Sviluppare un plug-in per l'applicazione

SnapCenter Software 4.9

NetApp
March 20, 2024

Sommario

Sviluppare un plug-in per l’applicazione

Panoramica

Il server SnapCenter consente di implementare e gestire le applicazioni come plug-in per SnapCenter. Le applicazioni di tua scelta possono essere collegate al server SnapCenter per la protezione dei dati e. funzionalità di gestione.

SnapCenter consente di sviluppare plug-in personalizzati utilizzando diversi linguaggi di programmazione. È possibile Sviluppare un plug-in personalizzato utilizzando Perl, Java, BATCH o altri linguaggi di scripting.

Per utilizzare plug-in personalizzati in SnapCenter, è necessario eseguire le seguenti operazioni:

- Creare un plug-in per l’applicazione seguendo le istruzioni di questa guida
- Creare un file di descrizione
- Esportare il plug-in personalizzato per installarlo sull’host SnapCenter
- Caricare il file zip del plug-in nel server SnapCenter

Gestione di plug-in generici in tutte le chiamate API

Per ogni chiamata API, utilizzare le seguenti informazioni:

- Parametri del plug-in
- Codici di uscita
- Registrare i messaggi di errore
- Coerenza dei dati

Utilizzare i parametri del plug-in

Un insieme di parametri viene passato al plug-in come parte di ogni chiamata API effettuata. La seguente tabella elenca le informazioni specifiche per i parametri.

Parametro	Scopo
AZIONE	Determina il nome del flusso di lavoro. Ad esempio, Discover, backup, fileOrVolRestore o. CloneVolAndLun
RISORSE	Elenca le risorse da proteggere. Una risorsa è identificata da UID e Type. L’elenco viene presentato al plug-in nel seguente formato: "<UID>,<TYPE>;<UID>,<TYPE>". Ad esempio, "Instance1,Instance;Instance2,DB1,Database"

Parametro	Scopo
NOME_APP	Determina quale plug-in viene utilizzato. Ad esempio, DB2, MYSQL. Il server SnapCenter dispone di un supporto integrato per le applicazioni elencate. Questo parametro fa distinzione tra maiuscole e minuscole.
APP_IGNORE_ERROR	(Y o N) questo causa l'uscita di SnapCenter quando si verifica un errore dell'applicazione. Ciò è utile quando si esegue il backup di più database e non si desidera un singolo errore interrompere l'operazione di backup.
<RESOURCE_NAME>__APP_INSTANCE_USERNAME	La credenziale SnapCenter è impostata per la risorsa.
<RESOURCE_NAME>_APP_INSTANCE_PASSWORD	La credenziale SnapCenter è impostata per la risorsa.
<RESOURCE_NAME>_<CUSTOM_PARAM>	Il valore della chiave personalizzata di ogni livello di risorsa è disponibile per i plug-in con prefisso "<RESOURCE_NAME>_". Ad esempio, se un La chiave personalizzata è "MASTER_SLAVE" per una risorsa Denominato "MySQLDB", sarà disponibile come MySQLDB_MASTER_SLAVE

Utilizzare i codici di uscita

Il plug-in restituisce lo stato dell'operazione all'host mediante i codici di uscita. Ciascuno il codice ha un significato specifico e il plug-in utilizza il codice di uscita destro per indicare lo stesso.

La tabella seguente illustra i codici di errore e il relativo significato.

Codice di uscita	Scopo
0	Operazione riuscita.
99	L'operazione richiesta non è supportata o implementata.
100	Operazione non riuscita, ignorare e uscire. Unquiesce è per impostazione predefinita.
101	Operazione non riuscita, continuare con l'operazione di backup.
altro	Operazione non riuscita, eseguire senza problemi e uscire.

Registrare i messaggi di errore

I messaggi di errore vengono passati dal plug-in al server SnapCenter. Il messaggio include il messaggio, il livello di log e l'indicatore di data e ora.

La tabella seguente elenca i livelli e i relativi scopi.

Parametro	Scopo
INFO	messaggio informativo
ATTENZIONE	messaggio di avviso
ERRORE	messaggio di errore
DEBUG	messaggio di debug
TRACCIA	messaggio di traccia

Preservare la coerenza dei dati

I plug-in personalizzati mantengono i dati tra le operazioni della stessa esecuzione del workflow. Per ad esempio, un plug-in può memorizzare i dati alla fine di quiesce, che possono essere utilizzati durante le richieste operazione.

