CAN INSTALLATION AND CONFIGURATION

Cognitive Assistant for Networks (CAN) 7.0

AUGUST 7, 2023

AVANSEUS TECHNOLOGIES PVT. LTD.

REVISION HISTORY

Version	Date	Change description	Created by	Updated by	Reviewed by
V 1.0	November 2022	Release 6.0	Naveen	Sandeep Singh	Chiranjib
V 2.0	August 2023	Beta Release 7.0	Vidhya	Raksha	Chiranjib

Table of Contents

1. Initial Configuration Steps....2
2. Manual Verification Steps (Using Utilities)....8
3. Notes on CAN Entity Hierarchy and Their Usage....13
4. Notes on Reinstallation of CAN....15

1. Initial Configuration Steps

1. Setup the OpenDJ LDAP, MongoDB, VBI, PredictionController, PredictionWorker, BatchHandler & RecordProcessor modules using the Helm charts mentioned in the MOP documents (There may be other modules, which are optional for CAN to run).
2. Setup the CAN pod by configuring the “config.properties” ConfigMap correctly. i.e., IP’s,

   Ports, service names and encrypted passwords. See the sample content below:

```
#Domain details
avanseus.app.can.domain=avanseuscanintegration.com

#VBI configuration
avanseus.vbi.ip=vbicontainer
avanseus.vbi.port=12001

#Domain protocol
avanseus.protocol=https

#Host details
avanseus.app.can.host=canapp:2000

#Memcached configuration
avanseus.memcached.url=memcached:11211

#MongoDb configuration
avanseus.mongodb.ssl.enabled=true
avanseus.mongodb.validHostName=false
avanseus.mongodb.keystore.path=/data/workspace/pemfile/mongo.pkcs12
avanseus.mongodb.keystore.password=r7qs0Kobh5s+q0Wtlj1JZbtkdqGxBdx/

avanseus.mongodb.ip=database
avanseus.mongodb.port=30001
avanseus.mongodb.username=canupgradedb
avanseus.mongodb.password=qsoAXxeOGkmqvmFXbKvKd8u14350Trdi
avanseus.mongodb.dbName=canupgradedb

avanseus.mongodb.admin.username=admin
avanseus.mongodb.admin.password=qsoAXxeOGkmqvmFXbKvKd8u14350Trdi
avanseus.mongodb.admin.dbName=admin
```

```
#PostgreSQL configuration
avanseus.postgresql.url=jdbc:postgresql://postgresql.keycloak.svc.cluster.local:5432/keycloak
avanseus.postgresql.username=keycloak
avanseus.postgresql.password=NXkYoyjKahGcbvbjpS910fTiruHd4DHd

avanseus.app.googleKey=AIzaSyBIvCdlZrceLDdNMzLPOVPzS2ZisLfne4k

#log configuration
avanseus.log.path=/data/workspace/logs/
avanseus.log.thread.poolsize=20

#LDAP server configuration
avanseus.ldap.baseDN=o=employee,dc=avanseus,dc=com
avanseus.ldap.bindDN=cn=Directory Manager
avanseus.ldap.bindPassword=D+nmhhDVgn9B+l0lnv4sig==
avanseus.ldap.authFilter=uid=%u
avanseus.ldap.url1=ldaps://opendjldap:1636
avanseus.ldap.url2=ldaps://opendjldap:1636
avanseus.ldap.ssl.enabled=true
avanseus.ldap.keystore.path=/data/workspace/pemfile/ldap.jks
avanseus.ldap.keystore.password=7T76ayOUOK6piG86h/pea6L/+RE3Kw4M
avanseus.ldap.validHostName=false
avanseus.ldap.readyOnly=false

#NAS path
avanseus.app.nas.path=/data/workspace/nasmount/

#Email Config
avanseus.email.fromId=can.admin@avanseus.com
avanseus.email.pwd=Candev@53us
avanseus.email.host=smtp.office365.com
avanseus.email.fromName=CAN Test server
avanseus.email.port=587
avanseus.email.startTlsEnabled=true

#Cluster node configuration
```

```
avanseus.predictionNode.name=nodeA

#Python processor node configuration
avanseus.pythonprocessor.host=pythonprocessornode
avanseus.pythonprocessor.port=7001

#Ansible processor node configuration
avanseus.ansibleprocessor.host=ansibleprocessornode
avanseus.ansibleprocessor.port=3000

#Customer Experience (boolean)
avanseus.can.customerExperienceRequired=false

#Javascript processor node configuration
avanseus.jsprocessor.host=jsprocessornode
avanseus.jsprocessor.port=7002

#PCP node configurations
avanseus.app.pcp.domain=avanseuscanintegration.com
avanseus.app.pcp.host=pcpnode:5056
```

