Développez un plug-in pour votre application

SnapCenter Software 4.9

NetApp
March 20, 2024

Sommaire

Développez un plug-in pour votre application

Présentation

SnapCenter Server vous permet de déployer et de gérer vos applications en tant que plug-ins pour SnapCenter. Les applications de votre choix peuvent être connectées au serveur SnapCenter pour la protection des données et fonctionnalités de gestion.

SnapCenter vous permet de développer des plug-ins personnalisés à l'aide de différents langages de programmation. C'est possible Développez un plug-in personnalisé à l'aide de Perl, Java, BATCH ou d'autres langages de script.

Pour utiliser des plug-ins personnalisés dans SnapCenter, vous devez effectuer les tâches suivantes :

- Créez un plug-in pour votre application en suivant les instructions de ce guide
- Créez un fichier de description
- Exportez le plug-in personnalisé pour l'installer sur l'hôte SnapCenter
- Chargez le fichier zip du plug-in dans SnapCenter Server

Gestion générique du plug-in dans tous les appels API

Pour chaque appel d'API, utilisez les informations suivantes :

- Paramètres du plug-in
- Codes de sortie
- Consigner les messages d'erreur
- La cohérence des données

Utiliser les paramètres du plug-in

Un ensemble de paramètres est transmis au plug-in dans le cadre de chaque appel d'API effectué. Le tableau suivant répertorie les informations spécifiques pour les paramètres.

Paramètre	Objectif
ACTION	Détermine le nom du flux de travail. Par exemple, découverte, sauvegarde, fileOrVolRestore ou CloneVolAndLun
RESSOURCES	Répertorie les ressources à protéger. Une ressource est identifiée par un UID et un Type. La liste est présentée au plug-in au format suivant : "<UID>,<TYPE>;<UID>,<TYPE>". Par exemple : « Instance1,instance;Instance2\\DB1,Database »

Paramètre	Objectif
NOM_APPLICATION	Détermine le plug-in utilisé. Par exemple, DB2 et MYSQL. Le serveur SnapCenter prend en charge les applications répertoriées. Ce paramètre est sensible à la casse.
APP_IGNORE_ERROR	(O ou N) cela entraîne la fermeture ou l'arrêt de SnapCenter en cas d'erreur d'application. Ceci est utile lorsque vous sauvegardez plusieurs bases de données et que vous ne voulez pas qu'une seule panne arrêtez l'opération de sauvegarde.
<RESOURCE_NAME>__APP_INSTANCE_USERNAME	Les informations d'identification SnapCenter sont définies pour la ressource.
<NOM_RESSOURCE>_APP_INSTANCE_PASSWORD	Les informations d'identification SnapCenter sont définies pour la ressource.
<NOM_RESSOURCE>_<PARAM_PERSONNALISÉE>	Chaque valeur de clé personnalisée de niveau de ressource est disponible pour les plug-ins préfixés avec « <RESOURCE_NAME>_ ». Par exemple, si un La clé personnalisée est "MASTER_SLAVE" pour une ressource Nommé "MySQLDB", il sera disponible en tant que MySQLDB_MASTER_SLAVE

Utiliser les codes de sortie

Le plug-in renvoie l'état de l'opération à l'hôte au moyen de codes de sortie. Chacun le code a une signification spécifique et le plug-in utilise le code de sortie approprié pour indiquer la même chose.

Le tableau suivant décrit les codes d'erreur et leur signification.

Code de sortie	Objectif
0	Opération réussie.
99	L'opération demandée n'est pas prise en charge ou mise en œuvre.
100	Échec de l'opération, ignorer la mise en veille et quitter. La mise en attente est par défaut.
101	Échec de l'opération, poursuivre l'opération de sauvegarde.
autre	Échec de l'opération, exécution sans mise en veille et fermeture.

Consigner les messages d'erreur

Les messages d'erreur sont transmis du plug-in au serveur SnapCenter. Le message inclut le message, le niveau de journal et l'horodatage.

Le tableau suivant répertorie les niveaux et leurs objectifs.

Paramètre	Objectif
INFO	message informatif
AVERTISSEMENT	message d'avertissement
ERREUR	message d'erreur
DÉBOGAGE	message de débogage
TRACÉ	message de trace

Préservez la cohérence des données

Les plug-ins personnalisés préservent les données entre les opérations liées à la même exécution du flux de travail. Pour par exemple, un plug-in peut stocker des données à la fin de la mise en veille, qui peuvent être utilisées pendant la mise en veille fonctionnement.

Les données à préserver sont définies dans le cadre de l'objet résultat par le plug-in. Il suit un format spécifique et est décrit en détail sous chaque style de développement de plug-in.

Développement BASÉ SUR PERL

Vous devez respecter certaines conventions lors du développement du plug-in à l'aide DE PERL.

