Citrix XenServer ® 7.3 Changed Block Tracking Developer Guide

Published December 2017
1.0 Edition

Citrix XenServer ® 7.3 Changed Block Tracking Developer Guide

Version: 7.3

Citrix Systems, Inc.
851 West Cypress Creek Road
Fort Lauderdale, FL 33309
United States of America

Disclaimers

Trademarks

Citrix®
XenServer ®
XenCenter ®

CITRIX

Contents

CITRIX

Chapter 1. Introduction

Changed block tracking provides a set of features and APIs that enable you to develop fast and space-efficient incremental backup solutions for XenServer.

Changed block tracking is available only to customers with XenServer Enterprise Edition. If a customer without Enterprise Edition attempts to use an incremental backup solution for XenServer that uses changed block tracking, they are prevented from enabling changed block tracking on new VDIs. However, if the customer has existing VDIs with changed block tracking enabled, they can still perform other changed block tracking actions on these VDIs.

1.1. How changed block tracking works

When changed block tracking is enabled for a virtual disk image (VDI), any blocks that are changed in that VDI are recorded in a log file. Every time the VDI is snapshotted, this log file can be used to identify the blocks that have changed since the VDI was last snapshotted. This provides the capability to backup only those blocks that have changed.

After the changed blocks have been exported, the full VDI snapshots can now be changed into metadata-only snapshots by destroying the data associated with them and leaving only the changed block information. These metadata only snapshots are linked both to the preceding metadata-only snapshot and to the following metadata-only snapshot. This provides a chain of metadata that records the full history of changes to this VDI since changed block tracking was enabled.

The changed block tracking feature also takes advantage of network block device (NBD) capabilities to perform the export of data from the changed blocks.

1.2. Benefits of changed block tracking

Unlike some other incremental backup solutions, changed block tracking does not require that the customer keep a snapshot of the last known good state of a VDI available on the host or a storage repository (SR) to compare the current state to. The customer needs less disk space because, instead of handling and storing large VDIs, with changed block tracking they instead can choose to store space-efficient metadata-only snapshot files.

Changed block tracking also saves the customer time as well as space. Other backup solutions export a snapshot of the whole VDI every single time the VDI is backed up. This is a time-consuming process and the customer has to pay that time cost every time they take a backup. With changed block tracking enabled, the first back up exports a snapshot of the whole VDI. However, subsequent backups only export the blocks in the VDI that have changed since the previous backup. This decreases the time required to export the backup in proportion to the percentage of blocks that have changed.

For example, it can take around 10 hours to export a backup of a full 1 TB VDI. If, after a week, 5% of the blocks in that VDI have changed, exporting the backup takes 5% of the time - 30 minutes. A backup taken after a day has even fewer changed blocks and takes even less time to export.

The savings in time and space that changed block tracking provides makes it a preferable backup solution for customers using XenServer. The simple API that XenServer exposes makes it easy for you to develop an incremental backup solution that delivers these benefits to the end user. You can use this API through the language-agnostic remote procedure calls (RPCs) or take advantage of the language bindings provided for C, C#, Java, Python and PowerShell.

CITRIX

Chapter 2. Getting started using changed block tracking

This section steps through the process of using changed block tracking to create incremental backups.

Before getting started with changed block tracking, we recommend that you read the Citrix XenServer Software Developer Kit Guide. This document contains information to help you become familiar with developing for XenServer.

The examples provided in these steps use the Python binding for the Management API.

- For more information about the individual XML-RPC calls, see the Citrix XenServer Management API.
- For more detailed information about individual steps in this process, see the following chapters.

Full Python examples are provided on GitHub.

The NBD connection examples provided in these steps use the Linux nbd-client. However, you can use any NBD client that supports the "fixed newstyle" version of the NBD protocol. For more information, see the NBD protocol documentation.

> **Note:**
>
> If you are using the Linux upstream NBD client, a minimum version of 3.15 is required to support TLS.

2.1. Prerequisites

Before you start, set up or implement an NBD client at the backup location that supports the “fixed newstyle” version of the NBD protocol. For more information, see Section 7.2, “Exporting the changed blocks using an NBD client”.

Enable NBD connections on your network. For more information, see Chapter 3, *Enabling NBD connections on XenServer*.

