Desarrolle un complemento para la aplicación

SnapCenter Software 4.7

NetApp
January 18, 2024

Tabla de contenidos

Desarrolle un complemento para la aplicación

Descripción general

El servidor de SnapCenter permite poner en marcha y gestionar sus aplicaciones como complementos en SnapCenter. Las aplicaciones de su elección pueden conectarse al servidor de SnapCenter para disfrutar de funcionalidades de protección y gestión de datos.

SnapCenter le permite desarrollar complementos personalizados utilizando diferentes lenguajes de programación. Puede desarrollar un complemento personalizado utilizando Perl, Java, BATCH u otros lenguajes de scripting.

Para utilizar plugins personalizados en SnapCenter, debe realizar las siguientes tareas:

- Cree un complemento para su aplicación siguiendo las instrucciones de esta guía
- Cree un archivo de descripción
- Exporte el plugin personalizado para instalarlo en el host de SnapCenter
- Cargue el archivo zip del plugin en el servidor de SnapCenter

Gestión de complementos genérica en todas las llamadas API

Para cada llamada a la API, utilice la siguiente información:

- Parámetros del plugin
- códigos de salida
- Registrar mensajes de error
- Coherencia de datos

Utilice los parámetros del plugin

Se pasa un conjunto de parámetros al plug-in como parte de cada llamada API realizada. En la siguiente tabla, se muestra información específica de los parámetros.

Parámetro	Específico
ACCIÓN	Determina el nombre del flujo de trabajo. Por ejemplo, descubra, copia de seguridad, archivoOrVolRestore o cloneVolAndLun
RECURSOS	Enumera los recursos que se deben proteger. UID y tipo identifican un recurso. La lista se presenta al plugin con el siguiente formato: “<UID>,<TYPE>;<UID>,<TYPE>”. Por ejemplo, “Instance1,instancia;Instance2\\DB1,base de datos”

Parámetro	Específico
NOMBRE_APLICACIÓN	Determina qué plugin se está utilizando. Por ejemplo, DB2, MYSQL. El servidor SnapCenter cuenta con compatibilidad integrada para las aplicaciones de la lista. Este parámetro distingue mayúsculas de minúsculas.
APP_IGNORE_ERROR	(Y o N) esto hace que SnapCenter salga o no salga cuando se encuentra un error de aplicación. Esto es útil cuando se realiza el backup de varias bases de datos y no se desea que un solo fallo detenga la operación de backup.
<RESOURCE_NAME>__APP_INSTANCE_USERNAME	Se han establecido las credenciales de SnapCenter para el recurso.
<RESOURCE_NAME>_APP_INSTANCE_PASSWORD	Se han establecido las credenciales de SnapCenter para el recurso.
<CUSTOM_PARAM>_<RESOURCE_NAME>	Todos los valores de clave personalizada de nivel de recursos están disponibles para los plug-ins con “<RESOURCE_NAME>_”. Por ejemplo, si una clave personalizada es “MASTER_SLAVE” para un recurso llamado “MySQLDB”, estará disponible como MySQLDB_MASTER_SLAVE

Utilice los códigos de salida

El plugin devuelve el estado de la operación a su host mediante códigos de salida. Cada código tiene un significado específico y el plug-in utiliza el código de salida derecho para indicar lo mismo.

En la siguiente tabla se muestran los códigos de error y su significado.

Código de salida	Específico
0	Funcionamiento correcto.
99	La operación solicitada no es compatible o está implementada.
100	Error en la operación, omita la pausa y salga. La función de inactividad está predeterminada.
101	Error en la operación, continúe con la operación de backup.
otros	Error en la operación, ejecución de la reanudación y salida.

Registrar mensajes de error

Los mensajes de error pasan del plugin al servidor de SnapCenter. El mensaje incluye el mensaje, el nivel de registro y la Marca de hora.

En la tabla siguiente se enumeran los niveles y sus propósitos.

Parámetro	Específico
INFORMACIÓN	mensaje informativo
ADVERTIR	mensaje de advertencia
ERROR	mensaje de error
DEPURAR	depurar mensaje
TRAZA	mensaje de seguimiento

Conserve la consistencia de datos

Los plugins personalizados conservan datos entre operaciones de la misma ejecución del flujo de trabajo. Por ejemplo, un plugin puede almacenar datos al final de la inactividad, que se puede utilizar durante la operación de inactivación.