3. Setup setenv.sh properties with relevant Java heap space for CAN application in the ConfigMap. Mentioned below is a default sample with Email SSL protocol version, Xms & Xmx configuration in setenv.sh

```
JAVA_OPTS="$JAVA_OPTS -Dmail.smtp.ssl.protocols=TLSv1.2 -Dcom.sun.jndi.ldap.object.disableEndpointIdentification=true -Djdk.serialSetFilterAfterRead=true -Xms1024m -Xmx2048m"
```

4. Create a user in MongoDB and import the collections that are available in “MasterTables” directory of release repository. The collections and their purposes are as follows:

Entity name	Purpose
3GPP5GConfigProperties	Contains 3GPP configuration properties
5bed7c01fd9b9d519fedd798, 5bed7c22fd9b9d519fedd79b, 5d0ce5a1cbaaab22bc8aa286, 5bed7f6afd9b9d519ffb5776, Resource & ResourceEntity	Contains default data for resource file load scree
Cause	Default alarm causes

Entity name	Purpose
AnsibleWorkflowConfigurationTemplate	Contains master code template used for processing the task output
AnsibleWorkFlowConfig	Contains ansible workflow schema configurations
AnsibleRuleConfiguration	Consists of equipment component, cause & workflow schema mappings
AnsibleJob	Contains ansible job execution details
AnsibleTask	Contains ansible task execution details
AnnouncementRuleConfiguration	Contains sample data for Announcement rule configuration screen
AlertEmailConfiguration	Contains initial data for Notification handler screen
DataCollectionConfigurationTemplate	Contains master code template used for Data Collection & Configuration
DefaultComplexCodeROE	Contains default configuration for ROE screen
DefaultPolicyConfiguration	Contains default policy configuration, which will be used to retrieve the original state of a default policy in the policy configuration screen.
DomainIcon	Contains default icon paths for default domains
Config	Generic configuration table
CauseStandardization	Consists of rawCause(raw data), typifiedCause and its associated standardCause as fields.
RegexCauseStandardization	Consists of regex for pattern matching for rawCause and its associated standardCause as fields.
EmployeeAccess	Contains the initial user id as 'canadmin'.
EntityFilterMaster, EntityFilterMasterQuery & EquipmentIcon	Contains default values for Cross domain correlation screen
EquipmentComponent	Default equipment components
EventFileFormat	Contains default data for Parser screen
EventFileFormatTemplate	Contains master code template used for parser
ExcelPageConfiguration	Contains default data for Excel page configuration screen

Entity name	Purpose
ExcelPageConfigurationTemplate	Contains master code template used for excel report generation
ExcelReportConfiguration	Contains default data for Excel report configuration screen
FieldLearntCauses	Contains default data of Root cause learnt from the field
FileCollectionConfiguration	Contains default data for file collection screen
JavaScriptTemplate	Contains javascript code template used for javascript execution
KafkaConfigProperties	Contains default configurations for Kafka
PerformanceCounterCause	Contains distinct KPI names for which we will run the prediction.
PostParsingProcessingLogic	Contains a sample for record splitting logic, which was used in 1 of the samples added in record parser.
PostPredictionProcess	Contains default data for Post prediction process screen
PostPredictionProcessTemplate	Contains master code template for Post Prediction. It consists of templateName and templateCodeSnippet as its fields.
PostProcessorTemplate	Contains master code template used for parser's post-processor
Postprocessor	Contains default data for post processor screen
PostParsingTemplate	Contains templates for Spitting and grouping screens of file-postprocessor
PreProcessorTemplate	Contains master code template used for parser's pre-processor
PredictionFilter	Contains master code logic for prediction filtration
PredictionFilterTemplate	Contains master code template for prediction filtration
PredictionKeyFilter	Contains default data for prediction filter screen
PredictionFilterRule	Contains filter rules
PredictionNode	Contains default data for Prediction nodes. Default is 3 nodes as of now.

