cisco Live!
Amsterdam | February 5-9, 2024

DEVNET-1116

Cisco Full-Stack Observability API: An Introduction for Beginners

Speaker name – Anuj Modi

Learning Objectives

Upon completion of this lab, you will be able to:

- Clone the "ciscolivedemo2024" repository.
- Gain proficiency in making API calls with AppDynamics to create health rules for the Supercar-Trader application.
- Acquire knowledge on utilizing ThousandEyes API calls to generate a web test for the Supercar-Trader application, incorporating enterprise agents.

Disclaimer

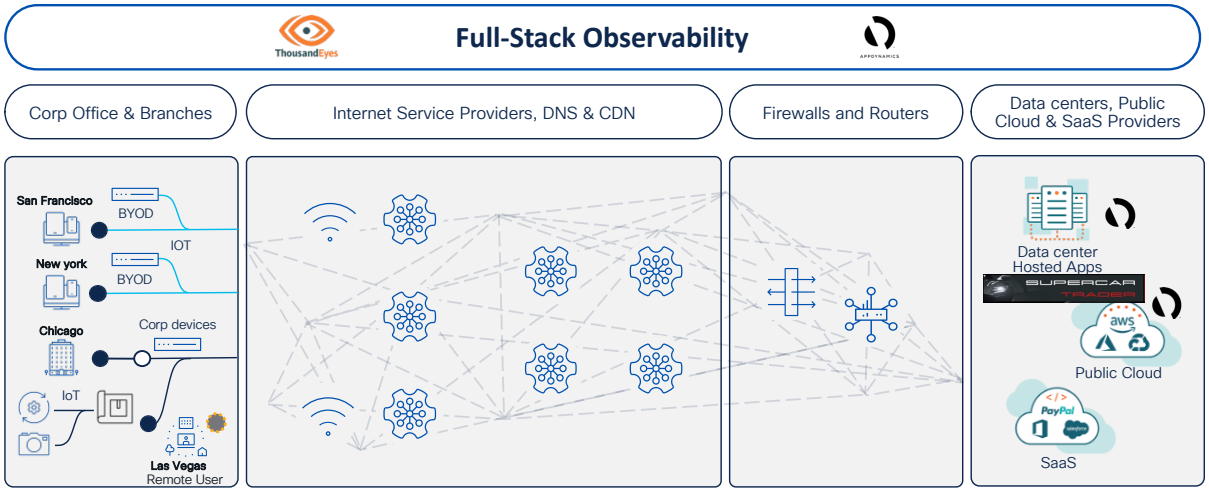
This training document is to familiarize with Full-Stack Observability APIs. Although the lab design and configuration examples could be used as a reference, it's not a real design, thus not all recommended features are used, or enabled optimally. For the design related questions please contact your representative at Cisco, or a Cisco partner.

Scenario

The Supercar-Trader company is a fictional entity currently undergoing global expansion. The company offers an e-commerce platform to its customers and partners, facilitating online car sales. The Supercar-Trader application is widely utilized across the globe. However, the company has been receiving complaints from customers and partners who are unable to place orders online, significantly impacting the company's revenue.

In response to this issue, the company recently implemented the Full-Stack Observability (FSO) solution in the American region, receiving positive feedback. Encouraged by the success in America, the company now aims to extend the FSO solution to Europe and the Asia-Pacific regions. The leadership team has tasked the DevOps team with deploying the solution in 200 branches. To achieve this, the DevOps team plans to utilize the FSO APIs to automate the solution-building process.

Network Diagram



Trader Application:http://app.cloudkareai.com:8080/Supercar-Trader/home.do

AppDynamics cSaaS URL: https://kickstarter.saas.appdynamics.com

Access Token: will be fetched directly from the vault token.

API Reference:

https://docs.appdynamics.com/appd/23.x/latest/en/extend-appdynamics/appdynamics-apis

ThousandEyes URL: https://app.thousandeyes.com/login

Access Token: will be fetched directly from the vault token. Password will be shared during the session.

API Reference:

https://developer.thousandeyes.com/v6/

Task 1: Clone the repository on your laptop

Step 1: Create a local directory and clone the repository.

Create a directory on your local machine and navigate to the newly created <dir name> directory. Then, clone the repository into this directory:

https://github.com/anujmodi1/ciscolivedemo2024.git

```
# Create the directory on the local machine
mkdir -p fsolab<yourname>
cd fsolab<yourname>

# Clone the repository
git clone https://github.com/anujmodi1/ciscolivedemo2024.git
cd ciscolivedemo2024
```

Verify the git status of the folder:

```
# Check the git status
git status
```

Open the ciscolivedemo2024 folder using Visual Studio or terminal.

