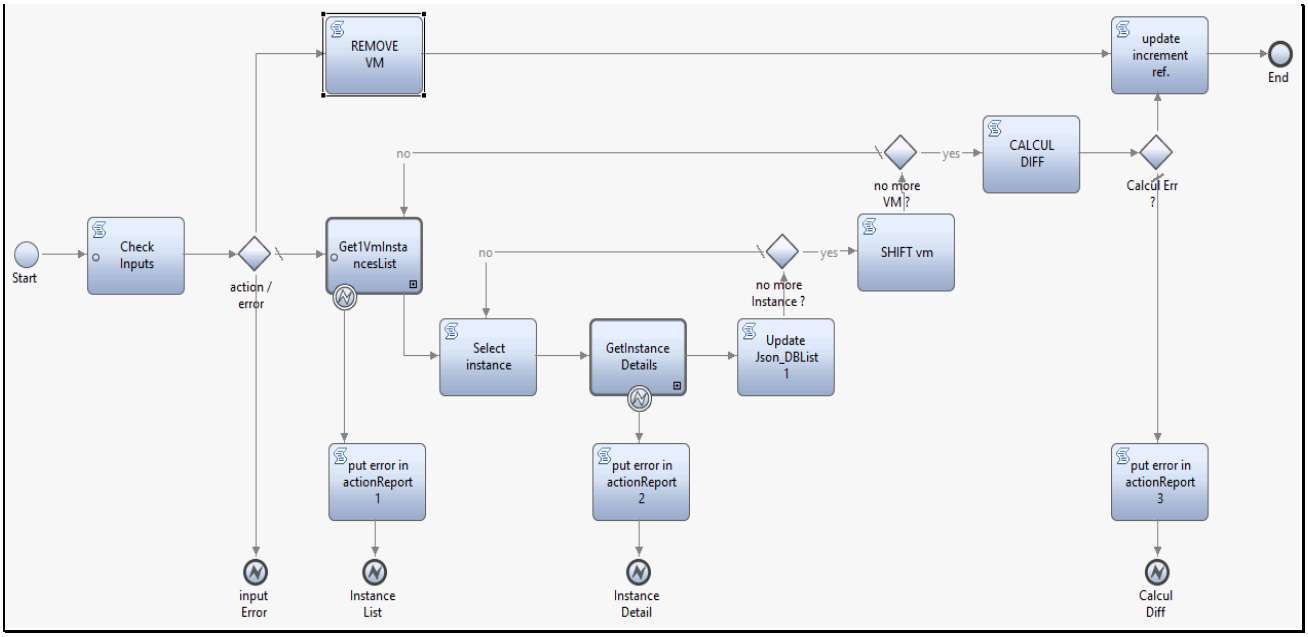
LISTER BASE

Laurent Graignic le 05/02/2018

OBJECTIFS :

1) Notifier tout changement de configuration Oracle (Orchestré ou non) intervenu sur des groupes de VM.
2) Fournir un référentiel d'incréments utilisables, (Lettres encore disponibles entre A et Z) pour chaque nom d'instance Oracle (Utilisé dans le cadre ou non de l'orchestration).
3) Pouvoir supprimer les informations obsolètes

DESIGN DE LA SOLUTION :



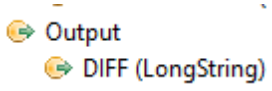
3 actions possibles :

1) **UPDATE** (Pour détecter et notifier les changements)
2) **REMOVE** (Pour retirer les informations d'une ou plusieurs VM obsolètes)
3) **REMOVE_ALL** (Pour nettoyer la base sans connaître sa composition)

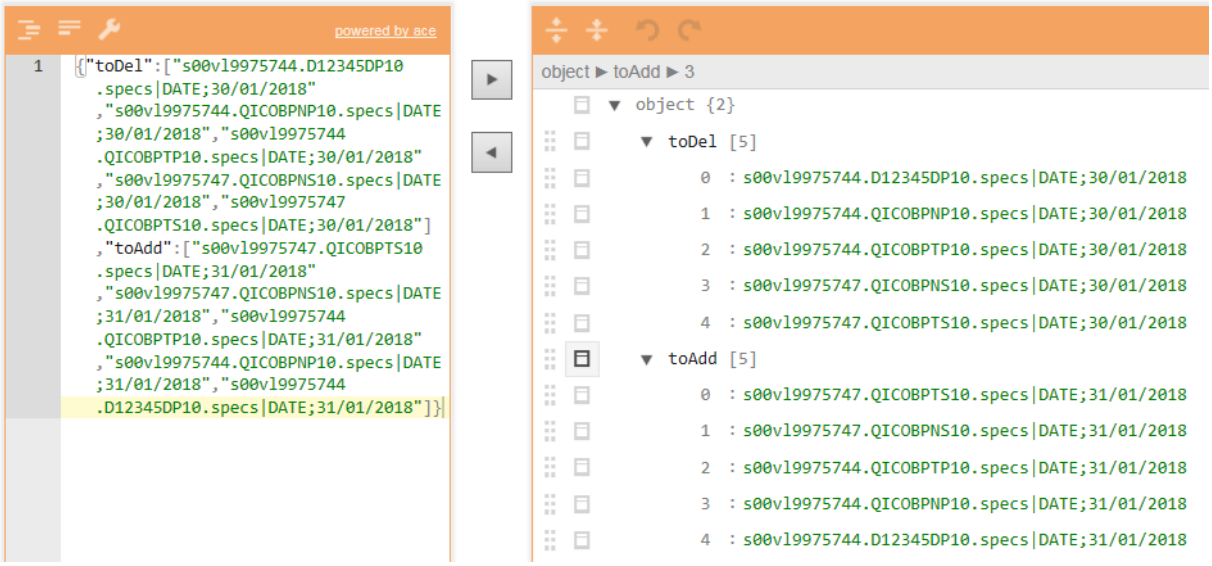
Action « UPDATE »

