MICRO FOCUS

Service Virtualization version 2022

Hands-on Exercises

The Story behind this Hands-on Series

In this set of hands-on exercises you will face a lot of troubles with the **Portfolio Suggestion Service**, a dependency you heavily rely on.

"The team behind the Portfolio Suggestion Service was initially unable to deliver anything. When they finally did, after months, the service was buggy, provided inconsistent output, performed badly, contained sensitive data, and some important features were undocumented, or missing with no clear outlook to fix. They were also unable to generate test data of a realistic size, and at the end they even started to charge your team for over-using their test environment."

Luckily in the end your own app was still a success.

Fortunately you used

Micro Focus Service Virtualization

Purpose of Portfolio Suggestion Service

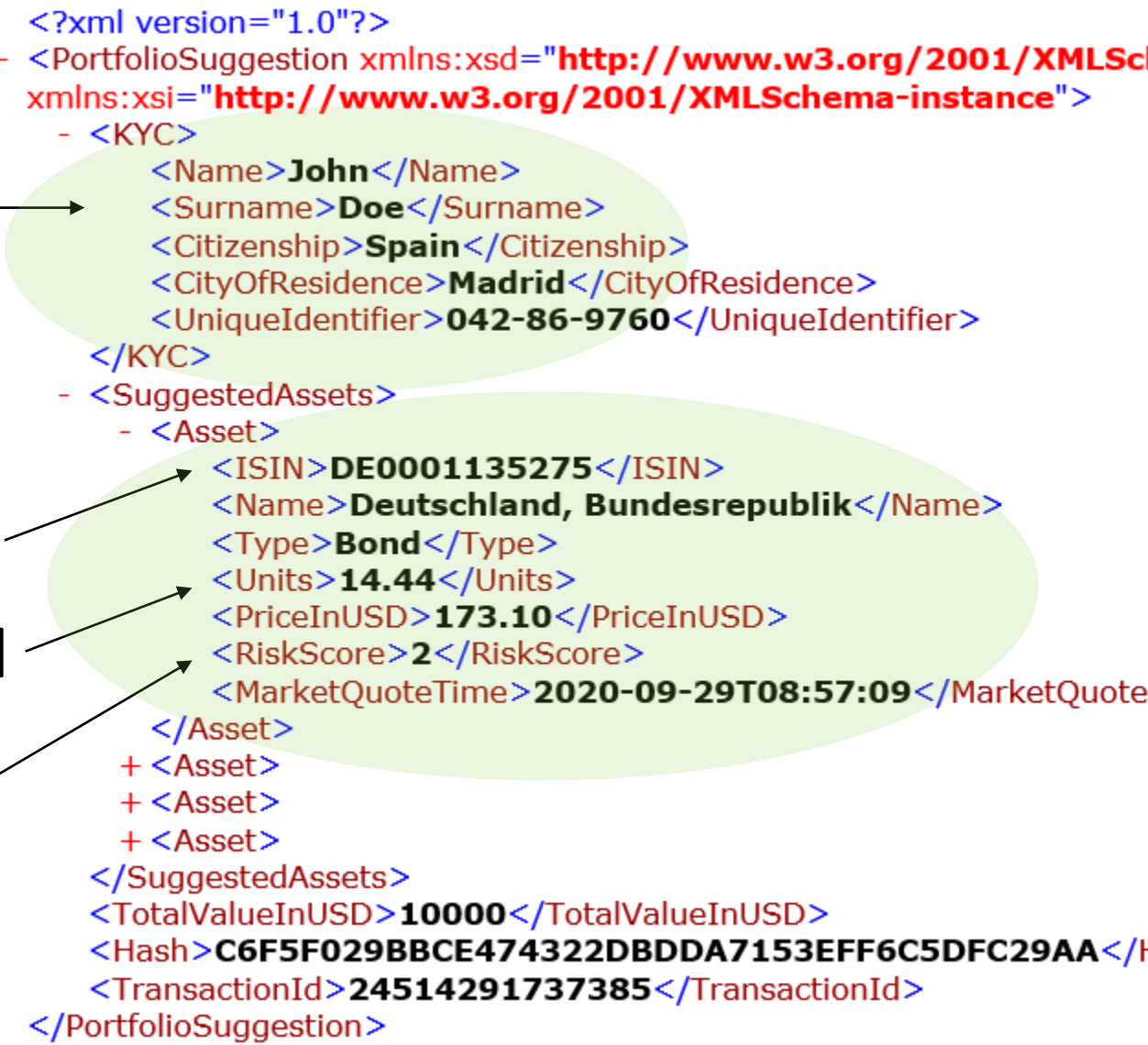
- **You develop an application responsible for maintaining a portfolio of financial assets.**
- Using AI the Portfolio Suggestion Service suggests portfolio assets depending on whether the **Purpose** is ***Hedging*** or ***Speculation***, based on a permissible **Risk Level** (0-100) and the **Intended Investment Amount**.

