Contract testing with Pact, Pact Broker and ChatGPT

Contract testing though existed for over twenty years. The company behind the Pact Foundation is called DiUS Computing Pty Ltd. DiUS Computing was founded in 2004 in Melbourne, Australia. Recently, the contract testing started to take broader acceptance because Microservices became more dominant and because the DevOps teams realized that E-2-E testing becomes excessively time consuming not because of detecting bugs in high level business processes but because most problems detected in E-2-E ironically are low level coding and in mapping bugs.

Contract Testing is not functional testing. The Pact Framework does not support the formulas in the contracts. It is focused on syntax and completeness of interfaces between pairs of servers, one pair at a time.

Importantly, today's Contract testing became Client driven, as opposed to Provider driven. Being Client driven aligns better with principles of the Agile development, that is it allows the DevOps team to sharper focus on the purpose of the entire system.

It has been recognized, as well, that Contract testing faces its own challenges.

1. It requires involvement of Client development or maintenance teams. These teams often:
    a. Believe that Provider needs to work before they invest their time.
    b. Do not exist yet.
    c. Will never come to existence because Client is a legacy system no one will work on.
    d. Will never come to existence because Client is everyone else on our planet.
2. Confidentiality induced need for data masking and anonymization results in a perception that any QA-ing on Client side in early stages would result in a substantial and additional cost if implement.

However, the above challenges need to be looked at closer to determine if they actually exist in a particular project. Or more precisely, if any of them exist between two particular services or two particular microservices at all, one pair at a time.

By its philosophy the contract testing progresses in five major steps:

1. Consumer specifies all he needs from the Provider.
2. Consumer team writes a test code that verifies if Client need was satisfied.
3. Pact framework generates a Contract(s) by looking at test(s) written by Client Team.
4. Pact Broker stores the contract. The Client considers his part of testing accomplished.
5. Pact Verifier scans contracts in Pact Broker's repository. Pact Verifier automatically builds and sends requests to the Provider and evaluates the responses.

The above approach, and tools that support it, achieve a fundamental advantage from project management point of view: existence of Pact Broker allows Provider team to test their server on multiple repetitions scattered over arbitrarily long time without ever bothering Clients (one or multiple) to re-sed their requests. The test system is by design, persistently regression test ready. And, what sometimes

counts a lot from a business perspective, allows for continuous improvements of the Provider Server with a minimized risk of impacting ongoing service provided to the clients.

Now let us return to challenges again.

Masking:

Contract in Pact Broker cannot contain formulas, and is should not as Contract Testing should not aspire to extend to Functional Testing. Contract, on the other hand allows to use REGEX expressions to specify expected values.

Ability to use regex can benefit the test process two ways:

1. Pass criteria can be more generic, that is focus on form as opposed to functionality.
2. PII or other confidential items do not need to be faked, masked, encrypted.

Pact Broker allows to insert new Contracts to its repository not only via an automated mechanism of deriving Contracts from Client Test Cases but as well directly by REST API.

Why would anybody want to do it? Why bypass automated processes with manual work? Are we not going backwards?

Well, look up on a section about challenges with Client Test steps in Contract Testing.

If a formal specification of Client Tests from Client's Team is unavailable the Test Team can still reach out to a person with privileges to access the production data.

Such an increased privileges employee can intercept the traffic (requests and responses) for a while (might be quite a long while) and save them with values converted to regex (as opposed to explicit strings or numbers). While looking at payloads with values converted to regex, with a simple compare tool, we will, I bet, find that 99% of intercepted traffic are duplicates of each other. Ask your high-privilege co-worker to preserve unique payloads only (matching pairs request and response together) and hand them to the Test Team.

If neither the legacy Provider not new Provider is unavailable the DevOps is still benefiting from recording regex-ized requests. It is just that the work of BA needs to be involved to work out required responses. What we achieve at this stage is better understanding of statistics and syntax of incoming requests.