Entity name	Purpose
PredictedFaultToTicketMappingTemplate	Contains template needed to compile PredictionToTicketMapping code
Preprocessor	Contains default data for pre-processor screen
PythonTemplate	Contains python code template used for python execution
RemedyConfig	Contains remedy config values such as cron, mean threshold closed days
RemedyFieldValue	Contains fieldId and field names of remedy internal fields
RoeConfig	Stores default RoE configuration
RoeConfigTemplate	Contains master code template for RoE configuration
RoePolicyConfiguration	Stores default policy configuration for Policy configuration screen
RoeSheetConfiguration	Stores default sheet configuration for roe sheet configuration screen
Role	Contains initial role for “canadmin”. Only Super Admin role would be present in Role table by default.
RootCauseAnalysis & RootCauseAnalysisEntity	Contains default data for Root cause analysis configuration screen
SplunkFields	Contains splunk search job related fields and their default values
TechnicalAnalysedCauses	Contains default data of Root cause learnt from technical analysis
TopologyStitchingConfiguration	Contains default code for Topology Stitching Configuration screen
TopologyStitchingConfigurationTemplate	Contains master code template used for Topology Stitching Configuration screen
TSDisplayElements	Contains link & network element type default mapping for Topology Stitching
ThresholdConfiguration	Contains threshold type (min/max) and the threshold value for each of the KPI
VBIAnnotations	Contains necessary information for VBI module related to possible questions voice based module accepts.

Entity name	Purpose
WeatherStatusResponseCode	Contains weather codes, weather description, color code and image icon path
WebServiceConfig	Contains an entry with a threshold limit used for the maximum number of Prediction delivery API requests served from the CAN application to the client in a day.

5. Run the file with Index scripts on MongoDB available in “CAN_Indices.txt” file located in “MasterTables” directory of GIT. These will create initial set of indices required for CAN application.

6. Also update the nasmount path in few DB collections by executing the below script.

```
var nasPath = "/data/workspace/nasmount/"

db.getCollection('5d23282ebb19a8c46c88c105').update({},{$set:{"filePath":nasPath+"/CAN/Resources/RANSharedSites/Report_SSI_H3G.XLS"}}, {multi:true})
db.getCollection('5d23285dbb19a8c46c88c128').update({},{$set:{"filePath":nasPath+"/CAN/Resources/PlannedWorks/Planned 01.05-05.11.2018.xlsx"}}, {multi:true})
db.getCollection('5d232805bb19a8c46c88c0f1').update({},{$set:{"filePath":nasPath+"/CAN/Resources/LongPendingTickets/Sol.xlsx"}}, {multi:true})
db.getCollection('5d232845bb19a8c46c88c117').update({},{$set:{"filePath":nasPath+"/CAN/Resources/SitePriority/Localization_sites.xls"}}, {multi:true})
db.getCollection('FieldLearntCauses').find({}).forEach(function(e){db.getCollection('FieldLearntCauses').update({filePath:e.filePath},{$set:{filePath:nasPath+e.filePath}})})
db.getCollection('PostPredictionProcess').update({"classFilePath":"/CAN/Generated_Dir"},{$set:{"classFilePath":nasPath+"/CAN/Generated_Dir"}},{})
db.getCollection('PostPredictionProcess').update({"javaFilePath":"/CAN/Generated_Dir/postPredictionProcessUploadedFile"},{$set:{"javaFilePath":nasPath+"/CAN/Generated_Dir/postPredictionProcessUploadedFile"}},{})
db.getCollection('ResourceEntity').find({}).forEach(function(e){db.getCollection('ResourceEntity').update(
{filePath:e.filePath},{$set:{filePath:nasPath+e.filePath}})})
db.getCollection('RootCauseAnalysisEntity').find({}).forEach(function(e){db.getCollection('RootCauseAn                alysisEntity').update({filePath:e.filePath},{$set:{filePath:nasPath+e.filePath}})})
db.getCollection('TechnicalAnalysedCauses').update({"filePath":"/CAN/RootCauseAnalysis/technicalA
nalysis/SampleFileForTechnicalAnalysis.xlsx"},{$set:{"filePath":nasPath+"/CAN/RootCauseAnalysis/ technicalAnalysis/SampleFileForTechnicalAnalysis.xlsx"}},{multi:true})
```

NOTE: In the above script, replace the nasmount path value with respective nasmount directory of deployment environment.