2.2. Procedure

This procedure is broken down into three sections:

- Section 2.2.1, “Setting up changed block tracking”: Perform the steps in this section once, when you start using changed block tracking, to enable the changed block tracking capability and export a base snapshot that the incremental, changed block exported data is compared to.
- Section 2.2.2, “Taking incremental backups”: Perform the steps in this section every time you want to take an incremental back up of the changed blocks in a VDI.
- Section 2.2.3, “Restoring a VDI from exported changed block data”: Perform the steps in this section if you want to use your backed up data to restore a VDI to an earlier state.

2.2.1. Setting up changed block tracking

Before you can take incremental backups of a VDI using changed block tracking, you must first enable changed block tracking on the VDI and export a base snapshot. To set up changed block tracking for a VDI, complete the following steps

1. Use the Management API to establish a XenAPI session on the XenServer host:

```
import XenAPI
import shutil
import urllib3
import requests

session = XenAPI.xapi_local()
session.xenapi.login_with_password("<user>", "<password>", "<version>",
 "<originator>")
```

2. Optional: If you intend to create a new VM and new VDIs to restore your backed up data to, you must also export your VM metadata. Ensure that you export a copy of the VM metadata any time your VM properties change. This can be done by using HTTPS or by using the command line.

```
session_id = session._session
url = ("https://%s/export_metadata?session_id=%s&uuid=%s"
        "&export_snapshots=false"
        % (<xs_host>, session_id, <vm_uuid>))

with requests.Session() as session:
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
    request = session.get(url, verify=False, stream=True)
    with open(<export_path>, 'wb') as filehandle:
        shutil.copyfileobj(request.raw, filehandle)
    request.raise_for_status()
```

Where *<export_path>* is the location to save the VM metadata to.

The export URL includes the parameter `export_snapshots=false`. This parameter ensures that the snapshot history is not included in the VM metadata backup. The VM metadata is used to create a new VM and this snapshot history does not apply to the new VM.

If you intend to use your backed up data only to restore existing VDIs and VMs, you can skip this step.

3. Get a reference for the VDI you want to snapshot:

```
vdi_ref = session.xenapi.VDI.get_by_uuid("<vdi_uuid>")
```

4. Enable changed block tracking for the VDI:

```
session.xenapi.VDI.enable_cbt(<vdi_ref>)
```

For more information, see Chapter 4, *Using changed block tracking with a virtual disk image*.

5. Take a snapshot of the VDI:

```
base_snapshot_vdi_ref = session.xenapi.VDI.snapshot(<vdi_ref>)
```

This VDI snapshot is the base snapshot.

6. Export the base VDI snapshot to the backup location. This can be done by using HTTPS or by using the command line.

For example, at the xe command line run:

```
xe vdi-export uuid=<base-snapshot-vdi-uuid> filename=<name of export>
```

Or, in Python, you can use the following code:

```
session_id = session._session
url = ('https://%s/export_raw_vdi?session_id=%s&vdi=%s&format=raw'
       % (<xs_host>, session_id, <base_snapshot_vdi_uuid>))
with requests.Session() as http_session:
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
    request = http_session.get(url, verify=False, stream=True)
    with open(<export_path>, 'wb') as filehandle:
        shutil.copyfileobj(request.raw, filehandle)
    request.raise_for_status()
```

Where *<export_path>* is the location you want to write the exported VDI to.

7. Optional: For each VDI snapshot, delete the snapshot data, but retain the metadata:

   ```
   session.xenapi.VDI.data_destroy(<base_snapshot_vdi_ref>)
   ```

   This frees up space on the host or SR.

   For more information, see Chapter 5, *Deleting VDI snapshot data and retaining the snapshot metadata*.

2.2.2. Taking incremental backups

After taking the initial VDI snapshot and exporting all the data, the following steps can be repeated every time an incremental backup is taken of the VDI. These incremental backups export only the blocks that have changed since the previous snapshot was taken.

To take an incremental backup, complete the following steps:

1. Check that changed block tracking is enabled:

   ```
   is_cbt_enabled = session.xenapi.VDI.get_cbt_enabled(<vdi_ref>)
   ```

   If the value of `is_cbt_enabled` is not `true`, you must complete the steps in Section 2.2.1, "Setting up changed block tracking", before taking incremental backups. For more information, see Section 4.1, "Incremental backup sets".

   If changed block tracking is disabled and this is unexpected, this state might indicate that the host or SR has crashed since you last took an incremental backup or that a XenServer user has disabled changed block tracking.