- Le contenu doit être lisible
- Doit mettre en œuvre les opérations obligatoires setenv, quiesce et unquiesce
- Doit utiliser une syntaxe spécifique pour renvoyer les résultats à l'agent
- Le contenu doit être enregistré sous la forme d'un fichier <PLUGIN_NAME>.pm

Les opérations disponibles sont

- Valeur de consigne
- version
- mise au repos
- mise au repos
- clone_pre, clone_post
- restore_pre, restaurez
- nettoyage

Manipulation générale du plug-in

Utilisation de l'objet de résultats

Chaque opération de plug-in personnalisée doit définir l'objet de résultats. Cet objet envoie des messages, un code de sortie, un stdout et un stderr à l'agent hôte.

Objet Résultats :

```
my $result = {
```

```
      exit_code => 0,
      stdout => "",
      stderr => "",
};
```

Retour de l'objet de résultats :

```
return $result;
```

Préservation de la cohérence des données

Il est possible de conserver les données entre les opérations (à l'exception du nettoyage) dans le cadre de la même exécution du flux de travail. Pour ce faire, il faut des paires clé-valeur. Les paires de données clé-valeur sont définies comme faisant partie de l'objet résultat et sont conservées et disponibles dans les opérations suivantes du même flux de travail.

L'exemple de code suivant définit les données à conserver :

```
my $result = {
  exit_code => 0,
  stdout => "",
  stderr => "",
};
  $result->{env}->{'key1'} = 'value1';
  $result->{env}->{'key2'} = 'value2';
  ….
  return $result
```

Le code ci-dessus définit deux paires clé-valeur, qui sont disponibles comme entrée dans l'opération suivante. Les deux paires clé-valeur sont accessibles à l'aide du code suivant :

```
sub setENV {
    my ($self, $config) = @_;
    my $first_value = $config->{'key1'};
    my $second_value = $config->{'key2'};
    …
}
```

```
 === Logging error messages
Chaque opération peut renvoyer des messages à l'agent hôte, qui affiche et stocke le contenu. Un message contient le niveau du message, un horodatage et un texte de message. Les messages multilignes sont pris en charge.
```

```
Load the SnapCreator::Event Class:
my $msgObj = new SnapCreator::Event();
my @message_a = ();
```

Utilisez msgObj pour capturer un message à l'aide de la méthode de collecte.

```
$msgObj->collect(\@message_a, INFO, "My INFO Message");
$msgObj->collect(\@message_a, WARN, "My WARN Message");
$msgObj->collect(\@message_a, ERROR, "My ERROR Message");
$msgObj->collect(\@message_a, DEBUG, "My DEBUG Message");
$msgObj->collect(\@message_a, TRACE, "My TRACE Message");
```

Appliquer les messages à l'objet de résultats :

```
$result->{message} = \@message_a;
```

Utilisation de stubs enfichables

Les plug-ins personnalisés doivent exposer les stubs plug-in. Ce sont des méthodes que le serveur SnapCenter appelle, en fonction d'un flux de travail.

Embout enfichable	Facultatif/requis	Objectif
Valeur de consigne	obligatoire	Ce stub définit l'environnement et l'objet de configuration. L'analyse ou la gestion de l'environnement doit être effectuée ici. Chaque fois qu'un stub est appelé, le stub setenv est appelé juste avant. Elle n'est requise que pour LES plug-ins DE type PERL.
Version	Facultatif	Ce segment d'accroche est utilisé pour obtenir la version de l'application.
Découverte	Facultatif	Ce stub permet de découvrir des objets d'application tels que l'instance ou la base de données hébergées sur l'agent ou l'hôte. Le plug-in devrait renvoyer les objets d'application découverts dans un format spécifique dans le cadre de la réponse. Ce stub n'est utilisé que si l'application est intégrée à SnapDrive pour Unix. Le système de fichiers Linux (Linux Flavors) est pris en charge. AIX/Solaris (versions Unix) ne sont pas pris en charge.

Embout enfichable	Facultatif/requis	Objectif
discovery_complete	Facultatif	Ce stub permet de découvrir des objets d'application tels que l'instance ou la base de données hébergées sur l'agent ou l'hôte. Le plug-in devrait renvoyer les objets d'application découverts dans un format spécifique dans le cadre de la réponse. Ce stub n'est utilisé que si l'application est intégrée à SnapDrive pour Unix. Le système de fichiers Linux (Linux Flavors) est pris en charge. AIX et Solaris (versions Unix) ne sont pas pris en charge.
Mise au repos	obligatoire	Ce stub est chargé d'effectuer une mise au repos, ce qui signifie que l'application est mise à l'état dans lequel vous pouvez créer une copie Snapshot. Il s'agit de l'opération avant l'opération de copie Snapshot. Les métadonnées de l'application à conserver doivent être définies dans le cadre de la réponse, qui doivent être renvoyées au cours des opérations de clonage suivantes ou de restauration sur la copie Snapshot de stockage correspondante sous la forme de paramètres de configuration.
Mise au repos	obligatoire	Ce segment d'accroche est responsable de l'exécution d'une mise en veille, ce qui signifie que l'application est dans un état normal. Ce processus est appelé après la création d'une copie Snapshot.