OUTPUTS :

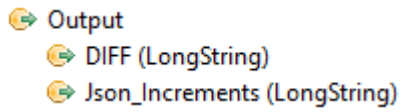
Les notifications de changement de configuration Oracle sont fournies dans l'Output « DIFF »



EXEMPLE :



Le référentiel pour les incréments disponibles est fourni dans l'output « Json_Increments »



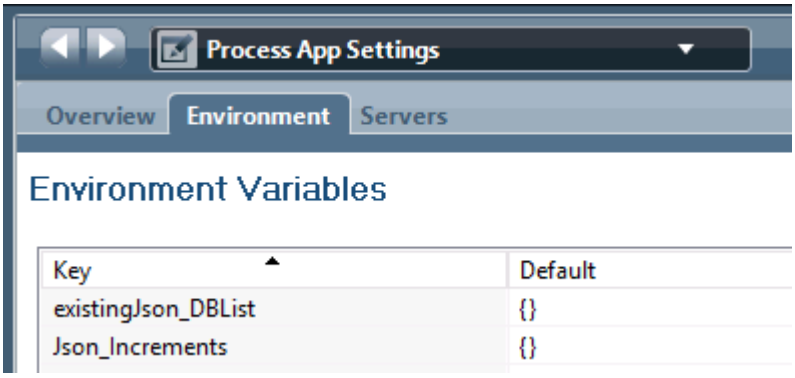
EXEMPLE :

*Noter, à droite, le nombre de lettres disponibles par instance (*25 et 23*) et leurs valeurs à gauche.*

IMPLICATIONS

1) Stocker d'une fois sur l'autre les informations de configuration Oracle pour pouvoir ensuite les comparer entre elles.
2) Stocker l'Output « Json_Increments » pour le mettre à disposition d'autres process sans passer par LISTER_BASE.

SOLUTION : Utiliser la capacité de stockage permanant existant dans les « Process App Settings » de BPM:



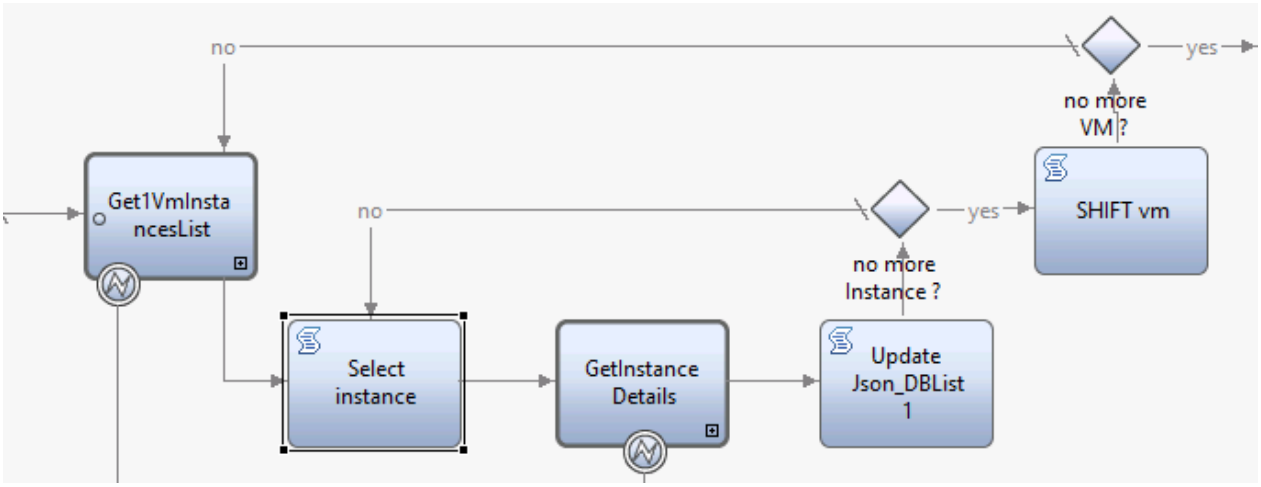
On voit ci-dessus, les deux variables qui sont mises à jour par le process à chaque fin d'exécution.

Note : L'affichage des valeurs initiales « {} » reste inchangé suite à leur mise à jour par le process.

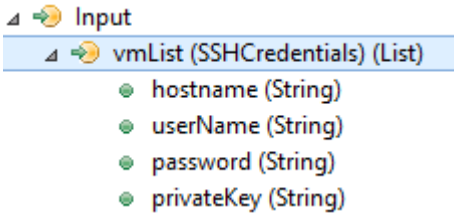
Implémentation au sein du process :

- `tw.env.update(tw.system.model.env.existingJson_DBList, JSON.stringify(jsonPreviousScan),true)`
- `tw.env.update(tw.system.model.env.Json_Increments, tw.local.Json_Increments,true)`

COLLECTE DES INFORMATIONS DE CONFIGURATION ORACLE :



Cette double boucle est alimentée par l'input « vmList » :



« vmList » est une liste de « SSHCredentials ». Autrement dit, une liste de VMs.

La première tâche, **« Get1VmInstancesList »,** fournit la liste des instances hébergées dans la VM courante.

Mapping de **« Get1VmInstancesList»** :

« Select instance » sélectionne ensuite l'instance courante et la passe en input de « **GetInstance Details** » qui en fournit les détails. Ces détails sont ensuite passés en input de **«Update json_DBList»** qui les ajoute dans l'objet JSON **currentJson_DBList.** Celui-ci est alors réinjecté sous forme « string » dans le circuit en attente du traitement de l'instance suivante. Quand la dernière instance est traitée, une autre VM devient la VM courante, et ainsi de suite. A la dernière instance de la dernière VM, on sort de la boucle pour passer aux calculs.