7. Deploy CAN master helm chart.

8. Configure the file collection & parser.

2. Manual Verification Steps (Using Utilities)

NOTE: URL Utilities mentioned below are run using the Javascript developer console in the browser after CAN login is successful. Due to security reasons, we do not allow direct URL access via GET method and hence “window.redirect()” JS utility is provided in invoke POST requests to mentioned below URL utilities.

1. Run the file collection & parsing:

   URL: http://<domain>/CAN/pickupData

In developer console, use the following after login:

```
window.redirect("pickupData", {}, "_blank");
```

Example: window.redirect("pickupData", {}, "_blank"); This step is to load the data.

2. Create the "PredictionKeyFilter" table entries for all causes having default rule (0, 0):
   URL: http://<domain>/CAN/functionalConfigurations/generateFilterKeys
   In developer console, use the following after login:

```
window.redirect("functionalConfigurations/generateFilterKeys", {filterId: "<filter_id>"},
"_blank");
```

   Example: window.redirect("functionalConfigurations/generateFilterKeys", {filterId: "5746ab0a66bba21bf99dc004"}, "_blank");

3. Running clustering:

   URL: http://<domain>/CAN/functionalConfigurations/performCluster

   In developer console, use the following after login:

```
window.redirect("functionalConfigurations/performCluster", {},
"_blank");
```

   Example: window.redirect("functionalConfigurations/performCluster", {}, "_blank");

4. Running prediction:

   URL: http://<domain>/CAN/predictionService/predict

   In developer console, use the following after login:

Alarm - window.redirect("predictionService/predict", {"time": "dd-MM-yyyy HH:mm, dd-MM-yyyy HH:mm, ……","predictionType":"ALARM"},"_blank");

KPI - window.redirect("predictionService/predictKPI", {"time": " dd-MM-yyyy HH:mm, dd-MM-yyyy HH:mm, ……","predictionType":"KPI"},"_blank");

Health Index - window.redirect("predictionService/predictHealthIndex", {"time": " dd-MM-yyyy HH:mm, dd-MM-yyyy HH:mm, ……","predictionType":"SUPERPOSED"},"_blank");

Multiple Type of Prediction - window.redirect("predictionService/predictAll", {"time": " dd-MM-yyyy HH:mm, dd-MM-yyyy HH:mm, ……"},"_blank");

(In this scenario the predictionType key in config table has to be changed)

Example:

Alarm - window.redirect("predictionService/predict", {"time": "07-01-2020 00:00,08-01-2020 00:00","predictionType":"ALARM"},"_blank");

KPI - window.redirect("predictionService/predictKPI", {"time": "07-01-2020 00:00,08-01-2020 00:00","predictionType":"KPI"},"_blank");

Health Index - window.redirect("predictionService/predictHealthIndex", {"time": "07-01-2020 00:00,08-01-2020 00:00","predictionType":"SUPERPOSED"},"_blank");

Multiple Type of Prediction - window.redirect("predictionService/predictAll", {"time": "07-01-2020

00:00,08-01-2020 00:00"},"_blank");

(In this scenario, the predictionType key in config table has to be changed)

5. Generate the fault trace:

   URL: http://<domain>/CAN/functionalConfigurations/generateAndUpdateFaultTrace

   In developer console, use the following after login:

```
window.redirect("functionalConfigurations/generateAndUpdateFaultTrace",
{"historyDays": "<days>","startDate": "<dd-mm-yyyy>", "endDate": "<dd-mm-yyyy>"
}, "_blank");
```

   Example: window.redirect("functionalConfigurations/generateAndUpdateFaultTrace", {"historyDays": 30","startDate": "11-01-2018", "endDate": "15-01-2018" }, "_blank");

   NOTE: The “endDate” parameter is optional. It may not be given, if only one-day fault trace has to be updated.