I dati da conservare vengono impostati come parte dell'oggetto risultato tramite plug-in. Segue un formato specifico ed è descritto in dettaglio in ogni stile di sviluppo del plug-in.

Sviluppo BASATO SU PERL

È necessario seguire alcune convenzioni durante lo sviluppo del plug-in con PERL.

- Il contenuto deve essere leggibile
- Devono implementare le operazioni obbligatorie setenv, quiesce e senza richieste
- Deve utilizzare una sintassi specifica per restituire i risultati all'agente
- Il contenuto deve essere salvato come file <PLUGIN_NAME>.pm

Le operazioni disponibili sono

- Setenv
- versione
- quiesce
- non fare domande
- clone_pre, clone_post
- restore_pre, ripristino
- pulizia

Gestione generale dei plug-in

Utilizzo dell'oggetto Results

Ogni operazione di plug-in personalizzata deve definire l'oggetto Results. Questo oggetto invia messaggi, codice di uscita, stdout e stderr all'agente host.

Oggetto Results (risultati):

```
my $result = {
```

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

Restituzione dell'oggetto Results:

```
return $result;
```

Preservare la coerenza dei dati

È possibile conservare i dati tra le operazioni (eccetto la pulizia) come parte della stessa esecuzione del flusso di lavoro. Ciò avviene utilizzando coppie chiave-valore. Le coppie chiave-valore dei dati vengono impostate come parte dell'oggetto risultato e vengono conservate e disponibili nelle successive operazioni dello stesso flusso di lavoro.

Il seguente esempio di codice imposta i dati da conservare:

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

Il codice precedente imposta due coppie chiave-valore, che sono disponibili come input nell'operazione successiva. Le due coppie chiave-valore sono accessibili utilizzando il seguente codice:

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

```
 === Logging error messages
Ciascuna operazione può inviare i messaggi all'agente host, che visualizza
e memorizza il contenuto. Un messaggio contiene il livello del messaggio,
un indicatore data e ora e un testo del messaggio. Sono supportati i
messaggi multilinea.
```