Output of Portfolio Suggestion Service

The Portfolio Suggestion Service runs basic **KYC** checks (KYC="know your customer").

And mainly it retrieves an array of assets identified by the **ISIN code**.

It also suggests the **recommended amount** to buy and the current **market price** and **risk score**.



Hands-on 0: Create a Virtual Service from scratch

Issue

The team responsible for developing the Portfolio Suggestion Service was unable to deliver anything yet. But you need to integrate the dependency and start testing your system.

Mitigation

Fortunately there is a documentation with some basic sample messages. Use them to create a virtual service that will simulate basic calls to the non-existent real service.

Hands-on 0: Instructions to follow 1/3

1. Install the free SV Designer as described here (or use other SV Designer 2022).
2. Create a project called e.g. "Portfolio Suggestion Services“ on the Embedded Server.
3. Add an empty virtual service based on no service description and the REST protocol. Call it “Portfolio Suggestion Service”. Select the “HTTPS Gateway” Agent. Don't specify the real service endpoint, since we don't have one yet. The virtual service path on the agent should be "**suggestportfolio**".
4. Make sure you can successfully call the service from your REST client using the POST method to get the 200 OK code
   (you'll likely need to disable SSL certificate validation first).
   The response message body will be empty, since there is nothing inside your Data Model yet.

Hands-on 0: Instructions to follow 2/3

5. Create a new Rule called "Samples from Documentation". Download the Training.zip file linked in the video description. There, inside the Sample Data subfolder take request*.xml and response*.xml files and import them to the newly created rule.
   The sample messages are meant to be used with the POST method, and invoked against the root URL https://localhost:7205/suggestportfolio - so no Uri path needs to be specified.
   Also, **uncheck the Create a Track checkbox when importing.**

6. Call the virtual service using one of the sample requests, make sure you receive the corresponding sample response. Make sure you use the correct Accept and Content-Type headers while using your REST client.

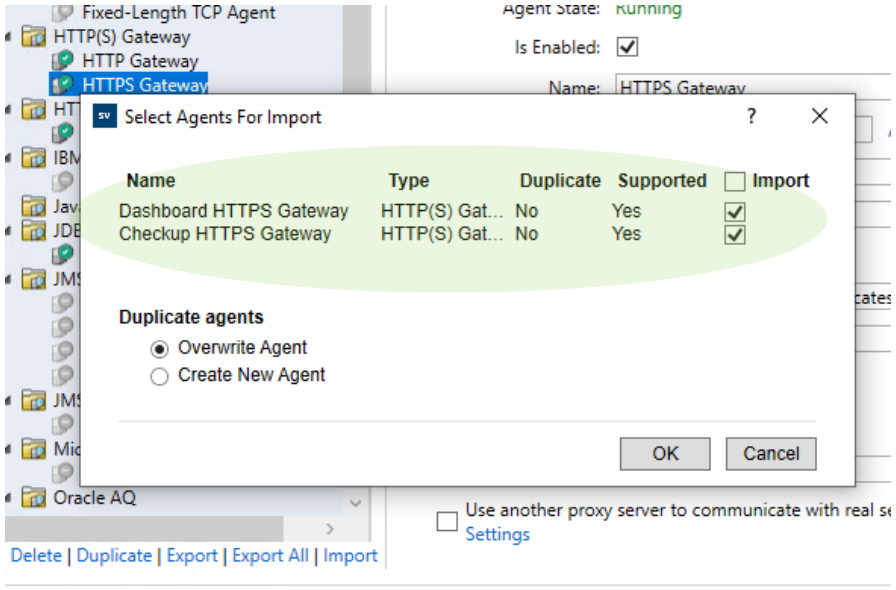
7. Change some value in your sample request and see the empty response generated from the Default Rule.

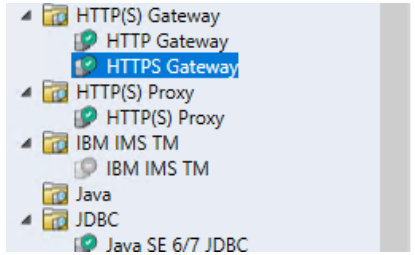
8. To make the Default Rule response easy to recognize, reorder the rows in the Default Rule, so that sim_default_values are returned instead of the empty response.
   You also need to set the correct Content Type for the corresponding row in the Default Rule.