However, even if we could benefit from having Provider responses recorded, the analytical work needs to be performed by humans here. The Contracts need to be manually written for each unique request-response pair and inserted into Pact Broker.

The author included a sample contract in lines 107 to 145 of the demo JSON file.

The remaining lines in JSON file are there to illustrate how converting values to REGEX expression that match these values might look in the resulting (marked or regex-ized) JSON file.

Code snippets:

Greg-to-ChatGPT

Please generate a python code that constructs a regex expression that would consume given string passed as a parameter to this code. For example, replace verbatim sequence of digits with generic match for a digit and verbatim strings of letters by sets of letters that would match the parameter passed.

ChatGPT:

```
import re

def construct_regex_pattern(input_string):
    # Replace verbatim sequences of digits with \d+ and verbatim strings of letters with [a-zA-Z]+ in the input string
    generic_pattern = re.sub(r'\d+', r'\d+', input_string)  # Replace digits with \d+
    generic_pattern = re.sub(r'[a-zA-Z]+', r'[a-zA-Z]+', generic_pattern)  # Replace letters with [a-zA-Z]+

    return generic_pattern

# Example usage:
input_str = "example123string"  # Replace with your input string
regex_pattern = construct_regex_pattern(input_str)

print("Input String:", input_str)
print("Constructed Regex Pattern:", regex_pattern)
```

If not done already go to python home and download PIP. and then use PIP to install Python.

Save the above code in a file with extension .py. Then open CMD prompt, cd to the path where your file is and type “python {yourfilename}.py”

You will find out that some tweaks are required because ChatGPT did not take into account that when substituting digits with a letter ‘d’ and subsequently substituting letters with another regex the code spoils its own work.

It might take a while to arrive to the set of regex transformations that meet your need. But you got a startup code and actually simple suggestion how to add sections to regex without thinking too much.

For my need I settled to a function below:

######################

```
import re
import json

def construct_generic_regex_pattern(input_string):
    # Escape special characters in the input string to create a valid regex pattern
    escaped_string = re.escape(input_string)
    #escaped_string = input_string

    # Replace verbatim digits with \d and verbatim letters with [a-zA-Z] in the escaped string
    generic_pattern = re.sub(r'[A-Za-z\s]+', r'[a-zA-Z\\s]+', escaped_string)
    # Replace digits with [0-9]+ or \d+
    generic_pattern = re.sub(r'[0-9]{9}', r'[0-9]{9}',generic_pattern )  # Replace digits with 9 digits
    generic_pattern = re.sub(r'[0-9]{4}', r'[0-9]{4}', generic_pattern)  # Replace digits with 4 digits
    generic_pattern = re.sub(r'[0-9]{3}', r'[0-9]{3}', generic_pattern)  # Replace digits with 4 digits
    generic_pattern = re.sub(r'[0-9]{2}', r'[0-9]{2}', generic_pattern)


    # Constructing the regex pattern
    regex_pattern = f"^{generic_pattern}$"  # Matching the whole string

    return regex_pattern
```

Then ask ChatGPT again:

Hey dude, generate python code that takes json file nested to an arbitrary depth and replaces values with regex expressions - that the script generates itself from the value being replaced = except if the value is in a specific set of strings. Such values should remain as is. And except for values of items with specific names contained in a specific set of strings, say set "A". Such items should be given always the same value (a string) specified for each excluded name distinctly. And except for values of items with specific names contained in a specific set of strings, say set "B". Such items should retain value as it is.

Try yourself.

Then combine both by hand.