6. Generate the fault history:

   URL: http://<domain>/CAN/functionalConfigurations/updateFaultHistory

   In developer console, use the following after login:

```
window.redirect("functionalConfigurations/updateFaultHistory",
{"startDate": "<dd-mm-yyyy>", "endDate": "<dd-mm-yyyy>" }, "_blank");
```

   Example: window.redirect("functionalConfigurations/updateFaultHistory", {"startDate": "11-01-2018", "endDate": "15-01-2018" }, "_blank");

   NOTE: The “endDate” parameter is optional. It may not be given, if only one-day fault history has to be updated.

7. Configure the Excel report format & mailing list.

8. Generate the particular day’s report:

   URL: http://<domain>/CAN/excelReport/downloadReport

   In developer console, use the following after login:

```
window.redirect("excelReport/downloadReport", {"time":
"<dd-mm-yyyy>", "predictionType": "<prediction_data>"
}, "_blank");
```

   Example-1: window.redirect("excelReport/downloadReport", {"time": "10-04-2019", "predictionType":"ALARM" },"_blank");

   Example-2: window.redirect("excelReport/downloadReport", {"time": "01-10-2019 00:00:05", "predictionType":"KPI" }, "_blank");
   Eg-3: window.redirect("excelReport/downloadReport", {"time": "18-10-2019 05:30:00", "predictionType":"Health Index" }, "_blank");

9. Generate RoE for an interval of dates:

   URL: http://<domain>/CAN/functionalConfigurations/updateRoePredictions

   In developer console, use the following after login:

```
window.redirect("functionalConfigurations/updateRoePredictions", {"startDate": "<dd-mm- yyyy>",
"endDate": "<dd-mm-yyyy>" }, "_blank");
```

Example: window.redirect("functionalConfigurations/updateRoePredictions", {"startDate": "11-01-2018", "endDate": "15-01-2018" }, "_blank");

10. Generate the particular prediction window's matching report:

    URL:

    http://<domain>/CAN/predictiveFaultAnalytics/downloadConsolidatedAlarmMatchingReport

    In developer console, use the following after login:

    ```
    window.redirect("predictiveFaultAnalytics/downloadConsolidatedAlarmMatchingReport", { windowIdentifier: "<time_in_milliseconds_of_the_first_day_of_prediction_window>" }, "_blank");
    ```

    Example: window.redirect("predictiveFaultAnalytics/downloadConsolidatedAlarmMatchingReport", { windowIdentifier: "1556060000000"}, "_blank");

11. Formatted Predicted Fault table will be generated automatically after Prediction. If the table is not present in the database, then create that table manually by running:

    URL: http://<domain>/CAN/ predictiveFaultAnalytics/generateFormattedPredictedFault

    In developer console, use the following after login:

    ```
    window.redirect("predictiveFaultAnalytics/generateFormattedPredictedFault", { startDate: "dd- MM-yyyy", endDate: "dd-MM-yyyy" }, "_blank");
    ```

    Example: window.redirect("predictiveFaultAnalytics/generateFormattedPredictedFault", { startDate: "29-04-2019", endDate: "30-04-2019"}, "_blank");

    NOTE: To run the above URL, please make sure that all the code snippets written in the Page Configuration screen are compiled and saved.

12. Rest API to run alarm post prediction activities

    URL: http://<domain>/CAN/predictionService/runPostPredictionActivity

    Request parameters:

    a. **time** – Prediction dates separated by "," (Date format should be dd-MM-yyyy).

    b. **initialExecutionStatus** – Initial execution status of all post-prediction executors. The value should be either true or false. (True indicates all post-prediction executors will be enabled. False indicates all post-prediction executors will be disabled.)

    c. **enableOrDisablePostPredictionExecutorsList** – Mapped index of post-prediction executors separated by ",". (The given post-prediction executors will get enabled/disabled based on the value of 'InitialExecutionStatus')

    Relation between InitialExecutionStatus and enableOrDisablePostPredictionExecutorsList parameters:

    a. **Use case 1:**

       initialExecutionStatus:true , *enableOrDisablePostPredictionExecutorsList*:"1,2,4"

       This enables all post-prediction executors except the executors, which are mapped to indexes 1, 2 and 4.

    b. **Use case 2:**

initialExecutionStatus:false , *enableOrDisablePostPredictionExecutorsList*:"2,3,4"

