\_modernisation-platform", "Upgrades": 279, "Downgrades": 100, "Changes": 379},

    {"Project": "ManagedKube\_\_kubernetes-ops", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "wireapp\_\_wire-server-deploy", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "google\_\_go-cloud", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "kube-hetzner\_\_terraform-hcloud-kube-hetzner", "Upgrades": 39, "Downgrades": 19, "Changes": 58},

    {"Project": "Optum\_\_dce", "Upgrades": 1, "Downgrades": 0, "Changes": 1},

    {"Project": "nasa\_\_cumulus", "Upgrades": 142, "Downgrades": 105, "Changes": 247},

    {"Project": "iits-consulting\_\_terraform-opentelekomcloud-project-factory", "Upgrades": 27, "Downgrades": 0, "Changes": 27},

    {"Project": "nebari-dev\_\_nebari", "Upgrades": 38, "Downgrades": 1, "Changes": 39},

    {"Project": "awslabs\_\_data-on-eks", "Upgrades": 14, "Downgrades": 9, "Changes": 23},

    {"Project": "uyuni-project\_\_sumaform", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "kbst\_\_terraform-kubestack", "Upgrades": 22, "Downgrades": 1, "Changes": 23},

    {"Project": "splunk\_\_attack\_range", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "GoogleCloudPlatform\_\_hpc-toolkit", "Upgrades": 287, "Downgrades": 199, "Changes": 486},

    {"Project": "Worklytics\_\_psoxy", "Upgrades": 19, "Downgrades": 6, "Changes": 25},

    {"Project": "chanzuckerberg\_\_cztack", "Upgrades": 3, "Downgrades": 0, "Changes": 3},

    {"Project": "Azure\_\_sap-automation", "Upgrades": 21, "Downgrades": 2, "Changes": 23},

    {"Project": "alphagov\_\_govuk-infrastructure", "Upgrades": 78, "Downgrades": 54, "Changes": 132},

    {"Project": "PaloAltoNetworks\_\_terraform-azurerm-vmseries-modules", "Upgrades": 89, "Downgrades": 41, "Changes": 130},

    {"Project": "kubernetes\_\_k8s.io", "Upgrades": 253, "Downgrades": 63, "Changes": 316},

    {"Project": "rancherfederal\_\_rke2-aws-tf", "Upgrades": 0, "Downgrades": 3, "Changes": 3},

    {"Project": "magma\_\_magma", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "RhinoSecurityLabs\_\_cloudgoat", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "microsoft\_\_azure\_arc", "Upgrades": 5, "Downgrades": 3, "Changes": 8},

    {"Project": "oracle-terraform-modules\_\_terraform-oci-oke", "Upgrades": 108, "Downgrades": 98, "Changes": 206},

    {"Project": "ministryofjustice\_\_cloud-platform-infrastructure", "Upgrades": 278, "Downgrades": 126, "Changes": 404},

    {"Project": "pingcap\_\_tidb-operator", "Upgrades": 0, "Downgrades": 0, "Changes": 0},

    {"Project": "camptocamp\_\_devops-stack", "Upgrades": 25, "Downgrades": 11, "Changes": 36},

]

# Create DataFrame

df = pd.DataFrame(data)

# Calculate descriptive statistics for Upgrades, Downgrades, and Changes

stats\_upgrades = df['Upgrades'].describe(percentiles=[.25, .5, .75])

stats\_downgrades = df['Downgrades'].describe(percentiles=[.25, .5, .75])

stats\_changes = df['Changes'].describe(percentiles=[.25, .5, .75])

# Add total to the stats

stats\_upgrades['total'] = df['Upgrades'].sum()

stats\_downgrades['total'] = df['Downgrades'].sum()

stats\_changes['total'] = df['Changes'].sum()

# Print the statistics

print("Descriptive Statistics for Upgrades:")

print(stats\_upgrades)

print("\nDescriptive Statistics for Downgrades:")

print(stats\_downgrades)

print("\nDescriptive Statistics for Changes:")

print(stats\_changes)

1. **Technical Lag**

Avant d’avoir le graphe du Technical Lag

* 1. On a extrait 48 fichiers contenant les données nécessaires pour voir le Technical Lag

Voici i exemple de données sur fichier csv « https://drive.google.com/file/d/1B95i00y9AF3za2ZTPAMmIh1OA7DpBCbW/view?usp=sharing»