Embout enfichable	Facultatif/requis	Objectif
clone_pre	facultatif	Ce stub est responsable de l'exécution des tâches de préclonage. Cela suppose que vous utilisez l'interface intégrée de clonage de SnapCenter Server et qu'elle est déclenchée lors de l'opération de clonage.
clone_post	facultatif	Ce stub est responsable de l'exécution des tâches post-clone. Cela suppose que vous utilisez l'interface intégrée de clonage de SnapCenter Server et que vous êtes déclenché uniquement lors de l'opération de clonage.
restore_pre	facultatif	Ce stub est responsable de l'exécution des tâches de préstockage. Cela suppose que vous utilisez l'interface intégrée de restauration du serveur SnapCenter et qu'elle est déclenchée lors de l'opération de restauration.
Restaurer	facultatif	Ce stub est responsable de l'exécution des tâches de restauration de l'application. Cela suppose que vous utilisez l'interface intégrée de restauration du serveur SnapCenter et qu'elle n'est déclenchée que lors de l'opération de restauration.

Embout enfichable	Facultatif/requis	Objectif
Nettoyage	facultatif	Ce stub est chargé d'effectuer le nettoyage après les opérations de sauvegarde, de restauration ou de clonage. Le nettoyage peut se faire lors de l'exécution normale du workflow ou en cas d'échec du flux de travail. Vous pouvez déduire le nom du flux de travail sous lequel le nettoyage est appelé en faisant référence à L'ACTION de paramètre de configuration, qui peut être backup, cloneVolAndLun ou fileOrVolRestore. Le paramètre de configuration ERROR_MESSAGE indique s'il y a eu une erreur lors de l'exécution du flux de travail. Si ERROR_MESSAGE est défini et NON NULL, alors le nettoyage est appelé pendant l'exécution de l'échec du workflow.
version_app	Facultatif	Ce stub est utilisé par SnapCenter pour obtenir l'application détails de version gérés par le plug-in.

Informations sur le module enfichable

Chaque plug-in doit disposer des informations suivantes :

```
package MOCK;
our @ISA = qw(SnapCreator::Mod);
=head1 NAME
MOCK - class which represents a MOCK module.
=cut
=head1 DESCRIPTION
MOCK implements methods which only log requests.
=cut
use strict;
use warnings;
use diagnostics;
use SnapCreator::Util::Generic qw ( trim isEmpty );
use SnapCreator::Util::OS qw ( isWindows isUnix getUid
createTmpFile );
use SnapCreator::Event qw ( INFO ERROR WARN DEBUG COMMENT ASUP
CMD DUMP );
my $msgObj = new SnapCreator::Event();
my %config_h = ();
```

Exploitation

Vous pouvez encoder diverses opérations telles que setenv, version, Quiesce et unquiesce, qui sont prises en charge par les plug-ins personnalisés.

Opération setenv

L'opération setenv est requise pour les plug-ins créés à l'aide DE PERL. Vous pouvez régler l'ENV et accéder facilement aux paramètres du plug-in.

```
sub setENV {
    my ($self, $obj) = @_;
    %config_h = %{$obj};
    my $result = {
      exit_code => 0,
      stdout => "",
      stderr => "",
    };
    return $result;
}
```

Exploitation de version

L'opération de version renvoie les informations de version de l'application.

```
sub version {
  my $version_result = {
    major => 1,
    minor => 2,
    patch => 1,
    build => 0
  };
  my @message_a = ();
  $msgObj->collect(\@message_a, INFO, "VOLUMES
$config_h{'VOLUMES'}");
  $msgObj->collect(\@message_a, INFO,
"$config_h{'APP_NAME'}::quiesce");
  $version_result->{message} = \@message_a;
  return $version_result;
}
```

Opérations de mise en veille

L'opération de mise en veille effectue une opération de mise en veille de l'application sur les ressources répertoriées dans le paramètre RESSOURCES.

```
sub quiesce {
  my $result = {
      exit_code => 0,
      stdout => "",
      stderr => "",
  };
  my @message_a = ();
  $msgObj->collect(\@message_a, INFO, "VOLUMES
$config_h{'VOLUMES'}");
  $msgObj->collect(\@message_a, INFO,
"$config_h{'APP_NAME'}::quiesce");
  $result->{message} = \@message_a;
  return $result;
}
```