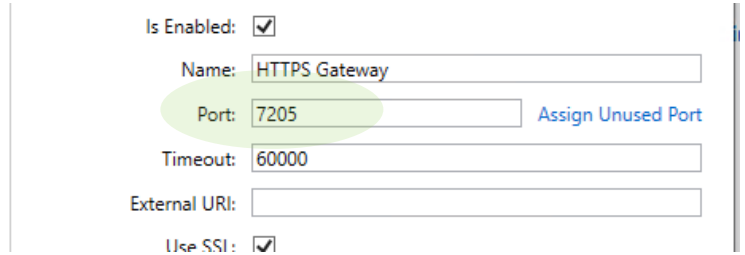
Hands-on 0: Checking your solution 3/3

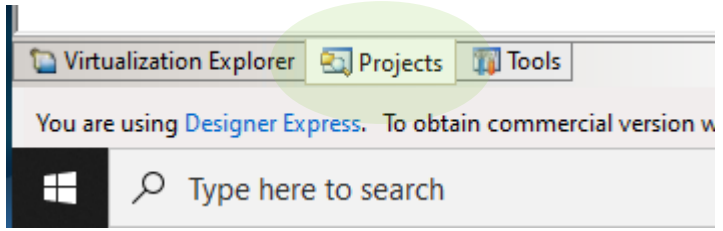
9. To automatically check your progress on the hands-on exercises, you need to import 2 special agents and a project archive file with the dashboard and checkup services.

10. Go to the agents dialog and import the “Training Agents.agce” file (from Training.zip).
Also make sure the HTTPS Gateway agent listens on port 7205.









11. Switch to the Projects tab and right-click the Solution root to add an existing project.
Select the “Training Services.vproja” file and import it to your solution.
Pass the first checkpoint test in the dashboard at https://localhost:7203

Hands-on 1: Condition Functions

Issue

My virtual service works just for the exact same requests
I imported in the past. I want the virtual service to return some meaningful responses for any meaningful request.

Mitigation

Make the virtual service more versatile by introducing
a more flexible request matching mechanism.

Hands-on 1: Instructions to follow

In the "Sample from Documentation" Rule configure Condition Functions so that:

1. You receive some sample response values in all the response fields
   even when you invoke the service using a request having arbitrary values inside the **IntendedInvestmentValue** structure, and the **Customer** structure.
2. Make sure you receive a **Speculation** or **Hedging** response when asked
   for one in the request's **Purpose** field.
3. By creating a Custom Condition Function make sure for any given **RiskLevel**
   you always match a request with the closest lower **RiskLevel** (if there is one).
4. Pass the second checkpoint test in the dashboard at https://localhost:7203

Hands-on 2: Dynamic Functions

Issue

I want the virtual service to produce on output
the same Customer values it received on input.

I also want to receive an increasing sequence of numbers in the TransactionId field.

Mitigation

Use Dynamic Functions to copy data from request to response.

Use Value Generators to produce an increasing sequence.

Hands-on 2: Instructions to follow

1. Create a new Rule called "Copy Customer data to response" which will copy the Customer element data from request to response.
2. Create a new Rule called "Generate Transaction ID sequentially". Find out what's the current maximum Transaction ID in the sample data and make the simulation generate an increasing sequence of Transaction IDs higher than this historical maximum
   (you can generate e.g. just the last 4 digits, the prefix can remain the same, since we don't expect running tests with > 10000 iterations).
3. Pass the third checkpoint test in the dashboard at https://localhost:7203

Hands-on 3: Arrays

Issue

The real service is unable to provide a realistic amount of response data. I need to get ready for displaying 50 asset suggestions, not just 4-8 we saw in the sample data (and in their test environment).

Mitigation

Append some randomly generated assets, following the sample data assets to produce an output of a realistic size.

Hands-on 3: Instructions to follow

1. Make the Simulation generate 50 assets. All the assets in the response should have the MarketQuoteTime set to about 10 minutes in the past. The time zone should be UTC.
2. The sample data imported from the documentation should otherwise remain intact, but the newly generated items should be generated with ISIN = 'US000000####' (# is a random digit), Name = 'Asset ##', Type = 'Equity', Units = '##.##', PriceInUSD = '###.##'. For RiskScore do copy the value from the request.RiskLevel element.
3. You will likely need to create 3 new Rules called e.g.:
   "Set MarketQuoteTime to -10 minutes", "Generate random asset values", "Set asset count to 50".
4. Pass the 4th checkpoint test in the dashboard at https://localhost:7203

Hands-on 4: Message Types

Issue

Finally the first version of the Portfolio Suggestion Service was made available.
We've just learned they forgot to tell us about a special Message Type called Unsupported. They currently use it whenever they don't have a feature implemented yet, but in the future they want to use it to signalize a legal denial of services (e.g. due to Sanctions imposed on a country).
We need to get ready to display this use case properly ASAP!
For them it's just a detail with the lowest possible priority and no ETA.