Détails des appels ssh

Pour chacune des VM, la présence du fichier tar (Qui contient les scripts Oracle) est testée. En cas d'absence, la VM téléchargera le tar en https depuis un repository web (wget) puis le décompressera.

Une fois les scripts présents sur la VM, celui permettant de lister les instances est lancé. Le stdOut est parsé localement sur la VM puis la liste obtenue est mise en forme dans **currentInstance**(string) selon la forme:
«INSTANCE1,INSTANCE2,INSTANCE3»

7 commandes bash par instance sont réparties dans 3 connexions ssh :

1) Check tar, download tar et decompress tar
2) Get instances List
3) Get instances details

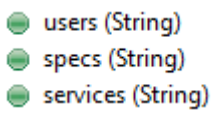
La connexion ssh de « Get instances details », réunie trois commandes lancées à la suite:

- db_show_parameters.ksh
- db_service_name.ksh
- db_user.ksh

Les informations sont alors préfixées localement selon ce format:

```
……
USERS:APPQOSSYS
USERS:AUDSYS
USERS:DBSNMP
SPECS:_use_single_log_writer;TRUE
SPECS:utl_file_dir;
SPECS:workarea_size_policy;AUTO
SPECS:xml_db_events;enable
SERVICES:QICOBPNS1
SERVICES:QICOBPNS10
…….
```

La tache suivante dispache les données de chaque catégorie dans 3 variables BPM privées (string), ou elles se trouvent sérialisées avec le séparateur « | » (Qui est un caractère toujours absent des données) :

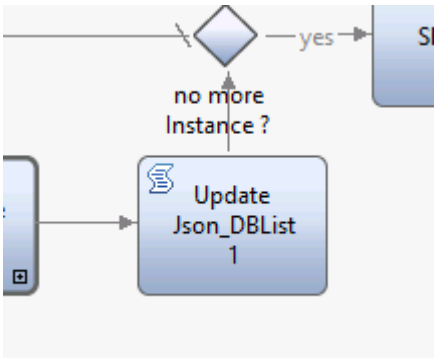



```
var specsTAB = [] ; var usersTAB = [] ; var servicesTAB = []
stdOutTAB.forEach(function(line){
    if(line.match(/^SPECS:/)) { specsTAB.push(line.replace(/^SPECS:/,"")) }
    if(line.match(/^USERS:/)) { usersTAB.push(line.replace(/^USERS:/,"")) }
    if(line.match(/^SERVICES:/)) { servicesTAB.push(line.replace(/^SERVICES:/,"")) }
})
tw.local.specs = specsTAB.join("|") //; log.info("tw.local.specs = "+tw.local.specs)
tw.local.users = usersTAB.join("|") //; log.info("tw.local.users = "+tw.local.users)
tw.local.services = servicesTAB.join("|") //; log.info("tw.local.services = "+tw.local.services)
```

Ce fonctionnement optimise la vitesse de collecte des informations.

Accumulation des données Oracles, instances après instances, VM après VM

Le principe est de parser le **json(string) courant** pour en faire un objet. D'y adjoindre les données de l'instance courante, puis de le « stringifyer » à nouveau pour le réinjecter au format string dans le circuit.




```
var users = [] ; if(tw.local.users != ""){ users = tw.local.users.split("|") } ; log.info(users.length+" users")
var specs = [] ; if(tw.local.specs != ""){ specs = tw.local.specs.split("|") } ; log.info(specs.length+" specs")
var services = [] ; if(tw.local.services != ""){ services = tw.local.services.split("|") } ; log.info(services.length+"
services")

var vmName = tw.local.currentVM.hostname
var currentJsonDBList = JSON.parse(tw.local.currentJsonDBList)
var instanceName = tw.local.currentInstance

// Create the main object "DBList" if it doesn't exist and fill it with an empty vm object if it doesn't exists, otherwise
empty the existing vm object.
if(!currentJsonDBList["DBList"]){
    currentJsonDBList["DBList"] = {}
}
if(!currentJsonDBList["DBList"][vmName]){
    currentJsonDBList["DBList"][vmName] = {}
}
if(!currentJsonDBList["DBList"][vmName][instanceName]){
    currentJsonDBList["DBList"][vmName][instanceName] = {}
}
currentJsonDBList["DBList"][vmName][instanceName]["users"] = users
currentJsonDBList["DBList"][vmName][instanceName]["specs"] = specs
currentJsonDBList["DBList"][vmName][instanceName]["services"] = services

tw.local.currentJsonDBList = JSON.stringify(currentJsonDBList)
```

Accumulation des données Oracles : STRUCTURE

object ▶ DBList ▶ s00vl9975744 ▶ QICOBPTP10 ▶ services ▶ 6

```
▼ object {1}
  ▼ DBList {2}
    ▼ s00vl9975744 {3}
      ▶ D12345DP10 {3}
      ▼ QICOBPNP10 {3}
        ▶ users [19]
        ▶ specs [413]
        ▶ services [8]
      ▼ QICOBPTP10 {3}
        ▶ users [19]
        ▶ specs [413]
        ▼ services [8]
            0 : QICOBPTP1
            1 : QICOBPTP10
            2 : QICOBPTP10_DGMGRL
            3 : QICOBPTP1_DGB
            4 : QICOBPTP1_DGMGRL
            5 : Q_ICOBP_00000_03
            6 : Q_ICOBP_00000_03_BATCH
            7 : Q_ICOBP_00000_03_TP
    ▶ s00vl9975747 {2}
```

Cette structure est adaptée à la finalité recherchée (Détection de différences entre 2 arbres de références) car les données de dernier niveau (services, users, specs) peuvent être exprimées par leur chemin, ce dernier pouvant constituer une référence unique à chaque donnée.