Opération de mise en veille

Une opération de mise en attente est requise pour arrêter l'application. La liste des ressources est disponible dans le paramètre RESSOURCES.

```
sub unquiesce {
  my $result = {
      exit_code => 0,
      stdout => "",
      stderr => "",
  };
  my @message_a = ();
  $msgObj->collect(\@message_a, INFO, "VOLUMES
$config_h{'VOLUMES'}");
  $msgObj->collect(\@message_a, INFO,
"$config_h{'APP_NAME'}::unquiesce");
  $result->{message} = \@message_a;
  return $result;
}
```

Style NATIF

SnapCenter prend en charge les langages de programmation ou de script non PERL pour créer des plug-ins. Il s'agit d'une programmation DE type NATIF, qui peut être un fichier script ou BATCH.

Les plug-ins DE style NATIF doivent respecter certaines conventions ci-dessous :

Le plug-in doit être exécutable

- Pour les systèmes Unix, l'utilisateur qui exécute l'agent doit disposer de privilèges d'exécution sur le plug-in
- Pour les systèmes Windows, les plug-ins PowerShell doivent avoir le suffixe .ps1, autres fenêtres les scripts doivent avoir le suffixe .cmd ou .bat et doivent être exécutables par l'utilisateur
- Les plug-ins doivent réagir à l'argument de ligne de commande comme «-quiesce », «-unquiesce »
- Les plug-ins doivent renvoyer le code de sortie 99 si une opération ou une fonction n'est pas implémentée
- Les plug-ins doivent utiliser une syntaxe spécifique pour renvoyer les résultats au serveur

Manipulation générale du plug-in

Journalisation des messages d'erreur

Chaque opération peut renvoyer des messages au serveur, qui affiche et stocke le contenu. Un message contient le niveau du message, un horodatage et un texte de message. Les messages multilignes sont pris en charge.

Format :

```
SC_MSG#<level>#<timestamp>#<message>
SC_MESSAGE#<level>#<timestamp>#<message>
```

Utilisation de stubs enfichables

Les plug-ins SnapCenter doivent être en place avec des stubs plug-in. Ce sont des méthodes que le serveur SnapCenter appelle en fonction d'un flux de travail spécifique.

Embout enfichable	Facultatif/requis	Objectif
mise au repos	obligatoire	Ce segment d'accroche est responsable de l'exécution d'une mise en veille. Il place le État de l'application dans lequel nous pouvons créer une copie Snapshot. Il s'agit de l'opération de copie Snapshot du stockage.
mise au repos	obligatoire	Ce segment d'accroche est responsable de l'exécution d'une mise au repos. Il place l'application dans un état normal. Ce processus est appelé après le stockage Opération de copie Snapshot.
clone_pre	facultatif	Ce stub est responsable de l'exécution des tâches de pré-clonage. Cela suppose que vous utilisez l'interface intégrée de clonage SnapCenter et que vous êtes également déclenché uniquement lors de l'exécution de l'action clone_vol ou clone_lun.
clone_post	Facultatif	Ce stub est responsable de l'exécution des tâches post-clone. Cela suppose que vous utilisez l'interface intégrée de clonage SnapCenter et que vous êtes uniquement déclenché lors des opérations clone_vol ou clone_lun.
restore_pre	Facultatif	Ce stub est responsable de l'exécution des tâches de pré-restauration. Cela suppose que vous utilisez l'interface intégrée de restauration SnapCenter et qu'elle n'est déclenchée que lorsque vous effectuez l'opération de restauration.

Embout enfichable	Facultatif/requis	Objectif
restaurer	facultatif	Ce stub est chargé d'effectuer toutes les actions de restauration. C'est ça suppose que vous n'utilisez pas l'interface de restauration intégrée. Il est déclenché lors de l'opération de restauration.

Exemples

Windows PowerShell

Vérifiez si le script peut être exécuté sur votre système. Si vous ne pouvez pas exécuter le script, définissez le contournement Set-ExecutionPolicy pour le script et relancez l'opération.

```
if ($args.length -ne 1) {
  write-warning "You must specify a method";
  break;
}
function log ($level, $message) {
  $d = get-date
  echo "SC_MSG#$level#$d#$message"
}
function quiesce {
  $app_name = (get-item env:APP_NAME).value
  log "INFO" "Quiescing application using script $app_name";
  log "INFO" "Quiescing application finished successfully"
}
function unquiesce {
  $app_name = (get-item env:APP_NAME).value
  log "INFO" "Unquiescing application using script $app_name";
  log "INFO" "Unquiescing application finished successfully"
}
  switch ($args[0]) {
    "-quiesce" {
    quiesce;
}
"-unquiesce" {
    unquiesce;
}
default {
    write-error "Function $args[0] is not implemented";
    exit 99;
  }
}
exit 0;
```

Style Java

Un plug-in personnalisé Java interagit directement avec une application telle qu'une base de données, une instance, etc.