Navigate to the terminal in Visual Studio Code, open the cloned folder. Verify that you are within the ciscolivedemo2024 folder.

Task 2: Interacting with AppDynamics APIs

Note: This step will have already been completed by the lab administrator.

Step 1: Generate the access token through Client Secret

File content:

```
curl -X POST -H "Content-Type: application/vnd.appd.cntrl+protobuf;v=1"
"https://kickstarter.saas.appdynamics.com/controller/api/oauth/access_token" -d
"grant_type=client_credentials&client_id=fsolab@kickstarter&client_secret=xxxxx"
```

Response:

{"access_token":
"eyJraWQiOiI5ZjUzZTZlNC01ZWMxLTQ5NDctOWI4ZC1mODNlZWM1MjNhNjgiLCJhbGciOiJIUzI1NiJ
9.eyJpc3MiOiJBcHBEeW5hbWljcyIsImF1ZCI6IkFwcERfQVBJcyIsImp0aSI6Im1QeUVfdE1BMDViZ3RM
UGZSYWNEbFEiLCJzdWIiOiJmc29sYWIiLCJpZFR5cGUiOiJBUElfQ0xJRU5UIiwiaWQiOiJkNGNmNGY4
Ny1kZGI1LTQzYzAtYTA1Zi05MzNhZWI2YmE1YzIiLCJhY2N0SWQiOiI5ZjUzZTZlNC01ZWMxLTQ5NDct
OWI4ZC1mODNlZWM1MjNhNjgiLCJ0bnRJZCI6IjlmNTNlNmU0LTVlYzEtNDk0Ny05YjhkLWY4M2VlYz
UyM2E2OCIsImFjY3ROYW1lIjoia2lja3N0YXJ0ZXIiLCJ0ZW5hbnROYW1lIjoia2lja3N0YXJ0ZXIiLCJmbW
1UbnRJZCI6bnVsbCwiYWNjdFBlcm0iOltdLCJyb2xlSWRzIjpbXSwiaWF0IjoxNzA1MTUwMTExLCJuYm
YiOjE3MDUxNDk5OTEsImV4cCI6MTcwNTE1MDQxMSwidG9rZW5UeXBlIjoiQUNDRVNTIn0.Y8bZruz
_r9n8HyPTTdeB7EADhx8pNXaCcDTg0U7Qug8","expires_in": 300}%

Step 2: Obtain the List of Business Applications

Navigate to the Section01-AppD_Test folder and execute the applications.py file to generate a list of all business applications instrumented in the AppDynamics Controller

Please note that the placeholders password=<paasword> should be replaced with the actual password shared in the session..

Execute the following command to retrieve the list

python3 applications.py

File content:

```
import requests, json

url="https://kickstarter.saas.appdynamics.com/controller/rest/applications?output=JSON"

payload={}
headers = {
   'Content-Type': 'application/x-www-form-urlencoded',
   'Authorization': 'Bearer ' + '<token>'
}
response = requests.request("GET", url, headers=headers, data=payload)

print(response.text)
print(response.json)
```

Response:

```
<applications><application>
  <id>10090</id>
  <name>UCCX</name>
```

```
  <accountGuid>9f53e6e4-5ec1-4947-9b8d-f83eec523a68</accountGuid>
</application>
<application>
  <id>10095</id>
  <name>Supercar-Trader</name>
  <accountGuid>9f53e6e4-5ec1-4947-9b8d-f83eec523a68</accountGuid>
</application>
<application>
  <id>10089</id>
  <name>UCCE125</name>
  <accountGuid>9f53e6e4-5ec1-4947-9b8d-f83eec523a68</accountGuid>
</application>
</applications>%
```

Step 3: Retrieve All Business Transactions in the Supercar-Trader Application (Optional)

Execute the transactions.py file to enlist all business transactions in the Supercar-Trader Application

#python3 transactions.py

File content:

```
import json, re, sys, os, json, subprocess, time, logging, requests, urllib3
from subprocess import call, check_output
from requests.structures import CaseInsensitiveDict
urllib3.disable_warnings()

url="https://kickstarter.saas.appdynamics.com/controller/rest/applications/Supercar-
Trader/business-transactions"
payload={}
headers = {
   'Content-Type': 'application/x-www-form-urlencoded',
   'Authorization': 'Bearer ' + '<token>'
}

response = requests.request("GET", url, headers=headers, data=payload)

print(response.text)
print(response.json)
```

Response:

```
<business-transactions><business-transaction>
  <id>1503311</id>
  <name>ActionHome.execute</name>
```

```
  <entryPointType>STRUTS_ACTION</entryPointType>
  <entryPointTypeString>STRUTS_ACTION</entryPointTypeString>
  <internalName>ActionHome.execute</internalName>
  <tierId>132938</tierId>
  <tierName>Web-Portal</tierName>
  <background>false</background>
</business-transaction>
<business-transaction>
  <id>1503314</id>
  <name>ActionSell.execute</name>
  <entryPointType>STRUTS_ACTION</entryPointType>
  <entryPointTypeString>STRUTS_ACTION</entryPointTypeString>
  <internalName>ActionSell.execute</internalName>
  <tierId>132938</tierId>
  <tierName>Web-Portal</tierName>
  <background>false</background>
</business-transaction>

-----------
```

Step 4: Retrieve the health rules in the Supercar-Trader Application

Execute the heathrules.py file to list all available health rules in the Supercar-Trader Application.