Exemple avec la data ci-dessus surlignée en jaune : On voit que

DBList.s00vl9975744.QICOBPTP10.specs[6] = «Q_ICOBP_00000_03_BATCH»

Or, ceci peut facilement être transformé en :

DBList.s00vl9975744.QICOBPTP10.specs|Q_ICOBP_00000_03_BATCH

Ou l'indice [6] qui était la seule référence flottante du chemin est remplacée par le caractère « | » suivi de la data. L'ensemble constituant une référence unique à la data.

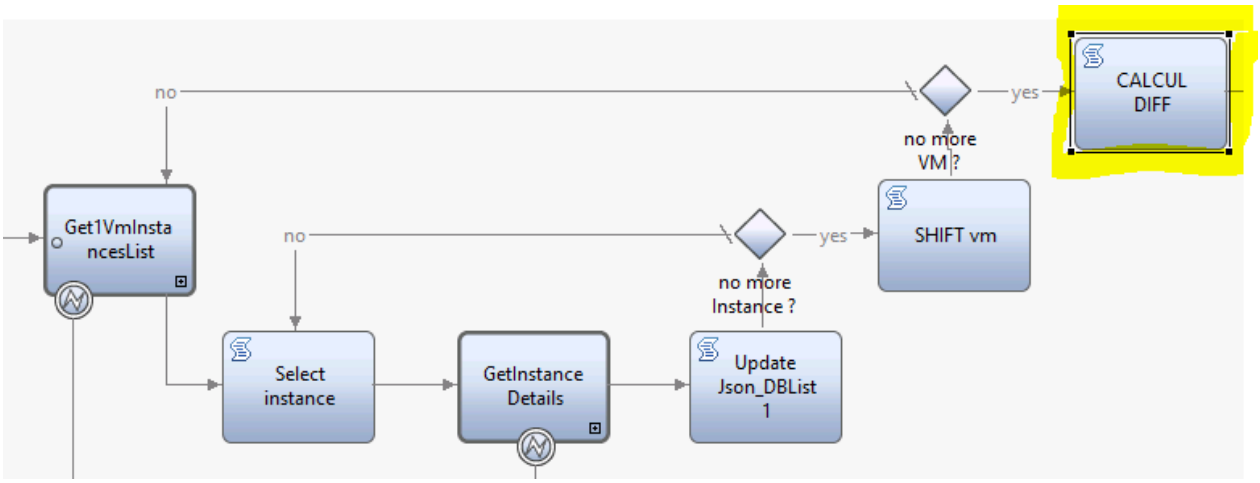
De ce fait, il est possible d'extraire toutes les données avec leur référence unique sous la forme d'une simple liste.

L'étape suivante consistant donc à comparer deux listes : Celle issue du json stocké précédemment et la liste issue du json courant (résultat de la collecte).

Deux fonctions restent alors à mettre en œuvre :

1) **Transformer un json en liste**
2) **Comparer deux listes**

Elles sont implémentées dans « CALCUL DIFF » :



TRANSFORMER UN JSON EN LISTE : (Avec le paramètre « inPath » limitant le scope de parcours de l'arbre)
(Fonction récursive basée sur l'évaluation d'expressions régulières et l'évaluation d'un nom d'objet JSON)

```
var OUT = []
function LookForID(JsonObj,finalKey,deep,inPath){
   for(var ELT in eval(JsonObj)) {
      var PATH = JsonObj+"[\""+ELT+"\"]"
      if(typeof eval(PATH) != "object") {
         if (ELT.match(eval(finalKey)) && PATH.match(eval(inPath))) {
            var LINE = JsonObj.replace(/\"]/g,"").replace(/\[\"/g,".").replace(/^json.*\.DBList\./,"")+"|"+eval(PATH)
            OUT.push(LINE)
            continue
         }
      }else{ if(deep){ LookForID(PATH,finalKey,deep,inPath) } else { continue } }
   }
   return OUT
}
```

UTILISATION de la fonction `LookForID(JsonObj,finalKey,deep,inPath)`

- `JsonObj(string)` : Nom de l'objet json a traiter
- `finalKey(string)` : UNUSED (Regexp filtrant sur les valeur finales) valeur = « /^..*$/ »
- `deep(booleen)` : Descendre ou pas dans l'aborescence de la structure
- **`inPath(string)` : Le paramètre « inPath » ne sera utilisé que sur le json stocké. (De façon à se limiter aux VM aussi présentes dans la collecte courante).** En effet le « json stocké » contient les informations relatives à toutes les VM déjà scannées auparavant. (Depuis le début de l'exploitation du BPM LISTER_BASE)

En revanche, le filtre « inPath », n'est pas exploité avec le json de la collecte courante (Sa valeur sera seulement positionné à « /^..*$/ »).