Limites

Certaines limitations doivent être prises en compte lors du développement d'un plug-in à l'aide du langage de programmation Java.

Caractéristique du plug-in	Plug-in Java
La complexité	Faible à moyen

Caractéristique du plug-in	Plug-in Java
Encombrement de la mémoire	Jusqu'à 10-20 Mo
Dépendances sur d'autres bibliothèques	Bibliothèques pour la communication d'applications
Nombre de threads	1
Exécution de filetage	Moins d'une heure

Raison des limitations de Java

L'objectif de l'agent SnapCenter est de garantir une intégration continue, sûre et fiable des applications. En prenant en charge les plug-ins Java, il est possible que les plug-ins introduisent des fuites de mémoire et d'autres problèmes indésirables. Ces problèmes sont difficiles à résoudre, surtout lorsque l'objectif est de simplifier l'utilisation des choses. Si la complexité d'un plug-in n'est pas trop complexe, il est beaucoup moins probable que les développeurs aient introduit les erreurs. Le danger du plug-in Java est qu'ils le sont S'exécutant dans la même JVM que l'agent SnapCenter lui-même. Lorsque le plug-in tombe en panne ou fuit de la mémoire, il peut également avoir un impact négatif sur l'agent.

Méthodes prises en charge

Méthode	Obligatoire	Description	Appelé quand et par qui?
Version	Oui.	Doit renvoyer la version du plug-in.	Par le serveur ou l'agent SnapCenter pour demander la version de le plug-in.
Mise au repos	Oui.	Doit effectuer une mise en veille sur l'application. Dans la plupart des cas, cela signifie que l'application est mise à l'état où le serveur SnapCenter peut créer une sauvegarde (par exemple, une copie Snapshot).	Avant que le serveur SnapCenter ne crée une copie Snapshot ou effectue une sauvegarde en général.
Mise au repos	Oui.	Doit effectuer une mise en veille sur l'application. Dans la plupart des cas, ceci signifie remettre l'application dans un état de fonctionnement normal.	Une fois que le serveur SnapCenter a créé une copie Snapshot ou le a créé a effectué une sauvegarde en général.

Méthode	Obligatoire	Description	Appelé quand et par qui?
Nettoyage	Non	Responsable du nettoyage de tout ce que le plug-in doit nettoyer.	Lorsqu'un flux de travail sur le serveur SnapCenter se termine (avec succès ou avec échec).
ClonePre	Non	Actions à effectuer avant l'exécution d'une opération de clonage	Lorsqu'un utilisateur déclenche une action « cloneVol » ou « cloneLun » et utilise l'assistant de clonage intégré (GUI/CLI).
Montant en clonePost	Non	Doit effectuer les actions qui doivent avoir lieu après l'exécution d'une opération de clonage.	Lorsqu'un utilisateur déclenche une action « cloneVol » ou « cloneLun » et utilise l'assistant de clonage intégré (GUI/CLI).
RestauePre	Non	Doit effectuer des actions qui doivent se produire avant l'appel de l'opération de restauration.	Lorsqu'un utilisateur déclenche une opération de restauration.
Restaurer	Non	Responsable de la restauration/restauration de l'application.	Lorsqu'un utilisateur déclenche une opération de restauration.
AppVersion	Non	Pour récupérer la version de l'application gérée par le plug-in.	Dans le cadre de la collecte de données ASUP, dans chaque workflow tel que sauvegarde/restauration/clonage.

Didacticiel

Cette section décrit comment créer un plug-in personnalisé à l'aide du langage de programmation Java.

Configuration d'eclipse

1. Créez un nouveau projet Java « TutorialPlugin » dans Eclipse
2. Cliquez sur **Terminer**
3. Cliquez avec le bouton droit de la souris sur **nouveau projet → Propriétés → chemin de construction Java → bibliothèques → Ajouter des fichiers JAR externes**
4. Accédez au dossier ../lib/ de Host Agent et sélectionnez JAR scAgent-5.0-core.jar et common-5.0.jar
5. Sélectionnez le projet et cliquez avec le bouton droit de la souris sur le dossier **src → Nouveau →**

Package et créez un nouveau paquet portant le nom com.netapp.snapcreator.agent.plugin.TutorialPlugin