Los datos que se van a conservar se establecen como parte del objeto de resultado mediante el complemento. Sigue un formato específico y se describe con detalle en cada estilo de desarrollo de complementos.

Desarrollo basado en PERL

Debe seguir ciertas convenciones mientras desarrolla el plugin con PERL.

- El contenido debe ser legible
- Debe implementar la configuración de operaciones obligatorias, el modo de inactividad y la reanudación
- Debe utilizar una sintaxis específica para devolver los resultados al agente
- El contenido debe guardarse como archivo <PLUGIN_NAME>.pm

Las operaciones disponibles son

- Setenv
- versión
- modo de inactividad
- inactivación
- clone_pre, clone_post
- restaurar_pre, restaurar
- limpieza

Manejo general del plug-in

Uso del objeto Results

Todas las operaciones de plugin personalizado deben definir el objeto Results. Este objeto envía mensajes, código de salida, stdout y stderr de vuelta al agente host.

Objeto resultados:

```
my $result = {
```

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

Devolver el objeto Results:

```
return $result;
```

Conservación de la coherencia de los datos

Es posible conservar datos entre operaciones (excepto limpieza) como parte de la misma ejecución del flujo de trabajo. Esto se logra usando pares clave-valor. Los pares clave-valor de los datos se establecen como parte del objeto de resultado y se conservan y están disponibles en las operaciones posteriores del mismo flujo de trabajo.

En el ejemplo de código siguiente se establecen los datos que se van a conservar:

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

El código anterior establece dos pares clave-valor, que están disponibles como entrada en la operación posterior. Los dos pares clave-valor se pueden acceder mediante el siguiente código:

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

```
 === Logging error messages
Cada operación puede enviar mensajes al agente host, que muestra y
almacena el contenido. Un mensaje contiene el nivel de mensaje, una Marca
de tiempo y un texto de mensaje. Se admiten mensajes multilínea.
```

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

Utilice el método msgObj para capturar un mensaje mediante el método Collect.

```
$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");
```

Aplicar mensajes al objeto resultados:

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

Uso de los espárragos del plug-in

Los plugins personalizados deben exponer los talones del plug-in. Estos son métodos a los que llama el servidor SnapCenter, en función de un flujo de trabajo.

Muñón de complemento	Opcional/obligatorio	Específico
Setenv	obligatorio	Este código auxiliar establece el entorno y el objeto de configuración. Aquí se debe realizar cualquier análisis o manejo del entorno. Cada vez que se llama un archivo stub, el archivo stub setenv se llama justo antes. Solo es necesario para complementos DE tipo PERL.
Versión	Opcional	Este código auxiliar se utiliza para obtener la versión de la aplicación.
Detección	Opcional	Este archivo stub se utiliza para detectar objetos de aplicación como la instancia o la base de datos alojada en el agente o host. Se espera que el complemento devuelva los objetos de aplicación detectados en un formato específico como parte de la respuesta. Este código auxiliar sólo se utiliza en caso de que la aplicación esté integrada con SnapDrive para Unix. Sistema de archivos Linux (Linux Flavors) es compatible. AIX/Solaris (Unix Flavors) no son compatibles.

Muñón de complemento	Opcional/obligatorio	Específico
discovery_complete	Opcional	Este archivo stub se utiliza para detectar objetos de aplicación como la instancia o la base de datos alojada en el agente o host. Se espera que el complemento devuelva los objetos de aplicación detectados en un formato específico como parte de la respuesta. Este código auxiliar sólo se utiliza en caso de que la aplicación esté integrada con SnapDrive para Unix. Sistema de archivos Linux (Linux Flavors) es compatible. AIX y Solaris (versiones Unix) no son compatibles.
Modo de inactividad	obligatorio	Este archivo stub es responsable de realizar un modo de inactividad, lo que significa colocar la aplicación en un estado en el que puede crear una copia Snapshot. Esto se llama antes de la operación de copia de Snapshot. Los metadatos de la aplicación que se conservará deben definirse como parte de la respuesta, que se devolverá durante las operaciones posteriores de clonado o restauración en la copia Snapshot de almacenamiento correspondiente en forma de parámetros de configuración.
Inactivación	obligatorio	Este código auxiliar es responsable de realizar un modo de inactividad, lo que significa poner la aplicación en un estado normal. Esto se llama después de crear una copia Snapshot.