Le ARRAY résultant, pour chaque json, sera de la forme :

```
……
DBList.s00vl9975744.QICOBPTP10.specs|Q_ICOBP_00000_03_BATCH
DBList.s00vl9975744.QICOBPTP10. specs|Q_ICOBP_00000_04_BATCH
DBList.s00vl9975744.QICOBPTP10.users|oracle_1
DBList.s00vl9975744.QICOBPTP10.users|oracle_2
DBList.s00vl9975744.QICOBPTP10.services|DeNettoyageDesChiottes
……
```

COMPARER DEUX LISTES : (En jaune la partie efficace de l'algorithme) - Le reste du code est du contrôle.

```
function onlyUnique(value, index, self) { return self.indexOf(value) === index }
function singleton(value, index, self) { return self.indexOf(value) ===  self.lastIndexOf(value) }
function multiples(value, index, self) { return self.indexOf(value) !==  self.lastIndexOf(value) }
function ListDiff(last,next){
   var toKeep = [] ; var toAdd = [] ; var toDel = [] ; var ERR=false
   if(typeof last !== "object" || isNaN(last.length) ){ Notif("calDiff: \"last\" bad  argument",false) ; ERR=true }
   if(typeof next !== "object" || isNaN(next.length) ){ Notif("calDiff: \"next\" bad  argument",false) ; ERR=true }
   var lastMultiple = last.filter(multiples)
   if(lastMultiple.length != 0){
      Notif("Warning: previous scan has multiples :"+lastMultiple.join(" "),false) ; ERR=true
   }else{ last = last.filter(onlyUnique) }
   var nextMultiple = next.filter(multiples)
   if(nextMultiple.length != 0){
      Notif("Warning: current scan has multiples :"+nextMultiple.join(" "),false) ; ERR=true
   }else{ next = next.filter(onlyUnique) }
   if(!ERR){
       toKeep = toKeep.concat(last,next).filter(multiples).filter(onlyUnique)
       toDel  = toDel.concat(toKeep,last).filter(singleton)
       toAdd  = toAdd.concat(next,toKeep).filter(singleton)
      log.info("DIIF: toAdd = "+toAdd.length+" elts   ;   toDel = "+toDel.length+" elts")
      var DIFF = JSON.stringify({"toDel":toDel,"toAdd":toAdd})
      return DIFF
   }else{ return "" }
}
```

Principe : **Soit List1 et List2, deux listes à comparer**

List1 = A B C D (last)
| | -> **C et D** sont communs aux deux listes
List2 = C D E F (next)

Pour passer de List1 à List2, **A et B** sont à supprimer de List1 ; **E et F** sont à ajouter à List1

Calcul intermédiaire : Trouver « toKeep » (C et D):

`toKeep = toKeep.concat(last,next).filter(multiples).filter(onlyUnique)`

`toKeep.concat(last,next)` vaut «ABCDCDEF»
«ABCDCDEF».`filter(multiples)` vaut «CCDD»
«CCDD».`filter(onlyUnique)` vaut **«CD»** **OK**

Trouver « toDel » (A et B)

`toDel = toDel.concat(toKeep,last).filter(singleton)`

`toDel.concat(toKeep,last)` vaut «CDABCD»
«CDABCD».`filter(singleton)` vaut **«AB»** **OK**

Trouver « toAdd » (E et F)

`toAdd = toAdd.concat(next,toKeep).filter(singleton)`

`toAdd.concat(next,toKeep)` vaut «CDEFCD»
«CDCDEF».`filter(singleton)` vaut **«EF»** **OK**

DETAILS DU CALCUL :

Transformation en liste du scan courant

```
//#############################################################################################################################
var jsonDBList = JSON.parse(tw.local.currentJsonDBList)
OUT = [] ; var currentScan = LookForID("jsonDBList[\"DBList\"]","/..*/",true,"/..*/")
Notif("currentScan is "+currentScan.length+" long",false)
```

Fabrication de la Regexp filtrant sur les noms de VM présents dans les credentials

```
//#############################################################################################################################
var vmNamesRegexp = tw.local.vmNames.split(",")
vmNamesRegexp = vmNamesRegexp.join("\..*$|^.*DBList....").replace(/^/,"/^.*DBList....").replace(/$/,"\..*$/")
```

Transformation en liste de la partie du PreviousScan se rapportant aux VM communes

```
//#############################################################################################################################
var jsonPreviousScan = JSON.parse(tw.env.existingJson_DBList)
OUT = [] ; var PreviousScan = LookForID("jsonPreviousScan[\"DBList\"]","/..*/",true,vmNamesRegexp)
Notif("PreviousScan is "+PreviousScan.length+" long",false)
```

Output DIFF : *Si (no error)-> Copy infoVM_of_current sur infoVM_of_previous*

```
//#############################################################################################################################
tw.local.DIFF = ListDiff(PreviousScan,currentScan)

if(tw.local.DIFF != ""){ // si chaine vide : erreur détectée dans la fonction
    Notif("############ END OF CALCUL ############",true) ; log.info(tw.local.DIFF) ; log.info("")
    var vmNames = tw.local.vmNames.split(",")
    for(var I in vmNames){  // remplacement des vm du PreviousScan dans le cash
        var vmName = vmNames[I]
        delete jsonPreviousScan["DBList"][vmName]
        jsonPreviousScan["DBList"][vmName] = jsonDBList["DBList"][vmName]
        }
    tw.env.update(tw.system.model.env.existingJson_DBList, JSON.stringify(jsonPreviousScan),true)
    Notif("Updating the previous scan with the current one: SUCCEED",true)
}else{
    Notif("UPDATE aborted",true)
    var lgth = tw.local.actionReport.listLength
    tw.local.errors = tw.local.actionReport.listToNativeArray().slice(lgth-10, lgth).join(",")
```

Dernière étape (Communes à toutes actions): «update increment ref», fourni une mise à jour des incréments disponibles pour chaque pentagramme.

Il s'agit de lister tous les noms d'instances présentes dans « **existingJson_DBList** » (Fraichement mis à jour par l'update), et d'en extraire la partie correspondant aux « noms uniques d'instance » qui possèdent un incrément.

Un incrément est une lettre de l'alphabet.

La création de nouvelles instances rattachées à des instances existantes nécessite l'emploi du même pentagramme, mais associé à un nouvel incrément.