6. Cliquez avec le bouton droit de la souris sur le nouveau package et sélectionnez Nouveau → classe Java.
   a. Entrez le nom en tant que TutorialPlugin.
   b. Cliquez sur le bouton de navigation de superclasse et recherchez "*AbstractPlugin". Un seul résultat doit apparaître :

```
 "AbstractPlugin - com.netapp.snapcreator.agent.nextgen.plugin".
.. Cliquez sur *Terminer*.
.. Classe Java :
```

```
package com.netapp.snapcreator.agent.plugin.TutorialPlugin;
import
com.netapp.snapcreator.agent.nextgen.common.result.Describe
Result;
import
com.netapp.snapcreator.agent.nextgen.common.result.Result;
import
com.netapp.snapcreator.agent.nextgen.common.result.VersionR
esult;
import
com.netapp.snapcreator.agent.nextgen.context.Context;
import
com.netapp.snapcreator.agent.nextgen.plugin.AbstractPlugin;
public class TutorialPlugin extends AbstractPlugin {
  @Override
  public DescribeResult describe(Context context) {
    // TODO Auto-generated method stub
    return null;
  }
  @Override
  public Result quiesce(Context context) {
    // TODO Auto-generated method stub
    return null;
  }
  @Override
  public Result unquiesce(Context context) {
    // TODO Auto-generated method stub
    return null;
  }
  @Override
  public VersionResult version() {
    // TODO Auto-generated method stub
    return null;
  }
}
```

Mise en œuvre des méthodes requises

Les méthodes de mise au repos, de mise au repos et de version sont obligatoires que chaque plug-in Java personnalisé doit implémenter.

La méthode de version suivante permet de renvoyer la version du plug-in.

```
@Override
public VersionResult version() {
    VersionResult versionResult = VersionResult.builder()
                                                .withMajor(1)
                                                .withMinor(0)
                                                .withPatch(0)
                                                .withBuild(0)
                                                .build();
    return versionResult;
}
```

```
Below is the implementation of quiesce and unquiesce method. These will be
interacting with   the application, which is being protected by SnapCenter
Server. As this is just a tutorial, the
application part is not explained, and the focus is more on the
functionality that SnapCenter   Agent provides the following to the plug-
in developers:
```

```
@Override
  public Result quiesce(Context context) {
    final Logger logger = context.getLogger();
    /*
      * TODO: Add application interaction here
    */
```

```
logger.error("Something bad happened.");
logger.info("Successfully handled application");
```

```
    Result result = Result.builder()
                    .withExitCode(0)
                    .withMessages(logger.getMessages())
                    .build();
    return result;
}
```

La méthode est passée dans un objet de contexte. Il contient plusieurs aides, par exemple un Logger et un Context Store, ainsi que des informations sur l'opération en cours (ID-workflow, ID-travail). Nous pouvons obtenir l'enregistreur en appelant l'enregistreur final Logger Logger = Context.GetLogger();. L'objet logger fournit des méthodes similaires connues d'autres frameworks de consignation, par exemple, la déconnexion. Dans l'objet résultat, vous pouvez également spécifier le code de sortie. Dans cet exemple, zéro est renvoyé, car il n'y a pas eu de problème. D'autres codes de sortie peuvent correspondre à différents scénarios de

défaillance.

Utilisation de l'objet résultat

L'objet résultat contient les paramètres suivants :

Paramètre	Valeur par défaut	Description
Gstn de la	Vide gstn de la	Ce paramètre peut être utilisé pour renvoyer les paramètres de configuration au serveur. Il peut être des paramètres que le plug-in souhaite mettre à jour. Si ce changement est La configuration du serveur SnapCenter dépend de celle-ci Le paramètre APP_CONF_PERSISTENT=y ou N dans la configuration.
Code exitcode	0	Indique l'état de l'opération. Un « 0 » signifie que l'opération a été exécution réussie. D'autres valeurs indiquent des erreurs ou des avertissements.
Stdout	Vide Liste	Il peut être utilisé pour transmettre des messages stdout à l'SnapCenter Serveur.
Stderr	Vide Liste	Il peut être utilisé pour transmettre des messages stderr à SnapCenter Serveur.
Messages	Vide Liste	Cette liste contient tous les messages qu'un plug-in souhaite retourner à l' serveur. Le serveur SnapCenter affiche ces messages dans l'interface de ligne de commande ou l'interface utilisateur graphique.

L'agent SnapCenter fournit les constructeurs ("Motif de création") pour tous ses types de résultats. L'utilisation est donc très simple :

```
Result result = Result.builder()
                    .withExitCode(0)
                    .withStdout(stdout)
                    .withStderr(stderr)
                    .withConfig(config)
                    .withMessages(logger.getMessages())
                    .build()
```

Par exemple, définissez le code de sortie sur 0, définissez des listes pour stdout et stderr, définissez les paramètres de configuration et ajoutez également les messages de journal qui seront renvoyés au serveur. Si vous n'avez pas besoin de tous les paramètres, envoyez uniquement ceux qui sont nécessaires. Comme chaque paramètre a une valeur par défaut, si vous supprimez .avecExitCode(0) du code ci-dessous, le résultat n'est pas affecté :

```
Result result = Result.builder()
                      .withExitCode(0)
                      .withMessages(logger.getMessages())
                      .build();
```

Version

Le versionResult informe le serveur SnapCenter de la version du plug-in. Car il hérite également Il contient les paramètres config, exitCode, stdout, stderr et messages.