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

Utilizzare msgObj per acquisire un messaggio utilizzando il metodo 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");
```

Applicare i messaggi all'oggetto Results:

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

Utilizzo di stub plug-in

I plug-in personalizzati devono esporre gli stub plug-in. Questi sono i metodi che il server SnapCenter chiama, in base a un flusso di lavoro.

Stub plug-in	Opzionale/obbligatorio	Scopo
Setenv	obbligatorio	Questo stub imposta l'ambiente e l'oggetto di configurazione. Qualsiasi analisi o gestione dell'ambiente deve essere eseguita qui. Ogni volta che viene chiamato uno stub, lo stub setenv viene chiamato poco prima. È necessario solo per i plug-in PERL-style.
Versione	Opzionale	Questo stub viene utilizzato per ottenere la versione dell'applicazione.
Scopri	Opzionale	Questo stub viene utilizzato per rilevare gli oggetti dell'applicazione come istanze o database ospitati sull'agente o sull'host. Il plug-in restituirà gli oggetti dell'applicazione rilevati in un formato specifico come parte della risposta. Questo stub viene utilizzato solo nel caso in cui l'applicazione sia integrata con SnapDrive per Unix. Il file system Linux (Linux Flavors) è supportato. AIX/Solaris (Unix Flavors) non sono supportati.

Stub plug-in	Opzionale/obbligatorio	Scopo
discovery_complete	Opzionale	Questo stub viene utilizzato per rilevare gli oggetti dell'applicazione come istanze o database ospitati sull'agente o sull'host. Il plug-in restituirà gli oggetti dell'applicazione rilevati in un formato specifico come parte della risposta. Questo stub viene utilizzato solo nel caso in cui l'applicazione sia integrata con SnapDrive per Unix. Il file system Linux (Linux Flavors) è supportato. AIX e Solaris (versioni Unix) non sono supportati.
Quiesce	obbligatorio	Questo stub è responsabile dell'esecuzione di un quiesce, il che significa mettere l'applicazione in uno stato in cui è possibile creare una copia Snapshot. Questo viene chiamato prima dell'operazione di copia Snapshot. I metadati dell'applicazione da conservare devono essere impostati come parte della risposta, che verrà restituita durante le successive operazioni di clonazione o ripristino sulla copia Snapshot dello storage corrispondente sotto forma di parametri di configurazione.
Senza richieste	obbligatorio	Questo stub è responsabile dell'esecuzione di un'operazione senza oggetto, il che significa mettere l'applicazione in uno stato normale. Questa operazione viene richiamata dopo la creazione di una copia Snapshot.

Stub plug-in	Opzionale/obbligatorio	Scopo
clone_pre	opzionale	Questo stub è responsabile dell'esecuzione delle attività di precolona. Ciò presuppone che si stia utilizzando l'interfaccia di clonazione del server SnapCenter integrata e che venga attivata durante l'esecuzione dell'operazione di clonazione.
clone_post	opzionale	Questo stub è responsabile dell'esecuzione delle attività post-clone. Ciò presuppone che si stia utilizzando l'interfaccia di clonazione del server SnapCenter integrata e che venga attivata solo quando si esegue un'operazione di clonazione.
ripristina_pre	opzionale	Questo stub è responsabile dell'esecuzione delle attività di prerestore. Ciò presuppone che si stia utilizzando l'interfaccia di ripristino del server SnapCenter integrata e che venga attivata durante l'esecuzione dell'operazione di ripristino.
Ripristinare	opzionale	Questo stub è responsabile dell'esecuzione delle attività di ripristino delle applicazioni. Questo presuppone che si stia utilizzando l'interfaccia di ripristino del server SnapCenter integrata e viene attivato solo quando si esegue l'operazione di ripristino.

Stub plug-in	Opzionale/obbligatorio	Scopo
Pulizia	opzionale	Questo stub è responsabile dell'esecuzione della pulizia dopo le operazioni di backup, ripristino o clonazione. La pulizia può avvenire durante la normale esecuzione del flusso di lavoro o in caso di errore del flusso di lavoro. È possibile dedurre il nome del flusso di lavoro con cui viene chiamata la pulizia facendo riferimento ALL'AZIONE del parametro di configurazione, che può essere backup, cloneVolAndLun o fileOrVolRestore. Il parametro di configurazione ERROR_MESSAGE indica se si è verificato un errore durante l'esecuzione del flusso di lavoro. Se ERROR_MESSAGE è definito e NON NULL, la pulitura viene richiamata durante l'esecuzione di un errore del workflow.
versione_app	Opzionale	Questo stub viene utilizzato da SnapCenter per ottenere l'applicazione dettagli della versione gestiti dal plug-in.

Informazioni sul pacchetto plug-in

Ogni plug-in deve avere le seguenti informazioni:

```
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 = ();
```

Operazioni

È possibile codificare diverse operazioni come setenv, Version, Quiesce e Unquiesce, supportate dai plug-in personalizzati.

Operazione setenv

L'operazione setenv è necessaria per i plug-in creati utilizzando PERL. È possibile impostare ENV e accedere facilmente ai parametri del plug-in.

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

Funzionamento della versione

L'operazione di versione restituisce le informazioni sulla versione dell'applicazione.

```
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;
}
```

Interrompere le operazioni

L'operazione quiesce esegue l'operazione di quiesce dell'applicazione sulle risorse elencate nel parametro 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;
}
```