2. Take a snapshot of the VDI:

   ```
   snapshot_vdi_ref = session.xenapi.VDI.snapshot(<vdi_ref>)
   ```

3. Compare this snapshot to a previous snapshot to find the changed blocks:

   ```
   bitmap =
    session.xenapi.VDI.list_changed_blocks(<base_snapshot_vdi_ref>, <snapshot_vdi_ref>)
   ```

   This call returns a base64-encoded bitmap that indicates which blocks have changed. For more information, see Chapter 6, *Get the list of blocks that changed between VDIs*.

4. Get details for a list of connections that can be used to use to access the VDI snapshot over the NBD protocol.

   ```
   connections = session.xenapi.VDI.get_nbd_info(<snapshot_vdi_ref>)
   ```

   This call returns a list of connection details that are specific to this session. Each set of connection details in the list contains a dictionary of the parameters required for an NBD client connection. For more information, see Section 7.1, "Getting NBD connection information for a VDI".

   Note:

   Ensure that this session with the host remains logged in until after you have finished reading from the network block device.

5. From your NBD client, complete the following steps to export the changed blocks to the backup location. For example, when using the Linux `nbd-client`:

   a. Connect to the NBD server.

   ```
   nbd-client <address> <port> -N <exportname> -cacertfile <cacert>
         -tlshostname <subject>
   ```

   - The *<address>*, *<port>*, *<exportname>*, and *<subject>* values passed as parameters to the connection command are the values returned by the `get_nbd_info` call.
   - The *<cacert>* is a file containing one or more trusted Certificate Authority certificates of which at least one has signed the NBD server's TLS certificate. That TLS certificate is included in the values returned by the `get_nbd_info` call. If the TLS certificate returned by the `get_nbd_info` call is self-signed, it can be used as the value of `cacert` here to authenticate itself.

   For more information about using these values, see Section 7.1, "Getting NBD connection information for a VDI".

   b. Read off the blocks that are marked as changed in the bitmap returned from step 3.

   c. Disconnect from the block device:

   ```
   nbd-client -d <block_device>
   ```

   d. Optional: We recommend that you retain the bitmap associated with each changed block export at your backup location.

   To complete the preceding steps, you can use any NBD client implementation that supports the "fixed newstyle" version of the NBD protocol. For more information, see Section 7.2, "Exporting the changed blocks using an NBD client".

6. Optional: On the host, delete the VDI snapshot, but retain the metadata:

   ```
   session.xenapi.vdi.data_destroy(<snapshot_vdi_ref>)
   ```

   This frees up space on the host or SR.

   For more information, see Chapter 5, *Deleting VDI snapshot data and retaining the snapshot metadata*.

2.2.3. Restoring a VDI from exported changed block data

When you want to use your incremental backups to restore or import data from a VDI, you cannot use individual exports of changed blocks to do this. You must first coalesce the exported changed blocks onto a base snapshot. Use this coalesced VDI to restore or import backed up data.

1. Create a coalesced VDI.

   For each set of exported changed blocks between the base snapshot and the snapshot you want to restore to, create a coalesced VDI from a previous base VDI and the subsequent set of exported changed blocks. Ensure that you apply sets of the changed blocks to the base VDI in the order that they were snapshotted.

   To create a coalesced VDI from a base VDI and the subsequent set of exported changed blocks, complete the following steps:

   a. Get the bitmap that was used in step 3 to derive this set of exported changed blocks.

   b. For each block in the VDI:

   - If the bitmap indicates that the block has changed, read the block data from the set of exported changed blocks and append that data to the coalesced VDI.
   - If the bitmap indicates that the block has not changed, read the block data from the base VDI and append that data to the coalesced VDI.

   c. Use the coalesced VDI as the base VDI for the next iteration of this step. Or, if you have reached the target snapshot level, use this coalesced VDI in the next step to restore a VDI in XenServer.

   For more information, see Chapter 8, *Coalescing changed blocks onto a base VDI*.

You can now use this coalesced VDI to either import backed up data into a new VDI or to restore an existing VDI.

2. Optional: Create a new VM and new VDI.

   You can create a new VM and new VDI to import the coalesced VDI into. However, if you intend to use the coalesced VDI to restore an existing VDI, you can skip this step.