Muñón de complemento	Opcional/obligatorio	Específico
clone_pre	opcional	Este archivo stub es responsable de realizar tareas previas a la clonación. Se supone que se utiliza la interfaz de clonación del servidor de SnapCenter integrada y se activa al realizar la operación de clonación.
clone_post	opcional	Este archivo stub es responsable de realizar tareas posteriores a la clonación. Esto supone que se utiliza la interfaz de clonación del servidor de SnapCenter integrada y se activa solo al realizar una operación de clonado.
restaurar_pre	opcional	Este archivo stub es responsable de realizar tareas prerestore. Esto supone que se utiliza la interfaz de restauración de servidor de SnapCenter incorporada y se activa al realizar una operación de restauración.
Restaurar	opcional	Este código auxiliar es responsable de realizar tareas de restauración de aplicaciones. Esto supone que se utiliza la interfaz de restauración de servidor de SnapCenter incorporada y que solo se activa al realizar una operación de restauración.

Muñón de complemento	Opcional/obligatorio	Específico
Limpieza	opcional	Este archivo stub es responsable de realizar una limpieza después de las operaciones de backup, restauración o clonado. La limpieza puede realizarse durante la ejecución normal del flujo de trabajo o en caso de que se produzca un error en el mismo. Puede inferir el nombre del flujo de trabajo bajo el cual se llama a la limpieza haciendo referencia a LA ACCIÓN de parámetro de configuración, que puede ser copia de seguridad, clonVolAndLun o archivoOrVolRestore. El parámetro DE configuración ERROR_MESSAGE indica si se produjo algún error al ejecutar el flujo de trabajo. Si ERROR_MESSAGE está definido y NO es NULL, se llama a la limpieza durante la ejecución de un fallo de flujo de trabajo.
versión_aplicación	Opcional	SnapCenter utiliza este archivo stub para que el complemento gestione el detalle de la versión de la aplicación.

Información sobre el paquete de plugins

Cada plugin debe tener la siguiente información:

```
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 = ();
```

Operaciones

Puede codificar varias operaciones, como setenv, Version, Quiesce y UnQUIESCE, que son compatibles con los plug-ins personalizados.

Funcionamiento de setenv

La operación setenv es necesaria para los complementos creados con PERL. Puede ajustar el ENV y acceder fácilmente a los parámetros del plug-in.

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

Operación de versión

La operación de versión devuelve la información de la versión de la aplicación.

```
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;
}
```

Operaciones de inactivación

La operación de inactividad realiza una operación de inactividad de la aplicación en los recursos que se enumeran en el parámetro RESOURCES.

```
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;
}
```

Funcionamiento de la reanudación

La operación de inactividad es necesaria para desactivar la activación de la aplicación. La lista de recursos está disponible en el parámetro RESOURCES.

```
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;
}
```

Estilo NATIVO

SnapCenter admite lenguajes que no SEAN DE programación PERL o lenguajes de scripting para crear complementos. Esto se conoce como programación DE estilo NATIVO, que puede ser un archivo de secuencia de comandos o LOTE.

Los plugins DE estilo NATIVO deben seguir ciertas convenciones indicadas a continuación:

El plugin debe ser ejecutable

- Para los sistemas Unix, el usuario que ejecuta el agente debe tener privilegios de ejecución en el plug-in
- En los sistemas Windows, los complementos de PowerShell deben tener el sufijo .ps1, los demás scripts de Windows deben tener el sufijo .cmd o .bat y el usuario debe ser ejecutable
- Los complementos deben reaccionar a los argumentos de la línea de comandos, como "-QUIESCE", "-unQUIESCE"
- Los plug-ins deben devolver código de salida 99 en caso de que no se haya implementado una operación o función
- Los plugins deben utilizar una sintaxis específica para devolver los resultados al servidor

Manejo general del plug-in

Mensajes de error de registro

Cada operación puede enviar mensajes al servidor, que muestra y almacena el contenido. Un mensaje contiene el nivel de mensaje, una Marca de tiempo y un texto de mensaje. Se admiten mensajes multilínea.