Paramètre	Valeur par défaut	Description
Majeur	0	Champ de version majeure du plug-in.
Mineur	0	Champ de version mineure du plug-in.
Correctif	0	Champ de version du correctif du plug-in.
Création	0	Champ version de build du plug-in.

Par exemple :

```
VersionResult result = VersionResult.builder()
                                    .withMajor(1)
                                    .withMinor(0)
                                    .withPatch(0)
                                    .withBuild(0)
                                    .build();
```

Utilisation de l'objet de contexte

L'objet de contexte offre les méthodes suivantes :

Méthode de contexte	Objectif
Chaîne GetWorkflowId();	Renvoie l'ID de flux de travail utilisé par le serveur SnapCenter pour le flux de travail actuel.
Config getconfig() ;	Renvoie la configuration en cours d'envoi du serveur SnapCenter vers le Agent.

ID-workflow

L'ID de flux de travail est l'ID que le serveur SnapCenter utilise pour faire référence à une exécution spécifique flux de travail.

Gstn de la

Cet objet contient (la plupart) des paramètres qu'un utilisateur peut définir dans la configuration du Serveur SnapCenter. Cependant, pour des raisons de sécurité, certains de ces paramètres peuvent obtenir filtré côté serveur. Voici un exemple d'accès à la configuration et à la récupération un paramètre :

```
final Config config = context.getConfig();
String myParameter =
config.getParameter("PLUGIN_MANDATORY_PARAMETER");
```

""// myParameter" contient maintenant le paramètre lu dans la configuration sur le serveur SnapCenter Si une clé de paramètre config n'existe pas, elle renvoie une chaîne vide (""").

Exportation du plug-in

Vous devez exporter le plug-in pour l'installer sur l'hôte SnapCenter.

Dans Eclipse, effectuez les tâches suivantes :

1. Cliquez avec le bouton droit de la souris sur le package de base du plug-in (dans notre exemple) com.netapp.snapcreator.agent.plugin.TutorialPlugin).
2. Sélectionnez **Exporter → Java → fichier jar**
3. Cliquez sur **Suivant**.

4. Dans la fenêtre suivante, spécifiez le chemin du fichier JAR de destination : tutorial_plugin.jar La classe de base du plug-in est appelée TutorialPlugin.class, le plug-in doit être ajouté à un dossier avec le même nom.

Si votre plug-in dépend de bibliothèques supplémentaires, vous pouvez créer le dossier suivant : lib/

Vous pouvez ajouter des fichiers JAR dont le plug-in dépend (par exemple, un pilote de base de données). Quand SnapCenter charge le plug-in, il associe automatiquement tous les fichiers jar de ce dossier à celui-ci et les ajoute au chemin de classe.

Plug-in personnalisé dans SnapCenter

Plug-in personnalisé dans SnapCenter

Le plug-in personnalisé créé à l'aide de Java, PERL ou NATIF peut être installé sur l'hôte à l'aide du serveur SnapCenter pour permettre la protection des données de votre application. Vous devez avoir exporté le plug-in pour l'installer sur l'hôte SnapCenter à l'aide de la procédure décrite dans ce tutoriel.

Création d'un fichier de description de plug-in

Pour chaque plug-in créé, vous devez avoir un fichier de description. Le fichier de description décrit les détails du plug-in. Le nom du fichier doit être Plugin_Descriptor.xml.

Utilisation des attributs de fichier descripteur du plug-in et de leur signification

Attribut	Description
Nom	Nom du plug-in. Les caractères alphanumériques sont autorisés. Par exemple, DB2, MYSQL et MongoDB Pour les plug-ins créés dans un style NATIF, assurez-vous de ne pas fournir l'extension du fichier. Par exemple, si le nom du plug-in est MongoDB.sh, indiquez le nom MongoDB.
Version	Version du plug-in. Peut inclure des versions majeures et mineures. Par exemple, 1.0, 1.1, 2.0, 2.1
DisplayName	Nom du plug-in à afficher dans le serveur SnapCenter. Si plusieurs versions d'un même plug-in sont écrites, assurez-vous que le nom d'affichage est le même pour toutes les versions.
PluginType	Langue utilisée pour créer le plug-in. Les valeurs prises en charge sont Perl, Java et Native. Le type de plug-in natif inclut les scripts shell Unix/Linux, les scripts Windows, Python ou tout autre langage de script.