Operazione senza problemi

L'operazione Unquiesce è necessaria per interrompere l'applicazione. L'elenco delle risorse è disponibile nel parametro 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;
}
```

Stile NATIVO

SnapCenter supporta linguaggi di programmazione o scripting non PERL per creare plug-in. Questa è nota come programmazione in stile NATIVO, che può essere un file script o BATCH.

I plug-in DI stile NATIVO devono seguire alcune convenzioni fornite di seguito:

Il plug-in deve essere eseguibile

- Per i sistemi Unix, l'utente che esegue l'agente deve disporre dei privilegi di esecuzione sul plug-in
- Per i sistemi Windows, i plug-in PowerShell devono avere il suffisso .ps1, altre finestre gli script devono avere il suffisso .cmd o .bat e devono essere eseguibili dall'utente
- I plug-in devono reagire a argomenti della riga di comando come "-quiesce", "-unquiesce"
- I plug-in devono restituire il codice di uscita 99 nel caso in cui un'operazione o una funzione non sia implementata
- I plug-in devono utilizzare una sintassi specifica per restituire i risultati al server

Gestione generale dei plug-in

Registrazione dei messaggi di errore

Ogni operazione può inviare messaggi al server, che visualizza e memorizza il contenuto. Un messaggio contiene il livello del messaggio, un indicatore data e ora e un testo del messaggio. Sono supportati i messaggi multilinea.

Formato:

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

Utilizzo di stub plug-in

I plug-in SnapCenter devono implementare i plug-in stub. Si tratta di metodi richiamati dal server SnapCenter in base a un flusso di lavoro specifico.

Stub plug-in	Opzionale/obbligatorio	Scopo
quiesce	obbligatorio	Questo stub è responsabile dell'esecuzione di un quiesce. Posiziona il Applicazione in uno stato in cui è possibile creare una copia Snapshot. Questo viene chiamato prima dell'operazione di copia Snapshot dello storage.
non fare domande	obbligatorio	Questo stub è responsabile dell'esecuzione di una richiesta. È così l'applicazione in uno stato normale. Questo viene chiamato dopo lo storage Operazione di copia Snapshot.
clone_pre	opzionale	Questo stub è responsabile dell'esecuzione delle attività pre-clonate. Questo presuppone che si stia utilizzando l'interfaccia di cloning SnapCenter integrata e che venga attivata solo durante l'esecuzione dell'azione "clone_vol o clone_lun".
clone_post	Opzionale	Questo stub è responsabile dell'esecuzione delle attività post-clone. Questo presuppone che si stia utilizzando l'interfaccia di cloning SnapCenter integrata e che venga attivata solo durante l'esecuzione delle operazioni "clone_vol o clone_lun".
ripristina_pre	Opzionale	Questo stub è responsabile dell'esecuzione delle attività di pre-ripristino. Ciò presuppone che si stia utilizzando l'interfaccia di ripristino SnapCenter integrata e che venga attivata solo durante l'esecuzione dell'operazione di ripristino.

Stub plug-in	Opzionale/obbligatorio	Scopo
ripristinare	opzionale	Questo stub è responsabile dell'esecuzione di tutte le azioni di ripristino. Questo presuppone che non si stia utilizzando un'interfaccia di ripristino integrata. Viene attivato durante l'esecuzione dell'operazione di ripristino.

Esempi

Windows PowerShell

Controllare se lo script può essere eseguito sul sistema. Se non è possibile eseguire lo script, impostare il bypass Set-ExecutionPolicy per lo script e riprovare l'operazione.