Formato:

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

Uso de los espárragos del plug-in

Los complementos de SnapCenter deben implementar espárragos de complemento. Estos son métodos a los que el servidor SnapCenter llama en función de un flujo de trabajo específico.

Muñón de complemento	Opcional/obligatorio	Específico
modo de inactividad	obligatorio	Este código auxiliar es responsable de realizar una pausa. Coloca la aplicación en un estado en el que podemos crear una copia Snapshot. Esto se denomina antes de la operación de copia de Snapshot de almacenamiento.
inactivación	obligatorio	Este código auxiliar es responsable de realizar una pausa. Coloca la aplicación en un estado normal. Esto se denomina después de la operación de copia de Snapshot de almacenamiento.
clone_pre	opcional	Este archivo stub es responsable de realizar tareas previas a la clonación. Esto supone que se utiliza la interfaz de clonado de SnapCenter incorporada y que solo se activa mientras se realiza la acción "clone_vol o clone_lun".
clone_post	Opcional	Este archivo stub es responsable de realizar tareas posteriores a la clonación. Esto supone que utiliza la interfaz de clonado de SnapCenter integrada y que solo se activa mientras se realizan operaciones de «clone_vol o clone_lun».
restaurar_pre	Opcional	Este archivo stub es responsable de realizar tareas previas a la restauración. Esto supone que se utiliza la interfaz de restauración de SnapCenter integrada y que solo se activa durante la operación de restauración.

Muñón de complemento	Opcional/obligatorio	Específico
restaurar	opcional	Este código auxiliar es responsable de realizar todas las acciones de restauración. Esto supone que no está utilizando la interfaz de restauración integrada. Se activa durante la operación de restauración.

Ejemplos

Windows PowerShell

Compruebe si la secuencia de comandos se puede ejecutar en el sistema. Si no puede ejecutar la secuencia de comandos, defina el desvío de Set-ExecutionPolicy para la secuencia de comandos y vuelva a intentar la operación.

```
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;
```

Estilo Java

Un complemento personalizado de Java interactúa directamente con una aplicación como base de datos, instancia, etc.

Limitaciones

Existen ciertas limitaciones que debe tener en cuenta al desarrollar un plug-in utilizando el lenguaje de programación Java.

Característica de plug-in	Plugin de Java
Complejidad	De bajo a mediano

Característica de plug-in	Plugin de Java
Huella de la memoria	Hasta 10-20 MB
Dependencias con otras bibliotecas	Bibliotecas para la comunicación de aplicaciones
Número de subprocesos	1
Tiempo de ejecución de subprocesos	Menos de una hora

Motivo de las limitaciones de Java

El objetivo del agente SnapCenter es garantizar una integración de aplicaciones sólida, segura y continua. Al admitir plug-ins de Java, es posible que los plug-ins introduzcan fugas de memoria y otros problemas no deseados. Esas cuestiones son difíciles de abordar, especialmente cuando el objetivo es mantener las cosas fáciles de usar. Si la complejidad de un complemento no es demasiado compleja, es mucho menos probable que los desarrolladores hubieran introducido los errores. El peligro del plug-in Java es que se ejecuten en la misma JVM que el propio agente de SnapCenter. Cuando el plug-in se bloquea o pierde memoria, también puede afectar negativamente al agente.

Métodos admitidos

Método	Obligatorio	Descripción	¿Cuándo y por quién?
Versión	Sí	Necesita obtener la versión del plugin.	El servidor o el agente de SnapCenter para solicitar la versión del plugin.
Modo de inactividad	Sí	Necesita realizar una pausa en la aplicación. En la mayoría de los casos, esto implica poner la aplicación en un estado donde SnapCenter Server puede crear un backup (por ejemplo, una copia Snapshot).	Antes de que el servidor de SnapCenter cree una copia de Snapshot o realice un backup en general.
Inactivación	Sí	Necesita realizar una reanudación de la aplicación. En la mayoría de los casos, esto significa volver a poner la aplicación en un estado de funcionamiento normal.	Después de que el servidor SnapCenter haya creado una copia de Snapshot o haya realizado un backup en general.