**En utilisant ce script**

import datetime

import requests

import pandas as pd

from packaging.version import parse

def fetch\_versions(provider):

    # Format the URL for the specific provider

    url = f"https://registry.terraform.io/v1/providers/{provider}/versions"

    # Make a GET request to the API

    try:

        response = requests.get(url)

        response.raise\_for\_status()  # Raises an exception for HTTP errors

        data = response.json()

        # Collect all version numbers

        versions = [version['version'] for version in data['versions']]

        versions.sort(key=parse, reverse=False)  # Sort versions in ascending order

        return versions

    except requests.RequestException as e:

        print(f"Failed to fetch versions for {provider}: {e}")

        return []

def get\_provider\_versions\_from\_csv(csv\_file):

    # Read the CSV file

    df = pd.read\_csv(csv\_file)

    # Filter for rows where blockType is 'provider'

    df = df[df['blockType'] == 'provider']

    # Get unique providers from the sourceName column

    providers = df['sourceName'].unique()

    # Dictionary to hold provider versions

    provider\_versions = {}

    # Loop through each provider and fetch versions

    for provider in providers:

        versions = fetch\_versions(provider)

        provider\_versions[provider] = versions

    return provider\_versions

# Usage

csv\_file = '/content/Azure\_\_az-hop\_\_versioning\_\_data\_\_1.2.2\_TEST\_TEST\_TEST\_file.csv'

versions = get\_provider\_versions\_from\_csv(csv\_file)

print(versions)

def find\_used\_version\_tilde\_greater\_than(declared\_version, versions, provider, commit\_date):

    ver = declared\_version.split('.')

    missing\_patch = False

    missing\_minor = False

    used\_version = ver[-1]

    v = '.'.join(ver[:-1])

    if used\_version == '-1' :

      if ver[1] == '-1' :

        return find\_used\_version\_greather\_than(declared\_version, versions, provider, commit\_date)

      used\_version = ver[1]

      ver = ver[0]

      missing\_patch = True

    for version in versions:

        if version.startswith(v):

            if missing\_patch:

                if int(used\_version) >= int(version.split('.')[-2]):

                  continue

                else:

                    if get\_version\_date(provider, version) > commit\_date:

                        break

                    used\_version = int(version.split('.')[-2])

            else :

                if int(used\_version) >= int(version.split('.')[-1]):

                  continue

                else:

                    if get\_version\_date(provider, version) > commit\_date:

                        break

                    used\_version = int(version.split('.')[-1])

    if(missing\_patch):

      print(used\_version)

      used\_version = declared\_version.split('.')[0]+'.'+ str(used\_version)+ '.0'

    else:

      used\_version = declared\_version[:-1] + str(used\_version)

    return used\_version

!pip install packaging

from packaging import version

def find\_used\_version\_smaller\_than(declared\_version, provider\_versions, provider, commit\_date):

  filtered\_versions = [v for v in provider\_versions if version.parse(v) < version.parse(declared\_version) and get\_version\_date(provider, v) <= commit\_date]

  return filtered\_versions[-1]

def find\_used\_version\_greather\_than(declared\_version, provider\_versions, provider, commit\_date):

    latest\_version = None

    for version in provider\_versions:

        date = get\_version\_date(provider, version)

        if date:

            commit\_date\_dt = datetime.datetime.strptime(commit\_date, "%Y-%m-%d %H:%M:%S")

            version\_date\_dt = datetime.datetime.strptime(date, "%Y-%m-%dT%H:%M:%SZ")

            if commit\_date\_dt > version\_date\_dt:

                latest\_version = version

                latest\_version\_date = date

            else:

                break

    return latest\_version  # Ensures two values are always returned

def fetch\_versions(provider):

    # Format the URL for the specific provider

    url = f"https://registry.terraform.io/v1/providers/{provider}/versions"

    # Make a GET request to the API

    try:

        response = requests.get(url)

        response.raise\_for\_status()  # Raises an exception for HTTP errors

        data = response.json()