This enables only the executors, which are mapped to indexes 2, 3 and 4. Others will be disabled.

```
window.redirect("predictionService/runPostPredictionActivity", {"time": "<dd-MM-yyyy, dd-MM-yyyy, …..>", "initialExecutionStatus" : <true/false>, enableOrDisablePostPredictionExecutorsList:"<comma_separated_executor_index_list> "}, "_blank");
```

Example: window.redirect("predictionService/runPostPredictionActivity", {"time": "20-04-2022, 21-04-2022, 22-04-2022, 23-04-2022, 24-04-2022, 25-04-2022, 26-04-2022, 27-04-2022, 28-04-2022, 29-04-2022, 30-04-2022, 01-05-2022, 02-05-2022, 03-05-2022, 04-05-2022, 05-05-2022, 06-05-2022, 07-05-2022, 08-05-2022, 09-05-2022, 10-05-2022", "initialExecutionStatus" : true, enableOrDisablePostPredictionExecutorsList:"1,2"}, "_blank");

Available post-prediction executor and its mapping index:

Post-Prediction Executor	Mapped index
APPLY_FILTER	1
DUMP_OUTPUT_FILE	2
APPEND_CLUSTER_ID	3
CUSTOMIZED_POST_PREDICTION	4
TICKET_CORRELATION	5
UPDATE_REPETITIVE_ALARM	6
UPDATE_FAULT_HISTORY	7
UPDATE_SOLUTION_REPORT	8
UPDATE_FAULT_TRACE	9
UPDATE_EFFECTIVE_FAULT_HISTORY	10
RCA_ANALYSIS	11
UPDATE_TOPOLOGY_PATH_SEQUENCE	12
PC_CORRELATION_ANALYSIS	13
ALARM_DURATION_PREDICTION	14
PUID_GENERATION	15
ENABLE_ROE	16
SEND_EMAIL	17

FORMATTED_PREDICTED_FAULT_CREATIO N	18
ANSIBLE_JOB	19
ALARM_MATCHING	20
PC_MATCHING	21

13. Initiate the call and repair probability calculation

    URL: http://<domain>/CAN/bcxp/predictCallRepairLikelihood

    In developer console, use the following after login:

    ```
    window.redirect("bcxp/predictCallRepairLikelihood", { time: “<dd- MM-yyyy>, <dd- MM-yyyy>…” }, "_blank");
    ```

    Example: window.redirect(("bcxp/predictCallRepairLikelihood", { time:”29-04-2019,30-04-2019” }, "_blank");

14. Run Weather predictions for a particular interval

    URL: http://<domain>/CAN/integrationConfiguration/weatherConfiguration/generateWeatherDataForGivenDate

    In developer console, use the following after login:

    ```
    window.redirect("integrationConfiguration/weatherConfiguration/generateWeatherDataForGivenDate", {"startTime":"dd-mm-yyyy hh:mm:ss","endTime":"dd-mm-yyyy hh:mm:ss"},"_blank")
    ```

    Example: window.redirect("integrationConfiguration/weatherConfiguration

    /generateWeatherDataForGivenDate", {"startTime":"14-01-2023 00:00:00","endTime":"15-01-2023 23:59:00"},"_blank")