#python3 heathrules.py

File content:

```
import json, re, sys, os, json, subprocess, time, logging, requests, urllib3
from subprocess import call, check_output
from requests.structures import CaseInsensitiveDict
urllib3.disable_warnings()

url="https://kickstarter.saas.appdynamics.com/controller/alerting/rest/v1/applications/10095/health-rules?output=JSON"
payload={}
headers = {
   'Content-Type': 'application/x-www-form-urlencoded',
   'Authorization': 'Bearer ' + '<token>'
}

response = requests.request("GET", url, headers=headers, data=payload)

print(response.text)
print(response.json)
```

Response:

[{"id":51517,"name":"Business Transaction Health","enabled":true,"affectedEntityType":"BUSINESS_TRANSACTION_PERFORMANCE"},{"id":51518,"name":"CPU utilization is too high","enabled":true,"affectedEntityType":"TIER_NODE_HARDWARE"},{"id":51519,"name":"Memory utilization is too high","enabled":true,"affectedEntityType":"TIER_NODE_HARDWARE"},{"id":51520,"name":"JVM Heap utilization is too high","enabled":true,"affectedEntityType":"TIER_NODE_HARDWARE"},{"id":51521,"name":"JVM Garbage Collection Time is too high","enabled":true,"affectedEntityType":"TIER_NODE_HARDWARE"},{"id":51522,"name":"CLR Garbage Collection Time is too high","enabled":true,"affectedEntityType":"TIER_NODE_HARDWARE"},{"id":51523,"name":"Network-Host : Packet drops too high","enabled":false,"affectedEntityType":"ADVANCED_NETWORK"},{"id":51524,"name":"Business Transaction Health?AlwaysRed1","enabled":true,"affectedEntityType":"BUSINESS_TRANSACTION_PERFORMANCE"}]

Step 4: Create a new the health rules in the Supercar-Trader Application

Execute the createhealthrule.py file to generate a new health rule in the Supercar-Trader Application.

#python3 createhealthrule.py

File content:

```
import json
import requests
import urllib3

urllib3.disable_warnings()

url =
"https://kickstarter.saas.appdynamics.com/controller/alerting/rest/v1/applications/10095/health-rules/"
health_rule_template = {
   "name": "Health Rule <yourname>",
   "enabled": True,
   "useDataFromLastNMinutes": 30,
   "waitTimeAfterViolation": 5,
   "affects": {
      "affectedEntityType": "BUSINESS_TRANSACTION_PERFORMANCE",
      "affectedBusinessTransactions": {
         "businessTransactionScope": "ALL_BUSINESS_TRANSACTIONS"
      }
   },
   "evalCriterias": {
```

```
    "criticalCriteria": {
      "conditionAggregationType": "ALL",
      "conditions": [
        {
          "name": "Condition 1",
          "shortName": "A",
          "evaluateToTrueOnNoData": False,
          "evalDetail": {
            "evalDetailType": "SINGLE_METRIC",
            "metricAggregateFunction": "VALUE",
            "metricPath": "Average CPU Used (ms)",
            "metricEvalDetail": {
              "metricEvalDetailType": "BASELINE_TYPE",
              "baselineCondition": "WITHIN_BASELINE",
              "baselineName": "All Data - Last 15 Days",
              "baselineUnit": "PERCENTAGE",
              "compareValue": 30.5
            }
          }
        }
      ]
    }
  }
}

headers = {
  'Content-Type': 'application/json',
  'Authorization': 'Bearer ' + '<token>'  # Replace with your actual token
}

response = requests.post(url, headers=headers, json=health_rule_template, verify=False)

print("Health rule created successfully.")
```

Response:

```
Health rule updated successfully.
```

Step 5: Update the health rules in the Supercar-Trader Application (Optional)

Execute the updatehealthrule.py file to modify an existing health rule in the Supercar-Trader Application.