Attribut	Description
OSName%	Nom du système d'exploitation hôte sur lequel le plug-in est installé. Les valeurs valides sont Windows et Linux : Il est possible qu'un seul plug-in soit disponible pour le déploiement sur plusieurs types de systèmes d'exploitation, comme LE plug-in DE type PERL.
OSversion	Version du système d'exploitation hôte sur laquelle le plug-in est installé.
ResourceName	Nom du type de ressource pris en charge par le plug-in. Par exemple, base de données, instance, collections.
Parent	Dans le cas contraire, ResourceName dépend de manière hiérarchique d'un autre type de ressource, alors Parent détermine le type de ressource parent. Par exemple, le plug-in DB2, le nom de ResourceName "Database" a une "instance" parent.
RequireFileSystemPlugin	Oui ou non Détermine si l'onglet récupération est affiché dans l'assistant de restauration.
ResourceRequiresAuthentication	Oui ou non Détermine si les ressources, qui sont automatiquement découvertes ou qui n'ont pas été découvertes la détection automatique a besoin d'informations d'identification pour effectuer les opérations de protection des données après détection du stockage.
RequireFileSystemClone	Oui ou non Détermine si le plug-in nécessite l'intégration du plug-in du système de fichiers pour le clone flux de travail.

Voici un exemple du fichier Plugin_descriptor.xml pour le plug-in DB2 personnalisé :

```
<Plugin>
<SMSServer></SMSServer>
<Name>DB2</Name>
<Version>1.0</Version>
<PluginType>Perl</PluginType>
<DisplayName>Custom DB2 Plugin</DisplayName>
<SupportedOS>
<OS>
<OSName>windows</OSName>
<OSVersion>2012</OSVersion>
</OS>
<OS>
<OSName>Linux</OSName>
<OSVersion>7</OSVersion>
</OS>
</SupportedOS>
<ResourceTypes>
<ResourceType>
<ResourceName>Database</ResourceName>
<Parent>Instance</Parent>
</ResourceType>
<ResourceType>
<ResourceName>Instance</ResourceName>
</ResourceType>
</ResourceTypes>
<RequireFileSystemPlugin>no</RequireFileSystemPlugin>
<ResourceRequiresAuthentication>yes</ResourceRequiresAuthentication>
<SupportsApplicationRecovery>yes</SupportsApplicationRecovery>
</Plugin>
```

Création d'un fichier ZIP

Après avoir développé un plug-in et créé un fichier descripteur, vous devez ajouter les fichiers du plug-in et Le fichier Plugin_descriptor.xml dans un dossier et le zip.

Vous devez tenir compte des éléments suivants avant de créer un fichier ZIP :

- Le nom du script doit être identique au nom du plug-in.
- Pour LE plug-in PERL, le dossier ZIP doit contenir un dossier contenant le fichier script et le le fichier descripteur doit se trouver en dehors de ce dossier. Le nom du dossier doit être identique à celui du nom du plug-in.
- Pour les plug-ins autres que LE plug-in PERL, le dossier ZIP doit contenir le descripteur et les fichiers de script.
- La version du système d'exploitation doit être un nombre.

Exemples :

- Plug-in DB2 : ajoutez les fichiers DB2.pm et Plugin_descriptor.xml à DB2.zip.
- Plug-in développé à l'aide de Java : ajoutez des fichiers JAR, des fichiers JAR dépendants et Fichier Plugin_descriptor.xml dans un dossier et zip-le.

Téléchargement du fichier ZIP du plug-in

Vous devez télécharger le fichier ZIP du plug-in sur le serveur SnapCenter pour que le plug-in soit disponible pour déploiement sur l'hôte souhaité.

Vous pouvez télécharger le plug-in à l'aide de l'interface utilisateur ou des applets de commande.

UI:

- Téléchargez le fichier ZIP du plug-in dans l'assistant de flux de travail **Ajouter** ou **Modifier hôte**
- Cliquez sur * "Sélectionner pour télécharger le plug-in personnalisé"*

PowerShell:

- Cmdlet upload-SmPluginPackage

  Par exemple, PS> Upload-SmPluginPackage -AbsolutePath c:\DB2_1.zip

  Pour obtenir des informations détaillées sur les applets de commande PowerShell, utilisez l'aide du cmdlet SnapCenter ou consultez les informations de référence de l'applet de commande.

"Guide de référence de l'applet de commande du logiciel SnapCenter".

Déploiement de plug-ins personnalisés

Le plug-in personnalisé téléchargé est désormais disponible pour le déploiement sur l'hôte souhaité dans le cadre du **Ajouter** et **Modifier le flux de travail hôte**. Vous pouvez télécharger plusieurs versions de plug-ins sur le Serveur SnapCenter et vous pouvez sélectionner la version à déployer sur un hôte spécifique.

Pour plus d'informations sur le téléchargement du plug-in, reportez-vous à la section "Ajoutez des hôtes et installez des modules plug-ins sur des hôtes distants"

Informations sur le copyright

Informations sur les marques commerciales