STEPS TO RUN THE PTHON SCRIPT AND GEENRATE THE TABLE

Sudo apt update

Sudo apt install -y python3 python3-pip

Sudo pip3 install happybase faker

Sudo mkdir -p /app

sudo chmod 777 /app

cd /app

Fisrst copy the python script inside the container

cat > data\_generator.py << 'EOL'

import random

import hashlib

from datetime import datetime, timedelta

from faker import Faker

import happybase

import logging

# Configure logging

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

class WebDataGenerator:

    def \_\_init\_\_(self, host='localhost', port=9090):

        """Initialize the data generator with Faker."""

        self.faker = Faker()

        self.connection = happybase.Connection(host=host, port=port)

        self.table\_name = 'webtable'

        # Fixed domains for the task

        self.domains = [

            'example.com',

            'test.org',

            'demo.net',

            'sample.edu',

            'tech.io'

        ]

        # Common URL paths with categories

        self.paths = {

            'main': ['/home', '/about', '/contact'],

            'products': ['/products', '/services', '/pricing'],

            'content': ['/blog', '/news', '/articles'],

            'user': ['/login', '/register', '/profile'],

            'support': ['/help', '/faq', '/support']

        }

        # Content size templates (in paragraphs)

        self.content\_sizes = {

            'small': 2,    # 2 paragraphs

            'medium': 5,   # 5 paragraphs

            'large': 10    # 10 paragraphs

        }

        # HTML templates with more realistic content

        self.html\_templates = [

            '''<html>

                <head>

                    <title>{title}</title>

                    <meta name="description" content="{description}">

                    <meta name="keywords" content="{keywords}">

                </head>

                <body>

                    <header>

                        <h1>{title}</h1>

                        <nav>{navigation}</nav>

                    </header>

                    <main>

                        <article>

                            <h2>{subtitle}</h2>

                            {content}

                        </article>

                    </main>

                    <footer>

                        <p>{footer}</p>

                    </footer>

                </body>

            </html>''',

            '''<html>

                <head>

                    <title>{title}</title>

                    <meta name="description" content="{description}">

                </head>

                <body>

                    <div class="container">

                        <nav>{navigation}</nav>

                        <section>

                            <h1>{title}</h1>

                            <div class="content">

                                {content}

                            </div>

                        </section>

                    </div>

                </body>

            </html>'''

        ]

    def generate\_row\_key(self, domain, url\_path):

        """Generate salted row key: [salt\_bucket]:[reverse\_domain]:[reversed\_url\_path]"""

        salt\_input = f"{domain}{url\_path}"

        salt\_bucket = int(hashlib.md5(salt\_input.encode()).hexdigest(), 16) % 10

        reverse\_domain = '.'.join(domain.split('.')[::-1])

        reversed\_url\_path = url\_path[::-1]

        return f"{salt\_bucket}:{reverse\_domain}:{reversed\_url\_path}"

    def generate\_metadata(self, domain, url\_path, creation\_date):

        """Generate realistic metadata for a webpage."""

        return {

            'title': f"{self.faker.catch\_phrase()} - {domain}",

            'status': random.choice([200, 200, 200, 301, 302, 404, 500]),  # Weighted towards 200

            'last\_modified': creation\_date.isoformat(),

            'content\_type': random.choice(['text/html', 'application/xhtml+xml']),

            'language': random.choice(['en-US', 'en-GB', 'es-ES', 'fr-FR', 'de-DE'])

        }

    def generate\_html\_content(self, metadata, size\_category):

        """Generate realistic HTML content with specified size."""

        template = random.choice(self.html\_templates)

        num\_paragraphs = self.content\_sizes[size\_category]

        # Generate content with specified number of paragraphs

        content = '\n'.join([f'<p>{self.faker.paragraph()}</p>' for \_ in range(num\_paragraphs)])

        return template.format(

            title=metadata['title'],

            description=self.faker.sentence(),

            subtitle=self.faker.sentence(),

            content=content,

            footer=self.faker.company(),

            keywords=', '.join(self.faker.words(nb=5)),

            navigation=self.faker.paragraph(nb\_sentences=1)

        )

    def generate\_links(self, domain, url\_path, all\_pages):

        """Generate interconnected links between pages."""