Método	Obligatorio	Descripción	¿Cuándo y por quién?
Limpieza	No	Responsable de la limpieza de cualquier cosa que el plug-in necesite limpiar.	Cuando termina un flujo de trabajo en el servidor SnapCenter (correctamente o con un error).
ClonPree	No	Debe realizar las acciones que deben realizarse antes de realizar una operación de clonado.	Cuando un usuario activa una acción "clonVol" o "clonLun" y utiliza el asistente de clonación integrado (GUI/CLI).
ClonPost	No	Debe realizar las acciones que deben realizarse después de realizar una operación de clonado.	Cuando un usuario activa una acción "clonVol" o "clonLun" y utiliza el asistente de clonación integrado (GUI/CLI).
RestauradoPre	No	Debe ejecutar acciones que deben realizarse antes de solicitar la operación de restauración.	Cuando un usuario activa una operación de restauración.
Restaurar	No	Responsable de la restauración/recuperación de una aplicación.	Cuando un usuario activa una operación de restauración.
Versión de appVersion	No	Para recuperar la versión de la aplicación que gestiona el plugin.	Como parte de la recogida de datos de ASUP en cada flujo de trabajo, como Backup/Restore/Clone.

Tutorial

En esta sección se describe cómo crear un complemento personalizado mediante el lenguaje de programación Java.

Configuración de eclipse

1. Cree un nuevo proyecto Java "TutorialPlugin" en Eclipse
2. Haga clic en **Finalizar**
3. Haga clic con el botón derecho del ratón en **nuevo proyecto** → **Propiedades** → **Java Build Path** → **Bibliotecas** → **Añadir tarros externos**
4. Desplácese a la carpeta ../lib/ del agente anfitrión y seleccione Jarras scAgent-5.0-core.jar y common-5.0.jar

5. Seleccione el proyecto y haga clic con el botón derecho del ratón en la carpeta **src** → **Nuevo** → **paquete** y cree un nuevo paquete con el nombre com.netapp.snapcreator.agent.plugin.TutorialPlugin
6. Haga clic con el botón derecho del ratón en el nuevo paquete y seleccione Nuevo → clase Java.
   a. Introduzca el nombre como TutorialPlugin.
   b. Haga clic en el botón de exploración de la superclase y busque "*AbstractPlugin". Sólo debe aparecer un resultado:

```
 "AbstractPlugin - com.netapp.snapcreator.agent.nextgen.plugin".
.. Haga clic en *Finalizar*.
.. Clase 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;
  }
}
```

Implementación de los métodos necesarios

La función de inactividad, la reanudación y la versión son métodos obligatorios que cada plugin de Java personalizado debe implementar.

A continuación, se muestra un método de versión para obtener la versión del plugin.

```
@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;
}
```

El método se pasa en un objeto de contexto. Contiene varios asistentes, por ejemplo, un registrador y un almacén de contexto, así como información sobre la operación actual (Workflow-ID, Job-ID). Podemos obtener el registrador llamando al registrador de registros final = context.getLogger();. El objeto logger proporciona métodos similares conocidos por otros marcos de registro, por ejemplo, logback. En el objeto Result, también puede especificar el código de salida. En este ejemplo, se devuelve cero, ya que no hubo ningún problema. Otros códigos de salida pueden asignar a diferentes situaciones de fallo.

Utilizando el objeto Resultado

El objeto Result contiene los parámetros siguientes:

Parámetro	Predeterminado	Descripción
Gestión de	Configuración vacía	Este parámetro se puede utilizar para enviar parámetros de configuración al servidor. Puede ser parámetros que el plugin desea actualizar. Si este cambio se refleja realmente en la configuración del servidor SnapCenter depende del parámetro APP_CONF_PERSISTENCY=y o N de la configuración.
ExitCode	0	Indica el estado de la operación. Un "0" significa que la operación se ejecutó correctamente. Otros valores indican errores o advertencias.
Apedrear	Lista vacía	Esto se puede utilizar para transmitir mensajes stdout al servidor SnapCenter.
Stderr	Lista vacía	Esto se puede utilizar para transmitir mensajes stderr de nuevo al servidor SnapCenter.
Mensajes	Lista vacía	Esta lista contiene todos los mensajes que un plug-in desea volver al servidor. El servidor SnapCenter muestra esos mensajes en la CLI o en la GUI.