```
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;
```

Stile Java

Un plug-in personalizzato Java interagisce direttamente con un'applicazione come database, istanze e così via.

Limitazioni

Esistono alcune limitazioni che è necessario tenere presenti durante lo sviluppo di un plug-in utilizzando il linguaggio di programmazione Java.

Caratteristica del plug-in	Plug-in Java
Complessità	Da basso a medio

Caratteristica del plug-in	Plug-in Java
Impatto della memoria	Fino a 10-20 MB
Dipendenze da altre librerie	Librerie per la comunicazione applicativa
Numero di thread	1
Runtime del thread	Meno di un'ora

Motivo delle limitazioni Java

L'obiettivo dell'agente SnapCenter è garantire un'integrazione applicativa continua, sicura e solida. Grazie al supporto dei plug-in Java, è possibile che i plug-in introducano perdite di memoria e altri problemi indesiderati. Questi problemi sono difficili da affrontare, soprattutto quando l'obiettivo è quello di mantenere le cose semplici da utilizzare. Se la complessità di un plug-in non è troppo complessa, è molto meno probabile che gli sviluppatori abbiano introdotto gli errori. Il pericolo del plug-in Java è che lo sia In esecuzione nella stessa JVM dell'agente SnapCenter stesso. In caso di crash o perdita di memoria, il plug-in potrebbe avere un impatto negativo sull'Agent.

Metodi supportati

Metodo	Obbligatorio	Descrizione	Chiamate quando e da chi?
Versione	Sì	Deve restituire la versione del plug-in.	Dal server o dall'agente SnapCenter per richiedere la versione di il plug-in.
Quiesce	Sì	Deve eseguire un quiesce sull'applicazione. Nella maggior parte dei casi, ciò significa mettere l'applicazione in uno stato in cui il server SnapCenter può creare un backup (ad esempio, una copia Snapshot).	Prima che il server SnapCenter crei una copia Snapshot o. esegue un backup in generale.
Senza richieste	Sì	Deve eseguire un'operazione senza domande sull'applicazione. Nella maggior parte dei casi, questo significa riportare l'applicazione in uno stato operativo normale.	Dopo che il server SnapCenter ha creato una copia Snapshot eseguito un backup in generale.

Metodo	Obbligatorio	Descrizione	Chiamate quando e da chi?
Pulizia	No	Responsabile della pulizia di qualsiasi elemento che il plug-in deve pulire.	Al termine di un flusso di lavoro sul server SnapCenter (con esito positivo o con errore).
ClonePre	No	Dovrebbe eseguire azioni che devono essere eseguite prima di eseguire un'operazione di clonazione.	Quando un utente attiva un'azione "cloneVol" o "cloneLun" e utilizza la procedura guidata di cloning integrata (GUI/CLI).
ClonePost	No	Dovrebbe eseguire azioni che devono avvenire dopo l'esecuzione di un'operazione di clonazione.	Quando un utente attiva un'azione "cloneVol" o "cloneLun" e utilizza la procedura guidata di cloning integrata (GUI/CLI).
RestorePre	No	Dovrebbe eseguire le azioni che devono essere eseguite prima di richiamare l'operazione di ripristino.	Quando un utente attiva un'operazione di ripristino.
Ripristinare	No	Responsabile dell'esecuzione di un ripristino/ripristino dell'applicazione.	Quando un utente attiva un'operazione di ripristino.
AppVersion	No	Per recuperare la versione dell'applicazione gestita dal plug-in.	Come parte della raccolta di dati ASUP in ogni flusso di lavoro come Backup/Restore/Clone.

Esercitazione

In questa sezione viene descritto come creare un plug-in personalizzato utilizzando il linguaggio di programmazione Java.

Configurare l'eclissi

1. Creare un nuovo progetto Java "TutorialPlugin" in Eclipse
2. Fare clic su **fine**
3. Fare clic con il pulsante destro del mouse su **nuovo progetto** → **Proprietà** → **Java Build Path** → **Librerie** → **Aggiungi jar esterni**

4. Accedere alla cartella ../lib/ dell'agente host e selezionare jars scAgent-5.0-core.jar e common-5.0.jar
5. Selezionare il progetto e fare clic con il pulsante destro del mouse sulla cartella **src** → **New** → **Package** e creare un nuovo pacchetto con il nome com.netapp.snapcreator.agent.plugin.TutorialPlugin
6. Fare clic con il pulsante destro del mouse sul nuovo pacchetto e selezionare New → Java Class.
   a. Immettere il nome come TutorialPlugin.
   b. Fare clic sul pulsante di ricerca delle superclassi e cercare "*AbstractPlugin". Dovrebbe essere visualizzato un solo risultato:

```
 "AbstractPlugin - com.netapp.snapcreator.agent.nextgen.plugin".
.. Fare clic su *fine*.
.. 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;
  }
}
```

Implementazione dei metodi richiesti

Quiesce, unquiesce e version sono metodi obbligatori che ogni plug-in Java personalizzato deve implementare.

Di seguito viene riportato un metodo di versione per restituire la versione del 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;
}
```

Il metodo viene passato in un oggetto di contesto. Contiene più assistenti, ad esempio un Logger e un archivio di contesto, nonché le informazioni sull'operazione corrente (workflow-ID, job-ID). Possiamo ottenere il logger chiamando il logger finale = Context.GetLogger();. L'oggetto logger fornisce metodi simili noti da altri framework di logging, ad esempio logback. Nell'oggetto risultato, è anche possibile specificare il codice di uscita. In questo esempio, viene restituito zero, poiché non si è verificato alcun problema. Altri codici di uscita possono essere associati a diversi scenari di guasto.

Utilizzo dell'oggetto risultato

L'oggetto Result contiene i seguenti parametri:

Parametro	Predefinito	Descrizione
Config	Vuoto config	Questo parametro può essere utilizzato per inviare nuovamente i parametri di configurazione al server. IT possono essere parametri che il plug-in desidera aggiornare. Se questo cambiamento è La configurazione sul server SnapCenter dipende da IL parametro APP_CONF_PERSISTENCY=Y o N nella configurazione.
ExitCode	0	Indica lo stato dell'operazione. "0" indica che l'operazione è stata eseguita eseguito correttamente. Altri valori indicano errori o avvisi.
Stdout	Vuoto Elenco	Questa funzione può essere utilizzata per trasmettere messaggi stdout a SnapCenter Server.
Stderr	Vuoto Elenco	Questa funzione può essere utilizzata per ritrasmettere i messaggi stderr a SnapCenter Server.
Messaggi	Vuoto Elenco	Questo elenco contiene tutti i messaggi che un plug-in desidera restituire a server. Il server SnapCenter visualizza questi messaggi nella CLI o nella GUI.