L'algorithme est le suivant :

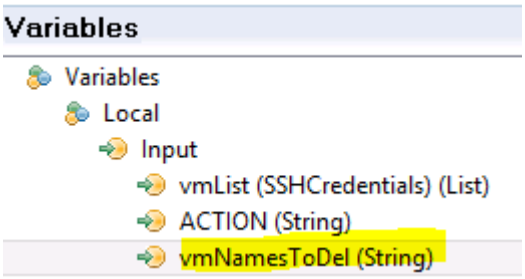
```
function onlyUnique(value, index, self) { return self.indexOf(value) === index }
function singleton(value, index, self) { return self.indexOf(value) ===  self.lastIndexOf(value) }
```

```
var jsonDBList = JSON.parse(tw.env.existingJson_DBList)
```

```
var uniqInstances = [] ; var uniqInstancesLettre = []
var instances = []
for(var vm_name in jsonDBList["DBList"]){
    var vms = jsonDBList["DBList"][vm_name]
    for(var inst_name in vms){
        var uniqInstanceLettre = inst_name.split("",7).join("") ; uniqInstancesLettre.push(uniqInstanceLettre)
        var uniqInstance = uniqInstanceLettre.split("",6).join("") ; uniqInstances.push(uniqInstance)
    }
}

uniqInstances = uniqInstances.filter(onlyUnique)
uniqInstancesLettre = uniqInstancesLettre.filter(onlyUnique)

var alfaB = ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"]

function returnIndice(fitre,data){
   var TAB = []
   var REGEXP = eval("/(^"+fitre+")([A-Z]$)/")
   TAB = REGEXP.exec(data)
   if(TAB){return TAB[2]}else{return ""}
}

var Json_Increments = {}
Json_Increments["uniqInstance"] = {}
var Lettres = []

for(var I in uniqInstances){
    var uniqInstance = uniqInstances[I]
    for(var II in uniqInstancesLettre){
    var uniqInstanceLettre = uniqInstancesLettre[II]
    var LettreToAdd = returnIndice(uniqInstance,uniqInstanceLettre)
    if(LettreToAdd != ""){ Lettres.push(LettreToAdd) }
    }
    var availableIncrements = alfaB.concat(Lettres).filter(singleton)
    Json_Increments["uniqInstance"][uniqInstance] = availableIncrements
}

tw.local.Json_Increments = JSON.stringify(Json_Increments)
tw.env.update(tw.system.model.env.Json_Increments, tw.local.Json_Increments,true)
```

ACTION REMOVE (vmName1,[vmName2,…])

Pour cette action, l'input est une suite de noms de VM passée à **vmNamesToDel(string) :**

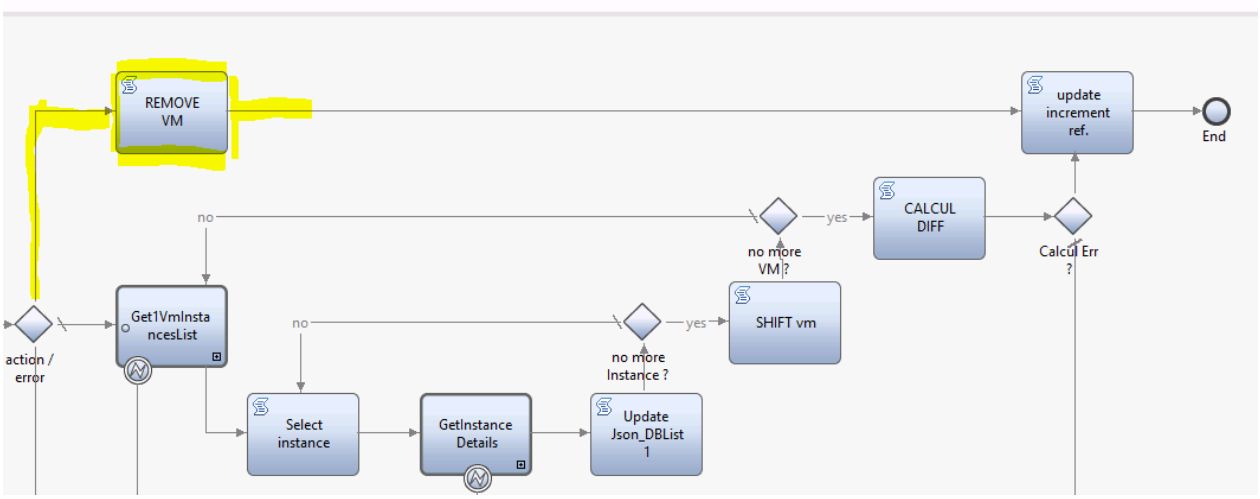


Le format est : « vmName1, vmName2,… »

ACTION REMOVE_ALL

Aucun autre input n'est requis pour cette action. Le fonctionnement est le même qu'avec REMOVE sauf que la liste des noms de VM est générée à partir de la base « existingJson_DBList » elle-même et ne provient pas d'un input.

Ces deux actions sont implémentées dans « REMOVE VM »




```
var vmNames = []
if(tw.local.ACTION == "REMOVE_ALL"){
    var vmNames = [] ; if(tw.local.vmNames != ""){ var vmNames = tw.local.vmNames.split(",") }// toutes les
vm de la base déterminées dans « check inputs »
}else{ // tw.local.ACTION == "REMOVE"
    var vmNames = tw.local.vmNamesToDel.split(",") // noms de vm fournis en input
}
log.info("") ; Notif("Deleting \""+vmNames.join(" and ")+"\" from database",true)

var jsonPreviousScan = JSON.parse(tw.local.existingJson_DBList)
for(var I in vmNames){
    var vmName = vmNames[I]
    delete jsonPreviousScan["DBList"][vmName]
}
Notif("\""+vmNames.join(" and ")+"\": deleted")
tw.env.update(tw.system.model.env.existingJson_DBList, JSON.stringify(jsonPreviousScan),true)
Notif("existingJson_DBList has been updated", true)
```

Gestion des erreurs

Le principe de l'actionReport est implémenté à toutes les étapes du BPM LISTER_BASE.

```
tw.local.actionReport = new tw.object.listOf.toolkit.TWSYS.String();
```

Il est enrichi du détail de chaque action réalisée. Lors d'une erreur, cette erreur est incorporée au rapport. Celui-ci est tronqué aux 10 dernières lignes et ces lignes sont sérialisées dans le message texte assigné à l'erreur.