El agente de SnapCenter proporciona constructores ("Patrón de creación") para todos sus tipos de resultados. Esto hace que su uso sea muy sencillo:

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

Por ejemplo, establezca el código de salida en 0, establezca las listas para stdout y stderr, defina los

parámetros de configuración y también agregue los mensajes de registro que se enviarán de nuevo al servidor. Si no necesita todos los parámetros, envíe sólo los que necesite. Como cada parámetro tiene un valor predeterminado, si quita .witEExitCode(0) del código siguiente, el resultado no se verá afectado:

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

VersionResult

VersionResult informa a SnapCenter Server de la versión del plugin. Como también hereda del resultado, contiene los parámetros config, exitCode, stdout, stderr y messages.

Parámetro	Predeterminado	Descripción
Importante	0	Campo de versión principal del plugin.
Menor	0	Campo de versión secundaria del plugin.
Parche	0	Campo de versión de revisión del plugin.
Cree	0	Cree el campo de versión del plugin.

Por ejemplo:

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

Uso del objeto de contexto

El objeto Context proporciona los siguientes métodos:

Método de contexto	Específico
String getWorkflowId();	Devuelve el ID de flujo de trabajo que utiliza el servidor SnapCenter para el flujo de trabajo actual.

Método de contexto	Específico
Config getconfig();	Devuelve la configuración que se envía desde el servidor SnapCenter al agente.

ID del flujo de trabajo

El ID de flujo de trabajo es el ID que utiliza el servidor de SnapCenter para hacer referencia a un flujo de trabajo en ejecución específico.

Gestión de

Este objeto contiene (la mayoría) los parámetros que un usuario puede establecer en la configuración del servidor SnapCenter. Sin embargo, debido a razones de seguridad, algunos de esos parámetros pueden filtrarse en el servidor. A continuación figura un ejemplo de cómo acceder a la configuración y recuperar un parámetro:

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

""// MyParameter" contiene ahora el parámetro leído en la configuración del servidor SnapCenter Si no existe una clave de parámetro config, devolverá un valor de tipo String vacío ("").

Exportando el plugin

Debe exportar el plugin para instalarlo en el host de SnapCenter.

En Eclipse, realice las siguientes tareas:

1. Haga clic con el botón derecho en el paquete básico del complemento (en nuestro ejemplo com.netapp.snapcreator.agent.plugin.TutorialPlugin).
2. Seleccione **Exportar** → **Java** → **Archivo Jar**
3. Haga clic en **Siguiente**.
4. En la siguiente ventana, especifique la ruta de acceso de archivo JAR de destino: tutorial_plugin.jar la clase base del plugin se denomina TutorialPlugin.class, el plug-in debe agregarse a una carpeta con el mismo nombre.

Si el plugin depende de bibliotecas adicionales, puede crear la siguiente carpeta: Lib/

Puede agregar archivos JAR en los que depende el plugin (por ejemplo, un controlador de base de datos). Cuando SnapCenter carga el plug-in, asocia automáticamente todos los archivos JAR de esta carpeta y los añade a la classpath.

Plugin personalizado en SnapCenter

Plugin personalizado en SnapCenter

El complemento personalizado creado con Java, PERL o estilo NATIVO puede instalarse en el host utilizando

SnapCenter Server para permitir la protección de datos de su aplicación. Debe haber exportado el plugin para instalarlo en el host SnapCenter mediante el procedimiento proporcionado en este tutorial.

Crear un archivo de descripción del plugin

Para cada plugin creado, debe tener un archivo de descripción. El archivo de descripción describe los detalles del plugin. El nombre del archivo debe ser Plugin_descriptor.xml.