L'agente SnapCenter fornisce costruttori ("Modello di costruttore") per tutti i tipi di risultato. Questo rende l'utilizzo molto semplice:

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

Ad esempio, impostare il codice di uscita su 0, impostare gli elenchi per stdout e stderr, impostare i parametri di configurazione e aggiungere anche i messaggi di registro che verranno rinviati al server. Se non sono necessari tutti i parametri, inviare solo quelli necessari. Poiché ogni parametro ha un valore predefinito, se si rimuove .withExitCode(0) dal codice riportato di seguito, il risultato non viene influenzato:

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

VersionResult

VersionResult informa il server SnapCenter della versione del plug-in. Come eredita anche Di conseguenza, contiene i parametri config, exitCode, stdout, stderr e messaggi.

Parametro	Predefinito	Descrizione
Maggiore	0	Principale campo di versione del plug-in.
Minore	0	Campo versione minore del plug-in.
Patch	0	Campo della versione della patch del plug-in.
Costruire	0	Campo della versione di build del plug-in.

Ad esempio:

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

Utilizzo dell'oggetto di contesto

L'oggetto Context fornisce i seguenti metodi:

Metodo di contesto	Scopo
Stringa GetWorkflowId();	Restituisce l'id del flusso di lavoro utilizzato dal server SnapCenter per flusso di lavoro corrente.

Metodo di contesto	Scopo
Config getConfig();	Restituisce la configurazione inviata dal server SnapCenter a Agente.

ID flusso di lavoro

L'ID del flusso di lavoro è l'id utilizzato dal server SnapCenter per fare riferimento a un'esecuzione specifica workflow.

Config

Questo oggetto contiene la maggior parte dei parametri che un utente può impostare nella configurazione di Server SnapCenter. Tuttavia, a causa di motivi di sicurezza, alcuni di questi parametri potrebbero ottenere filtrato sul lato server. Di seguito viene riportato un esempio di come accedere alla configurazione e al recupero un parametro:

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

""// myParameter" contiene ora il parametro letto dalla configurazione sul server SnapCenter Se una chiave del parametro di configurazione non esiste, restituirà una stringa vuota ("").

Esportazione del plug-in

È necessario esportare il plug-in per installarlo sull'host SnapCenter.

In Eclipse eseguire le seguenti operazioni:

1. Fare clic con il pulsante destro del mouse sul pacchetto di base del plug-in (nel nostro esempio com.netapp.snapcreator.agent.plugin.TutorialPlugin).
2. Selezionare **Export → Java → jar file**
3. Fare clic su **Avanti**.
4. Nella finestra seguente, specificare il percorso del file jar di destinazione: tutorial_plugin.jar La classe di base del plug-in è denominata TutorialPlugin.class, il plug-in deve essere aggiunto a una cartella con lo stesso nome.

Se il plug-in dipende da librerie aggiuntive, è possibile creare la seguente cartella: Lib/

È possibile aggiungere file jar, da cui dipende il plug-in (ad esempio, un driver di database). Quando SnapCenter carica il plug-in, che automaticamente associa tutti i file jar in questa cartella ad esso e. li aggiunge al classpath.

Plug-in personalizzato in SnapCenter

Plug-in personalizzato in SnapCenter

Il plug-in personalizzato creato utilizzando Java, PERL o lo stile NATIVO può essere installato sull'host

utilizzando il server SnapCenter per abilitare la protezione dei dati dell'applicazione. È necessario esportare il plug-in per installarlo sull'host SnapCenter utilizzando la procedura fornita in questo tutorial.

Creazione di un file di descrizione del plug-in

Per ogni plug-in creato, è necessario disporre di un file di descrizione. Il file di descrizione descrive i dettagli del plug-in. Il nome del file deve essere Plugin_descriptor.xml.