Deux exemples de rapport complet :

REMOVE s00vl9975744

<variable type="String[]">
<item type="String"><![CDATA["LISTER BASE" IS STARTING]]></item>
<item type="String"><![CDATA[existingJson_DBList exists: vmNames = "s00vl9975747"]]></item>
<item type="String"><![CDATA[1/2/2018 13:56:29:723: inputs are OK for: REMOVE s00vl9975744]]></item>
<item type="String"><![CDATA[1/2/2018 13:56:29:733: Deleting "s00vl9975744" from database]]></item>
<item type="String"><![CDATA["s00vl9975744": deleted]]></item>
<item type="String"><![CDATA[1/2/2018 13:56:29:773: existingJson_DBList has been updated]]></item>
<item type="String"><![CDATA[1/2/2018 13:56:29:782: Calculating updated increments]]></item>
<item type="String"><![CDATA[1/2/2018 13:56:29:826: updating the increment referential]]></item>
</variable>

UPDATE *[SSHcredentielOf(s00vl9975747,s00vl9975744,s00vl9975747)]*

<variable type="String[]">
<item type="String"><![CDATA["LISTER BASE" IS STARTING]]></item>
<item type="String"><![CDATA[existingJson_DBList exists: vmNames = "s00vl9975747"]]></item>
<item type="String"><![CDATA[1/2/2018 12:7:3:626: inputs are OK for: UPDATE s00vl9975747,s00vl9975744,s00vl9975747]]></item>
<item type="String"><![CDATA[Checking ORACLE_12102_1.60.1_LINUX_GEN_BE_FR_IT.tar on s00vl9975744:]]></item>
<item type="String"><![CDATA[Starting "wget Tar If Needed"]]></item>
<item type="String"><![CDATA[ssh root@s00vl9975744(passwd=*******1Xg) cmdLine = "if test -f ORACLE_12102_1.60.1_LINUX_GEN_BE_FR_IT.tar ; then true ; else wget http://kickstart-dev:8000/RedHat_7.0AS/x86_64/Src/RefPack_LIN/ORACLE_12102_1.60.1_LINUX_GEN_BE_FR_IT.tar && tar xf ORACLE_12102_1.60.1_LINUX_GEN_BE_FR_IT.tar ; fi"]]></item>
<item type="String"><![CDATA[Starting "List Instances"]]></item>
<item type="String"><![CDATA[ssh root@s00vl9975744(passwd=*******1Xg) cmdLine = "/tmp/install_oracle/ksh/db_instance.ksh list|sed 's/^.*ORACLE_SID=\(.....\).*$/\1/'"]]></item>
<item type="String"><![CDATA[GetVMinstanceList SUCCEED: D12345DP10,QICOBPNP10,QICOBPTP10]]></item>
<item type="String"><![CDATA[1/2/2018_12:7:7: currentInstance D12345DP10 selected to be processed. 2 remaining]]></item>
<item type="String"><![CDATA[Starting "GetVMinstanceList"]]></item>
<item type="String"><![CDATA[Starting "RemoteSSH for D12345DP10 instances details"]]></item>
<item type="String"><![CDATA[ssh root@s00vl9975744(passwd=*******1Xg) cmdLine = "INST=D12345DP10;cd /tmp/install_oracle/ksh;./db_user.ksh -sid $INST -list|awk '$2~/[0-9][0-9]*/ {print "USERS:"$1}'&& ./db_show_parameters.ksh -sid $INST -all|awk '$1~/^[^;][^;]*;.*/{print "SPECS:"$1}'&& ./db_service_name.ksh -sid $INST -list|awk '{print "SERVICES:"$1}'"]]></item>
<item type="String"><![CDATA[1/2/2018_12:7:11 -> stdOut returned a total of 438 elements]]></item>
<item type="String"><![CDATA[FOUND on D12345DP10: 19 users + 413 specs + 5 services]]></item>
<item type="String"><![CDATA[1/2/2018_12:7:11: currentInstance QICOBPNP10 selected to be processed. 1 remaining]]></item>
<item type="String"><![CDATA[Starting "GetVMinstanceList"]]></item>
<item type="String"><![CDATA[Starting "RemoteSSH for QICOBPNP10 instances details"]]></item>
<item type="String"><![CDATA[ssh root@s00vl9975744(passwd=*******1Xg) cmdLine = "INST=QICOBPNP10;cd /tmp/install_oracle/ksh;./db_user.ksh -sid $INST -list|awk '$2~/[0-9][0-9]*/ {print "USERS:"$1}'&& ./db_show_parameters.ksh -sid $INST -all|awk '$1~/^[^;][^;]*;.*/{print "SPECS:"$1}'&& ./db_service_name.ksh -sid $INST -list|awk '{print "SERVICES:"$1}'"]]></item>
<item type="String"><![CDATA[1/2/2018_12:7:15 -> stdOut returned a total of 441 elements]]></item>
<item type="String"><![CDATA[FOUND on QICOBPNP10: 19 users + 413 specs + 8 services]]></item>
<item type="String"><![CDATA[1/2/2018_12:7:15: currentInstance QICOBPTP10 selected to be processed. 0 remaining]]></item>
<item type="String"><![CDATA[Starting "GetVMinstanceList"]]></item>
<item type="String"><![CDATA[Starting "RemoteSSH for QICOBPTP10 instances details"]]></item>
<item type="String"><![CDATA[ssh root@s00vl9975744(passwd=*******1Xg) cmdLine = "INST=QICOBPTP10;cd /tmp/install_oracle/ksh;./db_user.ksh -sid $INST -list|awk '$2~/[0-9][0-9]*/ {print "USERS:"$1}'&& ./db_show_parameters.ksh -sid $INST -all|awk '$1~/^[^;][^;]*;.*/{print "SPECS:"$1}'&& ./db_service_name.ksh -sid $INST -list|awk '{print "SERVICES:"$1}'"]]></item>
<item type="String"><![CDATA[1/2/2018_12:7:19 -> stdOut returned a total of 441 elements]]></item>
<item type="String"><![CDATA[FOUND on QICOBPTP10: 19 users + 413 specs + 8 services]]></item>
<item type="String"><![CDATA[1/2/2018_12:7:19: QICOBPTP10 has been Processed. Switch to next VM (s00vl9975747)]]></item>
<item type="String"><![CDATA[Checking ORACLE_12102_1.60.1_LINUX_GEN_BE_FR_IT.tar on s00vl9975747:]]></item>
<item type="String"><![CDATA[Starting "wget Tar If Needed"]]></item>
<item type="String"><![CDATA[ssh root@s00vl9975747(passwd=*******105!) cmdLine = "if test -f ORACLE_12102_1.60.1_LINUX_GEN_BE_FR_IT.tar ; then true ; else wget http://kickstart-dev:8000/RedHat_7.0AS/x86_64/Src/RefPack_LIN/ORACLE_12102_1.60.1_LINUX_GEN_BE_FR_IT.tar && tar xf ORACLE_12102_1.60.1_LINUX_GEN_BE_FR_IT.tar ; fi"]]></item>
<item type="String"><![CDATA[Starting "List Instances"]]></item>
<item type="String"><![CDATA[ssh root@s00vl9975747(passwd=*******105!) cmdLine = "/tmp/install_oracle/ksh/db_instance.ksh list|sed 's/^.*ORACLE_SID=\(.....\).*$/\1/'"]]></item>
<item type="String"><![CDATA[GetVMinstanceList SUCCEED: QICOBPNS10,QICOBPTS10]]></item>