        # Collect all version numbers

        versions = [version['version'] for version in data['versions']]

        versions.sort(key=parse, reverse=False)  # Sort versions in ascending order

        return versions

    except requests.RequestException as e:

        print(f"Failed to fetch versions for {provider}: {e}")

        return []

import requests

import datetime

from functools import lru\_cache

@lru\_cache(maxsize=1000)

def get\_version\_date(provider, version):

    url = f"https://registry.terraform.io/v1/providers/{provider}/{version}"

    try:

        response = requests.get(url)

        response.raise\_for\_status()

        data = response.json()

        return data["published\_at"]

    except requests.RequestException as e:

        print(f"Failed to fetch version for {provider}: {e}")

        return None

def get\_latest\_available\_version\_at\_date(provider, provider\_versions, commit\_date):

    latest\_version = None

    latest\_version\_date = None

    for version in provider\_versions:

        date = get\_version\_date(provider, version)

        if date:

            commit\_date\_dt = datetime.datetime.strptime(commit\_date, "%Y-%m-%d %H:%M:%S")

            version\_date\_dt = datetime.datetime.strptime(date, "%Y-%m-%dT%H:%M:%SZ")

            if commit\_date\_dt > version\_date\_dt:

                latest\_version = version

                latest\_version\_date = date

            else:

                break

    return (latest\_version, latest\_version\_date)  # Ensures two values are always returned

def apply\_used\_version(row):

    global providers\_versions

    x = ''

    if(row['operator'] == '='):

      x =row['declared\_version']

    elif(row['operator'] == '~>'):

      x =find\_used\_version\_tilde\_greater\_than(row['declared\_version'], providers\_versions[row['sourceName']], row['sourceName'], row['date'])

    # elif(row['operator'] == '>='):

    #   x =find\_used\_version\_greather\_than(row['declared\_version'], providers\_versions[row['sourceName']], row['sourceName'], row['date'])

    # elif(row['operator'] == '<'):

    #   x =find\_used\_version\_smaller\_than(row['declared\_version'], providers\_versions[row['sourceName']], row['sourceName'], row['date'])

    return x

def main():

    global providers\_versions

    # Load the data

    file\_path = '/content/Azure\_\_az-hop\_\_versioning\_\_data\_\_1.2.2\_TEST\_TEST\_TEST\_file.csv'

    data = pd.read\_csv(file\_path)

    data = data[data['blockType'] == 'provider']

    # Get unique providers and fetch their versions

    providers\_versions = {provider: fetch\_versions(provider) for provider in data['sourceName'].unique()}

    # Apply get\_latest\_available\_version\_at\_date function and expand results into two columns

    data[['latest\_available\_version\_at\_date', 'date\_of\_latest\_available\_version\_at\_date']] = data.apply(

        lambda row: get\_latest\_available\_version\_at\_date(

            row['sourceName'],

            providers\_versions[row['sourceName']],

            row['date']

        ),

        axis=1,

        result\_type='expand'

    )

    data['declared\_version'] = data.apply(lambda row: f"{row['major']}.{row['minor']}.{row['patch']}", axis=1)

    data['used\_version'] = data.apply(lambda row : apply\_used\_version(row), axis=1)

    # Verify function 'get\_version\_date' is correctly applied to populate 'release date of used version'

    data['release date of used version'] = data.apply(

        lambda row: get\_version\_date(row['sourceName'], row['used\_version']),

        axis=1

    )

    # Debugging: Print DataFrame columns to confirm column addition

    print("Columns after processing:", data.columns.tolist())

    print("Sample release dates:", data['release date of used version'].head())

    # Ensure columns are converted to datetime format

    data['date\_of\_latest\_available\_version\_at\_date'] = pd.to\_datetime(data['date\_of\_latest\_available\_version\_at\_date'], errors='coerce')

    data['release date of used version'] = pd.to\_datetime(data['release date of used version'], errors='coerce')

    # Calculate 'DELTA3'

    data['DELTA3'] = (data['date\_of\_latest\_available\_version\_at\_date'] - data['release date of used version']).dt.days

    # Debugging: Print sample data for DELTA3 to verify correct calculations

    print("Sample data for DELTA3:", data[['latest\_available\_version\_at\_date', 'date\_of\_latest\_available\_version\_at\_date', 'DELTA3']].head())

    # Save the selected columns to a CSV file

    columns\_to\_display = ['workingDirectory', 'sourceName', 'date', 'operator', 'declared\_version', 'used\_version', 'release date of used version', 'latest\_available\_version\_at\_date', 'date\_of\_latest\_available\_version\_at\_date', 'DELTA3']

    data[columns\_to\_display].to\_csv('/content/tested\_on\_Azure\_\_Avere.csv  ', index=False)