   To create a new VM and new VDI, complete the following steps

   a. Create a new VDI:

   ```
   vdi_record = {
        "SR": <sr>,
        "virtual_size": <size>,
        "type": "user",
        "sharable": False,
        "read_only": False,
        "other_config": {},
        "name_label": "<name_label>"
    }
    vdi_ref = session.xenapi.VDI.create(vdi_record)
    vdi_uuid = session.xenapi.VDI.get_uuid(vdi_ref)
   ```

   Where <sr> is a reference to the SR that the original VDI was located on and <size> is the size of the original VDI.

   b. To create a new VM that uses the VDI created in the previous step, import the VM metadata associated with the snapshot level you are using to restore the VDI:

   ```
   vdi_string = "&vdi:%s=%s" % (<original_vdi_uuid>, <new_vdi_uuid>)

   task_ref = session.xenapi.task.create("import_vm", "Task to track vm import")

   url = ('https://%s/import_metadata?session_id=%s&task_id=%s%s'
          % (host, session._session, task_ref, vdi_string))

   with open(<vm_import_path>, 'r') as filehandle:
       urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
       with requests.Session() as http_session:
           request = http_session.put(url, filehandle, verify=False)
           request.raise_for_status()
   ```

   Where <vm_import_path> is the location of the VM metadata.

   The `vdi:` query parameter changes the VM from pointing to its original VDI to pointing to the new VDI created in the previous step. You might want to create multiple new VDIs. If you want to change multiple VDI references for your new VM, add a `vdi:` query parameter for each VDI to the import URL.

   The new VM is created from the imported metadata and its VDI reference is updated to point at the VDI created in the previous step. You can extract the reference to this new VM from the result of the task. For more information, see the samples on GitHub.

3. Import the coalesced VDI snapshot to the XenServer host at the UUID of the VDI you want to replace with the restored version. This VDI can be either an existing VDI or the VDI created in the previous step.

   In Python, you can use the following code:

```
session_id = session._session
url = ('https://%s/import_raw_vdi?session_id=%s&vdi=%s&format=%s'
       % (<xs_host>, session_id, <vdi_uuid>, 'raw'))
with open(<import_path>, 'r') as filehandle:
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
    with requests.Session() as http_session:
        request = http_session.put(url, filehandle, verify=False)
        request.raise_for_status()
```

Where *<vdi_uuid>* is the UUID of the VDI you want to overwrite with the restored data from the coalesced VDI and *<import_path>* is the location of the coalesced VDI.

Chapter 3. Enabling NBD connections on XenServer

XenServer acts as an NBD server and makes VDI snapshots available over NBD connections. However, to connect to XenServer over an NBD connection, you must enable NBD connections for one or more networks.

> **Important:**
>
> We recommend that you use a dedicated network for your NBD traffic.

By default, NBD connections are not enabled on any networks.

> **Note:**
>
> Networks associated with a XenServer pool that have NBD connections enabled must either all have the `nbd` purpose or all have the `insecure_nbd` purpose. You cannot have a mix of normal NBD networks (FORCEDTLS) and insecure NBD networks (NOTLS). To switch the purpose of all networks, you must first disable normal NBD connections on all networks before enabling either normal or insecure NBD connections on any networks.

3.1. Enabling an NBD connection for a network (FORCEDTLS mode)

We recommend that you use TLS in your NBD connections. When NBD connections with TLS are enabled, any NBD clients that attempt to connect to XenServer must use TLSv1.2. The NBD server runs in FORCEDTLS mode with the "fixed newstyle" NBD handshake. For more information, see the NBD protocol documentation.

To enable NBD connections with TLS, use the `purpose` parameter of the network. Set this parameter to include the value `nbd`. Ensure that you wait for the setting to propagate before attempting to use this network for NBD connections. The time it takes for the setting to propagate depends on your network and is at least 10 seconds. We recommend that you use a retry loop when making the NBD connection.

3.1.1. Examples

You can use any of our supported language bindings to enable NBD connections. The following examples show how to do it in Python and at the xe command line.

Python:

```
session.xenapi.network.add_purpose(<network_ref>, "nbd")
```

xe command line:

```
xe network-param-add param-name=purpose param-key=nbd uuid=<network-uuid>
```

3.2. Enabling an insecure NBD connection for a network (NOTLS mode)

We recommend that you do not enable insecure NBD connections. Instead use FORCEDTLS NBD connections. However, the ability to connect to the XenServer over an insecure NBD connection is provided for development and testing with the NBD server operating in NOTLS mode as described in the NBD protocol.