<item type="String"><![CDATA[1/2/2018_12:7:23: currentInstance QICOBPNS10 selected to be processed. 1 remaining]]></item>
<item type="String"><![CDATA[Starting "GetVMinstanceList"]]></item>
<item type="String"><![CDATA[Starting "RemoteSSH for QICOBPNS10 instances details"]]></item>
<item type="String"><![CDATA[ssh root@s00vl9975747(passwd=*******105!) cmdLine = "INST=QICOBPNS10;cd /tmp/install_oracle/ksh; ./db_user.ksh -sid $INST -list|awk '$2~/[0-9][0-9]*/ {print "USERS:"$1}'&& ./db_show_parameters.ksh -sid $INST -all|awk '$1~/^[^;][^;]*;.*/{print "SPECS:"$1}'&& ./db_service_name.ksh -sid $INST -list|awk '{print "SERVICES:"$1}'"]]></item>
<item type="String"><![CDATA[1/2/2018_12:7:26 -> stdOut returned a total of 419 elements]]></item>
<item type="String"><![CDATA[FOUND on QICOBPNS10: 0 users + 413 specs + 5 services]]></item>
<item type="String"><![CDATA[1/2/2018_12:7:27: currentInstance QICOBPTS10 selected to be processed. 0 remaining]]></item>
<item type="String"><![CDATA[Starting "GetVMinstanceList"]]></item>
<item type="String"><![CDATA[Starting "RemoteSSH for QICOBPTS10 instances details"]]></item>
<item type="String"><![CDATA[ssh root@s00vl9975747(passwd=*******105!) cmdLine = "INST=QICOBPTS10;cd /tmp/install_oracle/ksh; ./db_user.ksh -sid $INST -list|awk '$2~/[0-9][0-9]*/ {print "USERS:"$1}'&& ./db_show_parameters.ksh -sid $INST -all|awk '$1~/^[^;][^;]*;.*/{print "SPECS:"$1}'&& ./db_service_name.ksh -sid $INST -list|awk '{print "SERVICES:"$1}'"]]></item>
<item type="String"><![CDATA[1/2/2018_12:7:30 -> stdOut returned a total of 419 elements]]></item>
<item type="String"><![CDATA[FOUND on QICOBPTS10: 0 users + 413 specs + 5 services]]></item>
<item type="String"><![CDATA[1/2/2018_12:7:30: QICOBPTS10 has been Processed. No more instance. Switch to "CALCUL DIFF"]]></item>
<item type="String"><![CDATA[1/2/2018 12:7:30:838: ########## CALCULATING DIFF ##########]]></item>
<item type="String"><![CDATA[currentScan is 2153 long]]></item>
<item type="String"><![CDATA[PreviousScan is 836 long]]></item>
<item type="String"><![CDATA[DIIF: toAdd = 1317 elts ; toDel = 0 elts]]></item>
<item type="String"><![CDATA[1/2/2018 12:7:31:299: ############ END OF CALCUL ############]]></item>
<item type="String"><![CDATA[1/2/2018 12:7:31:339: Updating the previous scan with the current one: SUCCEED]]></item>
<item type="String"><![CDATA[1/2/2018 12:7:31:349: Calculating updated increments]]></item>
<item type="String"><![CDATA[1/2/2018 12:7:31:379: updating the increment referential]]></item>
</variable>