Usar atributos del archivo descriptor del plugin y su importancia

Atributo	Descripción
Nombre	Nombre del plugin. Se permiten caracteres alfanuméricos. Por ejemplo, DB2, MYSQL, MongoDB Para los plugins creados con un estilo NATIVO, asegúrese de no proporcionar la extensión del archivo. Por ejemplo, si el nombre del plugin es MongoDB.sh, especifique el nombre como MongoDB.
Versión	Versión de plugin. Puede incluir tanto la versión principal como la secundaria. Por ejemplo, 1.0, 1.1, 2.0, 2.1
DisplayName	El nombre del plugin que se mostrará en SnapCenter Server. Si se escriben varias versiones del mismo complemento, asegúrese de que el nombre para mostrar es el mismo en todas las versiones.
PluginType	Idioma utilizado para crear el plug-in. Los valores admitidos son Perl, Java y Native. El tipo de complemento nativo incluye scripts de shell de Unix/Linux, scripts de Windows, Python o cualquier otro lenguaje de scripting.
OSNAME	El nombre del sistema operativo del host donde se ha instalado el plugin. Los valores válidos son Windows y Linux. Es posible que un único complemento esté disponible para su puesta en marcha en varios tipos de sistemas operativos, como el complemento DE tipo PERL.
OSVersion	La versión del sistema operativo del host donde se instaló el plugin.
ResourceName	Nombre del tipo de recurso que admite el plugin. Por ejemplo, base de datos, instancia, colecciones.

Atributo	Descripción
Padre	En caso de que el ResourceName dependa jerárquicamente de otro tipo de recurso y, a continuación, Parent determina el atributo resourcetype primario. Por ejemplo, el complemento DB2, ResourceName “Database” tiene una “instancia” principal.
RequireFileSystemPlugin	Sí o no Determina si la pestaña Recovery se muestra en el asistente Restore.
ResourceRequiresAuthentication	Sí o no Determina si los recursos, que se detectan automáticamente o no se detectan automáticamente necesitan credenciales para realizar las operaciones de protección de datos después de detectar el almacenamiento.
RequireFileSystemClone	Sí o no Determina si el plugin requiere la integración del plugin del sistema de archivos para el flujo de trabajo de clonado.

A continuación, se muestra un ejemplo del archivo Plugin_descriptor.xml para el plugin personalizado DB2:

```
<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>
```

Creación de un archivo ZIP

Después de desarrollar un plugin y crear un archivo descriptor, es necesario añadir los archivos del plugin y el archivo Plugin_descriptor.xml a una carpeta y zip.

Debe tener en cuenta lo siguiente antes de crear un archivo ZIP:

- El nombre de script debe ser el mismo que el del plugin.
- Para el plugin PERL, la carpeta ZIP debe contener una carpeta con el archivo de script y el archivo descriptor debe estar fuera de esta carpeta. El nombre de la carpeta debe ser el mismo que el del plugin.
- Para los plugins distintos al plugin PERL, la carpeta ZIP debe contener el descriptor y los archivos de script.
- La versión de SO debe ser un número.

Ejemplos:

- DB2 plug-in: Agregue el archivo DB2.pm y Plugin_descriptor.xml a “DB2.zip”.
- Plug-in desarrollado con Java: Añada archivos JAR, archivos JAR dependientes y archivo Plugin_descriptor.xml a una carpeta y zip.

Cargando el archivo ZIP del plugin

Es necesario cargar el archivo ZIP del plugin en el servidor de SnapCenter para que el plugin se pueda implementar en el host deseado.

Puede cargar el plugin mediante la interfaz de usuario o cmdlets de.

UI:

- Cargue el archivo ZIP del plug-in como parte del asistente de flujo de trabajo **Add** o **Modify Host**
- Haga clic en **“Seleccionar para cargar el complemento personalizado”**

PowerShell:

- Cmdlet Upload-SmPluginPackage

  Por ejemplo, PS> Upload-SmPluginPackage -AbsolutePath c:\DB2_1.zip

  Para obtener información detallada sobre los cmdlets de PowerShell, use la ayuda de cmdlets de SnapCenter o consulte la información de referencia sobre cmdlets.

"Guía de referencia de cmdlets de SnapCenter Software".

Implementación de los plugins personalizados

El complemento personalizado cargado ahora está disponible para su implementación en el host deseado como parte del flujo de trabajo **Add** y **Modify Host**. Es posible cargar varias versiones de plugins en SnapCenter Server y seleccionar la versión deseada para implementarla en un host específico.

Para obtener más información sobre cómo cargar el plugin, consulte, "Añada hosts e instale paquetes de plugins en hosts remotos"