        links = []

        num\_links = random.randint(1, 3)  # 1-3 links per page

        # Get all possible target pages

        possible\_targets = [

            (d, p) for d, p in all\_pages

            if d != domain or p != url\_path  # Don't link to self

        ]

        if possible\_targets:

            # Select random targets

            targets = random.sample(possible\_targets, min(num\_links, len(possible\_targets)))

            for target\_domain, target\_path in targets:

                links.append(f"https://{target\_domain}{target\_path}")

        return links

    def store\_webpage(self, domain, url\_path, creation\_date, size\_category, all\_pages):

        """Generate and store a complete webpage entry."""

        metadata = self.generate\_metadata(domain, url\_path, creation\_date)

        html\_content = self.generate\_html\_content(metadata, size\_category)

        outlinks = self.generate\_links(domain, url\_path, all\_pages)

        row\_key = self.generate\_row\_key(domain, url\_path)

        table = self.connection.table(self.table\_name)

        data = {

            'content:html': html\_content,

            'metadata:title': metadata['title'],

            'metadata:status': str(metadata['status']),

            'metadata:last\_modified': metadata['last\_modified'],

            'metadata:content\_type': metadata['content\_type'],

            'metadata:language': metadata['language'],

            'metadata:content\_length': str(len(html\_content))

        }

        if outlinks:

            data['outlinks:list'] = ','.join(outlinks)

        table.put(row\_key, data)

        logger.info(f"Stored webpage: {domain}{url\_path} (Size: {size\_category}, Date: {creation\_date.date()})")

        return row\_key

    def generate\_test\_data(self):

        """Generate a structured dataset with specific characteristics."""

        try:

            # Generate list of all pages to create

            all\_pages = []

            for domain in self.domains:

                for category, paths in self.paths.items():

                    for path in paths:

                        all\_pages.append((domain, path))

            # Shuffle pages to randomize creation order

            random.shuffle(all\_pages)

            # Generate pages with different creation dates

            base\_date = datetime.now() - timedelta(days=30)  # Start 30 days ago

            for i, (domain, path) in enumerate(all\_pages):

                # Distribute creation dates over the past 30 days

                creation\_date = base\_date + timedelta(days=i \* 30 / len(all\_pages))

                # Assign size categories

                if i < len(all\_pages) // 3:

                    size\_category = 'small'

                elif i < 2 \* len(all\_pages) // 3:

                    size\_category = 'medium'

                else:

                    size\_category = 'large'

                self.store\_webpage(domain, path, creation\_date, size\_category, all\_pages)

            logger.info(f"Successfully generated {len(all\_pages)} test pages")

        finally:

            self.connection.close()

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

    generator = WebDataGenerator()

    generator.generate\_test\_data()

EOL

Sudo chmod 777 data\_genrator.py

ADD the table creation script

cat > create\_table.hbase << 'EOL'

# Disable table if exists

disable 'webtable'

drop 'webtable'

# Create table with optimized column families

create 'webtable',

  {NAME => 'content', VERSIONS => 3, TTL => 7776000, COMPRESSION => 'GZ', BLOOMFILTER => 'ROW', IN\_MEMORY => 'false'},

  {NAME => 'metadata', VERSIONS => 1, TTL => -1, COMPRESSION => 'SNAPPY', BLOOMFILTER => 'ROW', IN\_MEMORY => 'true'},

  {NAME => 'outlinks', VERSIONS => 2, TTL => 15552000, COMPRESSION => 'NONE', BLOOMFILTER => 'NONE', IN\_MEMORY => 'false'},

  {NAME => 'inlinks', VERSIONS => 2, TTL => 15552000, COMPRESSION => 'NONE', BLOOMFILTER => 'NONE', IN\_MEMORY => 'false'}

# Set table attributes for performance

alter 'webtable', {METHOD => 'table\_att', MAX\_FILESIZE => '10737418240'}  # 10GB max region size

alter 'webtable', {METHOD => 'table\_att', SPLIT\_POLICY => 'org.apache.hadoop.hbase.regionserver.ConstantSizeRegionSplitPolicy'}

# Enable table

enable 'webtable'

# Verify table creation

describe 'webtable'

EOL

hbase shell -n <<< "scan 'webtable', {LIMIT => 5}"