Mitigation

Learn the new Message Type from the real service and use it to signalize sanctions for Iran (which they insist is the only country on the sanctions list).

Hands-on 4: Instructions to follow

1. Connect your virtual service to the real service endpoint https://localhost:7203/suggestportfolio , switch it to Stand By, and invoke the virtual service with the 'request1.xml' sample request. See the response, then change the request Currency to CZK and send the request again. You should get a different type of message on response.
2. Switch the service to Recording and learn the new response type (called "Unsupported"). Check you can see a new row in the Learned Data Rule, and keep it there for later.
3. Create a new Rule called "Sanctions on Iran". Setup the Rule to make the service return the Unsupported response type with a message "Country is on a sanctions list.“ whenever the **request.Customer.Citizenship** field is equal to **Iran**.
4. Pass the 5th checkpoint test in the dashboard at https://localhost:7203

Hands-on 5: Simulation Troubleshooting

Issue

The virtual service is getting more complex than expected and I’m not sure why it returns a certain response for some request. The behavior seems to be incorrect.

Mitigation

Use the Simulation Report and the Message Log
to analyze the behavior.
Then fix it.

Hands-on 5: Instructions to follow

1. Switch the service to Simulation.
2. Try sending the 'request1.xml' sample request with the Currency changed to CZK. Do you receive the learned "Unsupported" response?
3. Using troubleshooting tools investigate why not, and introduce minimal changes to the Data Model to make it work.
   (The simplest solution can be done usually with less than 10 successive mouse clicks.)
4. Pass the 6th checkpoint test in the dashboard at https://localhost:7203

Hands-on 6: Data Masking

Issue

When using the real service we discovered they use non-sanitized sensitive data in their test environment. We need to sanitize the data when they enter our system during the learning process, not to be held legally accountable.

Mitigation

Use Data Masking to mask out the sensitive data.

Hands-on 6: Instructions to follow

1. **Backup your project** by exporting it to a *.vproja archive and undeploy your virtual service.
   For the Data Masking and Stateful Simulation hands-ons we'll create a new project called “Data Masking and Stateful Simulation".

2. Create the new SV project with a virtual REST service virtualizing the real service endpoint.
   Try invoking the service in Stand By Mode, using couple of requests from the sample XML files.

3. Before setting up Data Masking learn some sample request via recording the real service, so that the REST service could learn the respective data structures. Setup Data Masking on the **UniqueIdentifier** field in response. In case it contains the Social Security Number (regular expression \d{3}\-\d{2}\-\d{4}) do replace it with the value of "SSNRemoved".

4. Setting up Data Masking will wipe out all of your Data Model rows. Record new data by invoking the 'request1.xml' sample **request exactly 5 times**. Check that the Data Model contains the value of SSNRemoved inside the **UniqueIdentifier** field. Invoke the service in Simulation using the same sample request and check the **UniqueIdentifier** value gets properly masked.

5. Pass the 7th checkpoint test in the dashboard at https://localhost:7203

Hands-on 7: Stateful Simulation

Issue

While working with the real Portfolio Suggestion Service
we discovered it returns a wildly different set of suggested assets for the same request, when invoked multiple times.
This is inconsistent. It’s due to the incorrectly chosen AI algorithm on the real service side and they don’t know how to quickly fix it.
It will likely cost a lot of time and money.

Mitigation

Use Stateful Simulation to return only a reasonably different response for a repeatedly invoked request.
(Keep the same set of assets and change just the market price, but only a little bit).

Hands-on 7: Instructions to follow

1. In this exercise we'll continue working on the project, where we previously introduced Data Masking.
2. Switch the service to Simulation and periodically invoke the service using the 'request1.xml' sample.
3. See how the responses cycles over the 5 recorded rows within an SV Track of length 5.
4. Now change the Data Model so that the response always contains the same set of assets, but each time with a slightly different PriceInUSD.
5. Pass the 8th checkpoint test in the dashboard at https://localhost:7203

Hands-on 8: Service Call Activities

Issue

The Portfolio Suggestion Service team was wrong when insisting that Iran is the only country on the sanctions list.
There is actually a microservice returning info on countries facing sanctions and we need to consume its output before they can fix it on the real service side.