Utilizzo degli attributi del file del descrittore del plug-in e del relativo significato

Attributo	Descrizione
Nome	Nome del plug-in. Sono consentiti caratteri alfanumerici. Ad esempio, DB2, MYSQL, MongoDB Per i plug-in creati in stile NATIVO, assicurarsi di non fornire l'estensione del file. Ad esempio, se il nome del plug-in è MongoDB.sh, specificare il nome come MongoDB.
Versione	Versione del plug-in. Può includere sia la versione principale che quella secondaria. Ad esempio, 1.0, 1.1, 2.0, 2.1
DisplayName	Il nome del plug-in da visualizzare nel server SnapCenter. Se vengono scritte più versioni dello stesso plug-in, assicurarsi che il nome visualizzato sia lo stesso per tutte le versioni.
Tipo di plug-in	Lingua utilizzata per creare il plug-in. I valori supportati sono Perl, Java e Native. Il tipo di plug-in nativo include shell script Unix/Linux, script Windows, Python o qualsiasi altro linguaggio di scripting.
Nome dell'OSName	Il nome del sistema operativo host in cui è installato il plug-in. I valori validi sono Windows e. Linux. È possibile che un singolo plug-in sia disponibile per l'implementazione su diversi tipi di sistemi operativi, come IL plug-in DI TIPO PERL.
Versione del sistema operativo	La versione del sistema operativo host in cui è installato il plug-in.
ResourceName	Nome del tipo di risorsa supportato dal plug-in. Ad esempio, database, istanza, raccolte.

Attributo	Descrizione
Origine	Nel caso, ResourceName dipende gerarchicamente da un altro tipo di risorsa, quindi Il tipo di risorsa principale determina il tipo di risorsa principale. Ad esempio, il plug-in DB2, ResourceName "Database" ha un'istanza padre.
RequireFileSystemPlugin	Sì o No Determina se la scheda Recovery è visualizzato nella procedura guidata di ripristino.
ResourceRequiresAuthentication	Sì o No Determina se le risorse rilevate automaticamente o meno il rilevamento automatico richiede credenziali per eseguire le operazioni di protezione dei dati dopo rilevamento dello storage.
RequireFileSystemClone	Sì o No Determina se il plug-in richiede l'integrazione del plug-in del file system per il clone workflow.

Un esempio del file Plugin_descriptor.xml per il plug-in personalizzato DB2 è il seguente:

```
<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>
```

Creazione di un file ZIP

Dopo aver sviluppato un plug-in e creato un file descrittore, è necessario aggiungere i file plug-in e. Il file Plugin_descriptor.xml in una cartella e comprimerlo.

Prima di creare un file ZIP, è necessario prendere in considerazione quanto segue:

- Il nome dello script deve essere uguale al nome del plug-in.
- Per IL plug-in PERL, la cartella ZIP deve contenere una cartella con il file script e il il file del descrittore deve essere esterno a questa cartella. Il nome della cartella deve essere lo stesso di nome del plug-in.
- Per i plug-in diversi dal plug-in PERL, la cartella ZIP deve contenere il descrittore e. i file di script.
- La versione del sistema operativo deve essere un numero.

Esempi:

- Plug-in DB2: Aggiungere i file DB2.pm e Plugin_descriptor.xml a "DB2.zip".

- Plug-in sviluppato utilizzando Java: Aggiungere file jar, file jar dipendenti e. Plugin_descriptor.xml in una cartella e comprimerlo.

Caricamento del file ZIP del plug-in

È necessario caricare il file ZIP del plug-in sul server SnapCenter in modo che il plug-in sia disponibile per implementazione sull'host desiderato.

È possibile caricare il plug-in utilizzando l'interfaccia utente o i cmdlet.

UI:

- Caricare il file ZIP del plug-in come parte della procedura guidata del flusso di lavoro **Add** o **Modify host**
- Fare clic su **"Select to upload custom plug-in"**

PowerShell:

- Cmdlet Upload-SmPluginPackage

  Ad esempio, PS> Upload-SmPluginPackage -AbsolutePath c: DB2_1.zip

  Per informazioni dettagliate sui cmdlet PowerShell, utilizzare la guida in linea del cmdlet SnapCenter o. consultare le informazioni di riferimento del cmdlet.

"Guida di riferimento al cmdlet del software SnapCenter".

Implementazione dei plug-in personalizzati

Il plug-in personalizzato caricato è ora disponibile per l'implementazione sull'host desiderato come parte di Workflow **Add** e **Modify host**. È possibile caricare più versioni dei plug-in su Server SnapCenter ed è possibile selezionare la versione desiderata da implementare su un host specifico.

Per ulteriori informazioni su come caricare il plug-in, vedere "Aggiungere host e installare pacchetti plug-in su host remoti"

Informazioni sul copyright

Informazioni sul marchio commerciale