I ended up with the following code:

```
import re
import json

def construct_generic_regex_pattern(input_string):
    # Escape special characters in the input string to create a valid regex pattern
    escaped_string = re.escape(input_string)
    #escaped_string = input_string

    # Replace verbatim digits with \d and verbatim letters with [a-zA-Z] in the escaped string
    generic_pattern = re.sub(r'[A-Za-z\s]+', r'[a-zA-Z\\s]+', escaped_string)
    # Replace digits with [0-9]+ or \d+
```

```
    generic_pattern = re.sub(r'[0-9]{9}', r'[0-9]{9}',generic_pattern )  # Replace digits with 9 digits
    generic_pattern = re.sub(r'[0-9]{4}', r'[0-9]{4}', generic_pattern)  # Replace digits with 4 digits
    generic_pattern = re.sub(r'[0-9]{3}', r'[0-9]{3}', generic_pattern)  # Replace digits with 3 digits
    generic_pattern = re.sub(r'[0-9]{2}', r'[0-9]{2}', generic_pattern)


    # Constructing the regex pattern
    regex_pattern = f"^{generic_pattern}$"  # Matching the whole string

    return regex_pattern


def replace_values_with_regex(obj, exclude_set_values, specific_items_A, specific_items_B):
    if isinstance(obj, dict):
        for key, value in obj.items():
            if key in specific_items_A:
                obj[key] = specific_items_A[key]  # Replace specific items' values from set A
            elif key in specific_items_B:
                continue  # Retain values for specific items from set B
            elif isinstance(value, str) and value not in exclude_set_values:
                # Generate regex pattern based on the value being replaced
                regex_pattern = re.escape(value)  # Escaping special characters
                #regex = re.compile(regex_pattern)
                #obj[key] = regex.pattern  # Using regex pattern instead of the original value
                obj[key] = construct_generic_regex_pattern(regex_pattern)
            else:
                obj[key] = replace_values_with_regex(value, exclude_set_values, specific_items_A, specific_items_B)
        return obj
    elif isinstance(obj, list):
        return [replace_values_with_regex(elem, exclude_set_values, specific_items_A, specific_items_B) for elem in obj]
    else:
        return obj

# Read JSON file
file_path = 'article_regex_demo.json'
with open(file_path, 'r') as file:
    json_data = json.load(file)

# Define sets for excluded values, specific items from set A, and specific items from set B
exclude_set_values = {'developing-with-docker', 'sha1-IpnwLG3tMNSllhsLn3RSShj2NPw=', '1.3.7',
'FrontEndService','BackEndService','GET','SET', 'application/json','application/json; charset=utf-8','/products'}  # Replace with
your excluded values
specific_items_A = {'id': 'uniqueGUID', 'integrity': 'integritySHA'}  # Replace with item names and replacements for set A
specific_items_B = {'version', 'resolved', 'media-typer', 'mime-types','statuses','productId', 'status' }  # Replace with item names
for set B

# Process JSON data
processed_json = replace_values_with_regex(json_data, exclude_set_values, specific_items_A, specific_items_B)

# Display the processed JSON data
print(json.dumps(processed_json, indent=2))
```

Next step would be to ask ChatGPT to give you code to do the same thing to every json file in given directory.

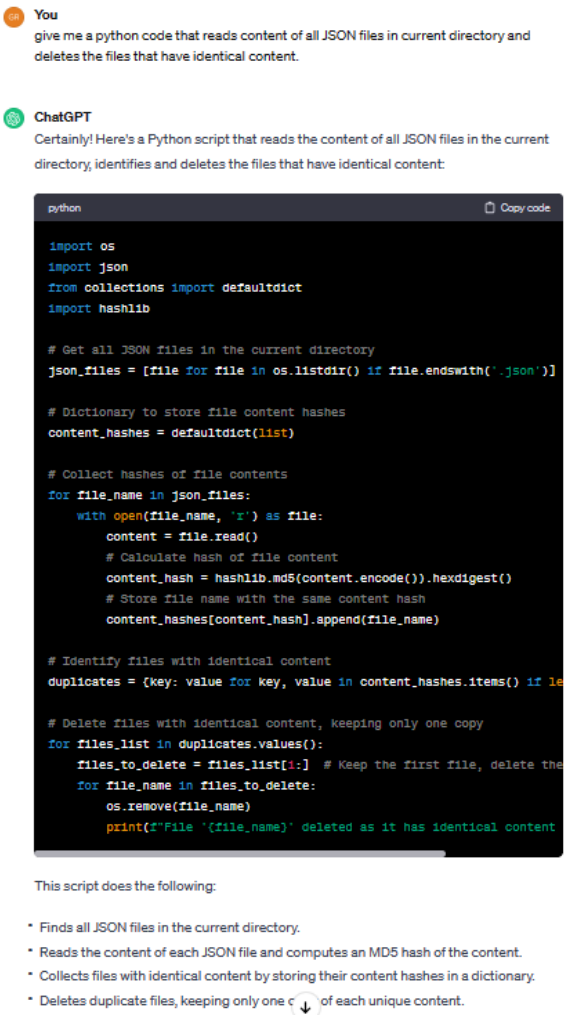
Than to ask ChatGPT to delete json files with douplicated content.