    # Print final DataFrame to console for verification

    print(data[columns\_to\_display])

if \_\_name\_\_ == "\_\_main\_\_":

    main()

* 1. **À partir des fichiers csv extrait à partir de tout les 48 projets on réussi à avoir le graphe du Technical Lag**

Une image contenant texte, Tracé, ligne, diagramme

Description générée automatiquement

**En utilisant ce script**

import pandas as pd

import matplotlib.pyplot as plt

import glob

# Step 1: Load and combine CSV files

path = '/content'  # CSV files directory

all\_files = glob.glob(path + "/\*.csv")

df = pd.concat((pd.read\_csv(f) for f in all\_files))

# Filter out rows where 'operator' is "\_" or 'sourceName' is "\_"

df = df[(df['operator'] != '\_') & (df['sourceName'] != '\_')]

# Step 2: Preprocess data

df['date'] = pd.to\_datetime(df['date'])

df['date\_of\_latest\_available\_version\_at\_date'] = pd.to\_datetime(df['date\_of\_latest\_available\_version\_at\_date'])

df['month'] = df['date'].dt.to\_period('M')

# Step 3: Aggregate data

monthly\_data = df.groupby('month')['DELTA3'].agg(['median', 'quantile', lambda x: x.quantile(0.25), lambda x: x.quantile(0.75)]).rename(columns={'quantile': 'mean', '<lambda\_0>': 'q1', '<lambda\_1>': 'q3'})

# Step 4: Plot

plt.fill\_between(monthly\_data.index.astype(str), monthly\_data['q1'], monthly\_data['q3'], color='green', alpha=0.3)

plt.plot(monthly\_data.index.astype(str), monthly\_data['median'], color='green', marker='o', linestyle='-')

plt.plot(monthly\_data.index.astype(str), monthly\_data['mean'], color='green', linestyle='--')

plt.xticks(rotation=90)  # Rotate date labels for better visibility

plt.xlabel('Date')

plt.ylabel('Technical Lag (days)')

plt.title('Monthly Distribution of Technical Lag')

plt.grid(True)

plt.tight\_layout()

plt.show()

**Après ça, on a extrait les graphes Technical Lag de quelques fournisseurs**

Une image contenant texte, ligne, diagramme, Tracé

Description générée automatiquement

**En utilisant ce script**

import pandas as pd

import matplotlib.pyplot as plt

import glob

# Step 1: Load and combine CSV files

path = '/content'  # CSV files directory

all\_files = glob.glob(path + "/\*.csv")

df = pd.concat((pd.read\_csv(f) for f in all\_files))

# Filter to keep only rows where 'sourceName' is "hashicorp/aws" and 'operator' is not "\_"

df = df[(df['sourceName'] == 'hashicorp/aws) & (df['operator'] != '\_')]

# Step 2: Preprocess data

df['date'] = pd.to\_datetime(df['date'])

df['date\_of\_latest\_available\_version\_at\_date'] = pd.to\_datetime(df['date\_of\_latest\_available\_version\_at\_date'])

df['month'] = df['date'].dt.to\_period('M')

# Step 3: Aggregate data

monthly\_data = df.groupby('month')['DELTA3'].agg(['median', 'quantile', lambda x: x.quantile(0.25), lambda x: x.quantile(0.75)]).rename(columns={'quantile': 'mean', '<lambda\_0>': 'q1', '<lambda\_1>': 'q3'})

# Step 4: Plot

plt.fill\_between(monthly\_data.index.astype(str), monthly\_data['q1'], monthly\_data['q3'], color='green', alpha=0.3)

plt.plot(monthly\_data.index.astype(str), monthly\_data['median'], color='green', marker='o', linestyle='-')

plt.plot(monthly\_data.index.astype(str), monthly\_data['mean'], color='green', linestyle='--')

plt.xticks(rotation=90)  # Rotate date labels for better visibility

plt.xlabel('Date')

plt.ylabel('Technical Lag (days)')

plt.title('Monthly Distribution of Technical Lag for hashicorp/aws)

plt.grid(True)

plt.tight\_layout()

plt.show()