#python3 updatehealthrule.py

File content:

```
import json
import requests
import urllib3

urllib3.disable_warnings()

url =
"https://kickstarter.saas.appdynamics.com/controller/alerting/rest/v1/applications/10095/health
-rules/"
health_rule_id = "51524"  # Replace with the actual health rule ID

# Fetch existing health rule
headers = {
   'Content-Type': 'application/json',
   'Authorization': 'Bearer ' + '<token>'  # Replace with your actual token
}

response = requests.get(url + health_rule_id, headers=headers, verify=False)

if response.status_code == 200:
   # Parse existing health rule configuration
   existing_health_rule = response.json()

   # Update the health rule configuration
   existing_health_rule['enabled'] = True  # Set 'enabled' to True to enable the health rule

   # Make a PUT request to update the health rule
   update_response = requests.put(url + health_rule_id, headers=headers,
json=existing_health_rule, verify=False)

   if update_response.status_code == 200:
      print("Health rule updated successfully.")
   else:
      print(f"Failed to update health rule. Status code: {update_response.status_code}")
      print(update_response.text)
else:
   print(f"Failed to fetch existing health rule. Status code: {response.status_code}")
   print(response.text)
```

Response:

```
Health rule updated successfully.
```

Congratulations! You have completed the AppDynamics Lab. You can now proceed to create the ThousandEyes Web Test for the Supercar Trader application with your own test name.

Task 3: Working with ThousandEyes APIs

Step 1: Retrieve the List of Enterprise Agents Deployed in Your Branches

Navigate to the Section02-TE_Test folder and execute the agent.py file to obtain a list of all enterprise agents deployed in your branches.

Please note that the placeholders <token> should be replaced with the actual token.

#python3 agents.py

File content:

```
import requests
import json
url = "https://api.thousandeyes.com/v6/agents.json"
payload={}
headers = {'Authorization': 'Bearer ' + "<token>"}
agent_response = requests.request("GET", url, headers=headers, data=payload)
print(agent_response)

agent_list_json = agent_response.json()
#print(agent_list_json)

agent_list = agent_list_json['agents']
list_of_dictionaries = agent_list
sought_value = "Enterprise"
found_values = []
for dictionary in list_of_dictionaries:
   if (dictionary["agentType"] == "Enterprise"):
      found_values.append(dictionary)
#print(found_values)

empty_list=[]
for item in found_values:
   agentId=item['agentId']
   print(agentId)
   empty_list.append({'agentId': agentId})
print(empty_list)
```

Response:

```
<applications><application>
```

Step 2: Create a New Web Test for the Supercar-Trader Application Using Your Enterprise Agents

Navigate to the Section02-TE_Test folder and execute the test.py file to create a new web test for the Supercar-Trader application using your enterprise agents.

#python3 test.py

File content:

```
import json, re, sys, os, json, subprocess, time, logging, requests, urllib3
from subprocess import call, check_output
from requests.structures import CaseInsensitiveDict
urllib3.disable_warnings()
url = "https://api.thousandeyes.com/v6/agents.json"
payload={}
headers = {'Authorization': 'Bearer ' + "<token>"}
agent_response = requests.request("GET", url, headers=headers, data=payload)
print(agent_response)

agent_list_json = agent_response.json()
#print(agent_list_json)

agent_list = agent_list_json['agents']
list_of_dictionaries = agent_list
sought_value = "Enterprise"
found_values = []
for dictionary in list_of_dictionaries:
   if (dictionary["agentType"] == "Enterprise"):
      found_values.append(dictionary)
#print(found_values)

empty_list=[]
for item in found_values:
   agentId=item['agentId']
   print(agentId)
   empty_list.append({'agentId': agentId})
print(empty_list)

test_name = '<yourname>test<no>'
url='https://api.thousandeyes.com/v6/tests/agent-to-server/new.json'
payload = {'interval': '300', 'agents': empty_list, 'testName': test_name, 'port': '80', 'server':
'www.thousandeyes.com','alertsEnabled': '0'}
header = {'content-type': 'application/json', 'authorization': 'Bearer ' + 'token'}
r = requests.post(url, data=json.dumps(payload), headers=header, verify=False)
print(r)
```

Response:

```
<applications><application>
```

Please note that the placeholders <token> and <yourname> should be replaced with the actual token and your desired name, respectively

Summary

This lab focuses on three key tasks: cloning the repo, leveraging AppDynamics APIs and ThousandEyes APIs..

Task 1 Participants begin by cloning a repository related to Supercar-Trader, ensuring efficient access and contributions to the codebase.

Task 2 guides users through AppDynamics API interactions, including Python script usage to retrieve and update information within the Supercar-Trader Application.

Task 3 covers ThousandEyes APIs, facilitating the retrieval of enterprise agent lists and creating web tests for Supercar-Trader using Python scripts.

Congratulations! You have completed the FSO API Lab.