“ GR

You

give me a python code that reads content of all JSON files in current directory and deletes the files that have identical content.

“



You

give me a python code that reads content of all JSON files in current directory and deletes the files that have identical content.

ChatGPT

Certainly! Here's a Python script that reads the content of all JSON files in the current directory, identifies and deletes the files that have identical content:

python — Copy code

```
import os
import json
from collections import defaultdict
import hashlib

# Get all JSON files in the current directory
json_files = [file for file in os.listdir() if file.endswith('.json')]

# Dictionary to store file content hashes
content_hashes = defaultdict(list)

# Collect hashes of file contents
for file_name in json_files:
    with open(file_name, 'r') as file:
        content = file.read()
        # Calculate hash of file content
        content_hash = hashlib.md5(content.encode()).hexdigest()
        # Store file name with the same content hash
        content_hashes[content_hash].append(file_name)

# Identify files with identical content
duplicates = {key: value for key, value in content_hashes.items() if le

# Delete files with identical content, keeping only one copy
for files_list in duplicates.values():
    files_to_delete = files_list[1:]  # Keep the first file, delete the
    for file_name in files_to_delete:
        os.remove(file_name)
        print(f"File '{file_name}' deleted as it has identical content
```

This script does the following:

- Finds all JSON files in the current directory.
- Reads the content of each JSON file and computes an MD5 hash of the content.
- Collects files with identical content by storing their content hashes in a dictionary.
- Deletes duplicate files, keeping only one c of each unique content.

Then give both or all three (depending how tou decided is the best to run them) above python files to your co-worker with privileges that allow him to access production data.

Once you get back unique pairs of files you need to manually produce Contracts.

Notes:

The Pact framework has official support for several languages including:

1. **Ruby**: The language in which the Pact Broker is built.
2. **Java**: With the Pact JVM library.
3. **JavaScript (Node.js)**: Using the Pact JS library.
4. **.NET/C#**: Supported by the Pact .NET library.
5. **Python**: Community-supported libraries exist for Python, but might not have as much official support as some other languages.

The sets of projects that use Pact, PactFlow, Can-I-Deploy, Pact Broker can be found at vendor's github:

__ https://github.com/pact-foundation/pact-python

__https://github.com/pact-foundation/pact-jvm

__https://docs.pact.io/getting_started/sharing_pacts

__ PactFlow (github.com) PactFlow is primarily focused on supporting the Pact ecosystem, which is more native to JVM-based languages like Java, as well as JavaScript, Ruby, and .NET.

__ https://docs.pact.io/pact_broker

Pact CLI
Pact JS
Pact JVM
Pact Net
Pact Go
Pact Python
Pact Swift
Pact Scala
Pact PHP
Pact Ruby
Pact Rust
Pact C++
Pact on Docker
Pact Plugins
https://slack.com/apps/A016SU90DSN-pact?tab=more_info

Technology choices your enterprise prefers are broad, and the aspects of these decisions are not considered in this article.

(46) Beyond REST - Contract Testing in the Age of gRPC, Kafka & GraphQL • Matt Fellows • YOW! 2022 - YouTube