15. Run Ansible jobs for a prediction day

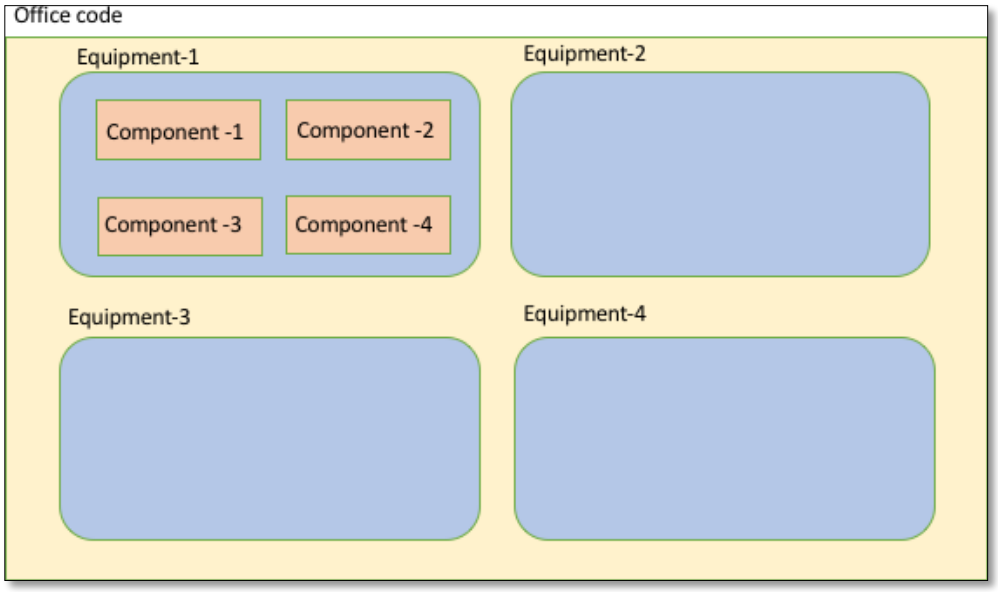
    URL: http://<domain>/CAN/ansibleExecution/runAnsibleJobs

    ```
    window.redirect("ansibleExecution/runAnsibleJobs", {"time": "dd-mm-yyyy"}, "_blank");
    ```

    Example: window.redirect("ansibleExecution/runAnsibleJobs", {"time": "07-01-2023"}, "_blank");

3. Notes on CAN Entity Hierarchy and Their Usage

A pictorial representation of Office code and internal components is shown below.



From the above picture, it is clear that a Site/Office code will have multiple equipment installed in it and each equipment will have multiple components. These components can be equipment's port, slot, rack etc.

In CAN application, multiple entities define these Site components. The multiple entities are:

1. Office Code - DB collection would be Office Code
2. Equipment - DB collection would be Equipment
3. Equipment Component - DB collection would be Equipment Component

Above entities are made mandatory in the Parser screen for the Alarm configuration.

The table shown below contains different operations along with the Entity level.

Operation	Entity level
Prediction	Equipment component
Fault history	Equipment
Fault trace	Equipment
Clustering/Cross domain correlation	Office code
RC analysis	Equipment
ROE (Return on effort)	Equipment
Topology Stitching (Cross Domain Topology Discovery)	Equipment
Ticket history	Equipment

4. Notes on Reinstallation of CAN

1. CAN application now runs in multiple instances in a cluster mode for high availability when accessing the UI. Due to this reason, the Cron triggers used in CAN application across many use cases are made cluster aware. Quartz scheduler is now cluster aware and stores the state of each Quartz trigger in MongoDB. Whenever CAN application is reinstalled or installed after a long period, there are some stale triggers in the database, which may trigger the moment the application comes up. This is good since the application needs to be updated with the latest data. However, there may be cases where the trigger is very old and you may not require that to run. For this, use the below mentioned collections from the database and then start the application.
    - quartz_calenders
    - quartz_jobs
    - quartz_locks
    - quartz_triggers
2. Batch Handler & Record processor caches
    - **Batch handler cache:** It contains file preprocessor cache that is refreshed in every 5 minutes.
    - **Record processor cache:**
        - **Record postProcessor cache** is refreshed in every 5 minutes.
        - **Alarm Filter cache** is refreshed in every 5 minutes.
        - **Event file format cache** is refresh every 5 minutes and contains maximum 20 entries.
        - **Respective master tables** are also stored as cache that is not refreshed. Hence, clearing collections of any master tables requires restart of Record processor. By default, this cache contains 500000 entries and stores ids.
        - Any other configurations to this cache can be done in recordProcessorCache field of Config table currently this is not visible in UI. Here size denotes size of cache and fields to be stored in cache are inside fields.

```
{
"_id":ObjectId("63861da42510e142e21b4ca5"),
"key":"recordProcessorCache",
"value":{

"default":{
"size":NumberInt(100000),
},
"Cause":{
"size":NumberInt(50000),
"fields":[
"serviceAffecting",
"domain",
```

```
"priority"
]
},
"OfficeCode":{
"size":NumberInt(50000),
"fields":[
"officeCodePriority"
]
},
"Equipment":{
"size":NumberInt(50000),
}
}
}
```