Mitigation

Use a Service Call Activity to consume the Sanctions microservice.

Hands-on 8: Instructions to follow 1/2

1. Undeploy the virtual service you used for the Data Masking and Stateful Simulation exercises and deploy the original service again.
2. Import a new REST Service Call Activity to your Data Model. We don't have a service description for it. The real endpoint of the SCA is https://localhost:7203/sanctions
3. Select "Do not add an activity now", as we need to learn data structures for the SCA first. Select "Learn Schema from Request/Response Message" and use the following sample request and response to learn the structures. The SCA uses the POST HTTP method:

Request

```
<CountryToCheck
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Name>Czechia</Name>
</CountryToCheck>
```

Response

```
<Sanctions
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <InEffect>false</InEffect>
</Sanctions>
```

Hands-on 8: Instructions to follow 2/2

1. Create a new Rule called "Call external Sanctions service“, which will call the Sanctions service for the country in the **request.Customer.Citizenship** field. Insert the SCA as “Before Response” activity.
2. You also need to set the **HttpInputParameters**. Namely you need to set the **UriPath** to the empty string, the **Method** to POST, the **Transport** to HTTPS, and in the Activity Service Description Editor you need to enable the **Content-Type** and **Accept** headers and set both to "text/xml" in the SCA's request.
3. Create another new Rule called "Retrieve Unsupported if country faces sanctions", and set the response type to Unsupported with the Message field set to "Country is on a sanctions list.".
4. Test the new feature for Iran and Russia (both should face sanctions).
5. Pass the 9th checkpoint test in the dashboard at https://localhost:7203

Hands-on 9: JavaScript Rule

Issue

All randomly generated assets have the asset Type set to ‘Equity’. This is not realistic, but there is no built-in function to implement it in a better way.

Mitigation

Use the JavaScript rule to populate also
the asset Type field randomly.

Hands-on 9: Instructions to follow

1. Create a JavaScript Rule called "Set Type of randomly generated assets" which will randomly populate the asset Type field for the randomly generated assets.

2. Below there is a JavaScript code, where you need to fill in a command to retrieve the data from the response part of the Processing Row.

3. Pass the 10th checkpoint test in the dashboard at https://localhost:7203

```
if(hpsv.response.PortfolioSuggestion) {
var assets = //TODO Here write a command to retrieve the assets array.
    for (i = 0; i < assets.length; i++) {
        var assetType = Math.floor(Math.random() * 4)%4;
        if(assetType == 0) {
            assets[i].Type = "Equity";
         }
        if(assetType == 1) {
            assets[i].Type = "Bonds";
         }
        if(assetType == 2) {
            assets[i].Type = "ETF";
         }
        if(assetType == 3) {
            assets[i].Type = "Commodity";
         }
    }
}
```

Hands-on 10: C# Rule

Issue

There is a rounding bug in the Portfolio Suggestion Service.

The TotalValueInUSD is being calculated incorrectly,

since they use a float number instead of a decimal number.

Mitigation

C# rule is fast and supports decimal numbers.

Use it to fix the real service behavior.

Hands-on 10: Instructions to follow

1. Create a new C# Rule and calculate the accurate **TotalValueInUSD** value using the code below. You need to fill in commands to retrieve the value of the suggested assets array, as well as to store the calculated result in the **TotalValueInUSD** field.

2. Pass the 11th checkpoint test in the dashboard at https://localhost:7203

```
if(!sv.Response.IsNotPresent("PortfolioSuggestion")) {
    decimal total = 0;
    var assets = //TODO Here write a command to retrieve the value assets array.
    foreach(var asset in assets)
    {
        var price = decimal.Parse(asset.PriceInUSD);
        var units = decimal.Parse(asset.Units);
        total += price * units;
    }
    //TODO here write the destination to store the calculated value
    = total.ToString();
}
```

Hands-on 11: Hybrid Simulation

Issue

The Portfolio Suggestion Service team decided to outsource some of their service behavior to an expensive external system. When using their test environment significant expenses get incurred on their side and they want us to share the burden.

Mitigation

Only call the real service for a tiny portion of traffic
to drive down the cost.

Hands-on 11: Instructions to follow

1. Use Hybrid Simulation and create a new Rule called "Contact the real service for customer Karel only" which will make the virtual service call the real service only when the **Customer.Name** is equal to "Karel".
2. Deploy the changes to the shared SV Server and pass the 12th checkpoint test.

Hands-on 12: Final Check

Now it’s the time to run the final check at

https://localhost:7203

to make sure all the features you have introduced work well together.

Fix possible issues.

Good luck!


MICRO®
FOCUS