To enable insecure NBD connections, use the `purpose` parameter of the network. Set this parameter to include the value `insecure_nbd`. Ensure that you wait for the setting to propagate before attempting to use this network for NBD connections. The time it takes for the setting to propagate depends on your network and is at least 10 seconds. We recommend that you use a retry loop when making the NBD connection.

3.2.1. Examples

You can use any of our supported language bindings to enable insecure NBD connections. The following examples show how to do it in Python and at the xe command line.

Python:

```
session.xenapi.network.add_purpose(<network_ref>, "insecure_nbd")
```

xe command line:

```
xe network-param-add param-name=purpose param-key=insecure_nbd uuid=<network-uuid>
```

3.3. Disabling NBD connections for a network

To disable NBD connections for a network, remove the NBD values from the `purpose` parameter of the network.

3.3.1. Examples

You can use any of our supported language bindings to disable NBD connections. The following examples show how to do it in Python and at the xe command line.

Python:

```
session.xenapi.network.remove_purpose(<network_ref>, "nbd")
```

Or, for insecure NBD connections:

```
session.xenapi.network.remove_purpose(<network_ref>, "insecure_nbd")
```

xe command line:

```
xe network-param-remove param-name=purpose param-key=nbd uuid=<network-uuid>
```

Or, for insecure NBD connections:

```
xe network-param-remove param-name=purpose param-key=insecure_nbd uuid=<network-
uuid>
```

CITRIX

Chapter 4. Using changed block tracking with a virtual disk image

The changed block tracking capability can be enabled and disabled for individual virtual disk images (VDIs).

4.1. Incremental backup sets

When you enable changed block tracking for a VDI you start a new set of incremental backups for that VDI. The first action you must take when starting a set of incremental backups is to create a baseline snapshot and to backup its full data.

After you disable changed block tracking, or after changed block tracking is disabled by XenServer or a user, no further incremental backups can be added to this set. If changed block tracking is enabled again, you must take another baseline snapshot and start a new set of incremental backups.

You cannot compare VDI snapshots taken as part of one set of incremental backups with VDI snapshots taken as part of a different set of incremental backups. If you attempt to list the changed blocks between snapshots that are part of different sets, you get an error with the message “Source and target VDI are unrelated”.

You can use some or all of the data in previous incremental backup sets to create VDIs that you can use to restore the state of a VDI. For more information, see Chapter 8, *Coalescing changed blocks onto a base VDI*.

4.2. Enabling changed block tracking for a VDI

By default, changed block tracking is not enabled for a VDI. You can enable changed block tracking by using the `enable_cbt` call.

When changed block tracking is enabled for a VDI, additional log files are created on the SR to list the changes since the last backup. Blocks of 64 kB within the VDI are tracked and changes to these blocks recorded in the log layer.

The associated VM remains in the same state as before changed block tracking was enabled.

To enable changed block tracking, an SR must be attached, be writable, and have enough free space for the log files to be created on it. The associated VM can be in any state when changed block tracking is enabled or disabled. It is not required that the VM be offline.

Changed block tracking can only be enabled for a VDI that is one of the following types:

- user
- system

In addition, if the `VDI.on_boot` field is set to `reset`, you cannot enable changed block tracking for the VDI.

4.2.1. Examples

You can use any of our supported language bindings to enable changed block tracking for a VDI. The following examples show how to do it in Python and at the xe command line.

Python:

```
session.xenapi.VDI.enable_cbt(<vdi_ref>)
```

xe command line:

```
xe vdi-enable-cbt uuid=<vdi-uuid>
```

4.2.2. Errors

You might see the following errors when using this call:

VDI_MISSING
: The call cannot find the VDI.

  Check that the reference or UUID you are using to refer to the VDI is correct. Check that the VDI exists.

VDI_INCOMPATIBLE_TYPE
: The VDI is of a type that does not support changed block tracking.

  Check that the type of the VDI is `system` or `user`. You can use the `get_type` call to find out the type of a VDI. If your VDI is an incompatible type, you cannot enable changed block tracking. For more information, see Section 5.3, "Checking the type of a VDI or VDI snapshot".

VDI_ON_BOOT_MODE_INCOMPATIBLE_WITH_OPERATION
: The value of the `on_boot` field of the VDI is set to `reset`.

  Check the value of the `on_boot` field by using the `get_on_boot` call. If appropriate, you can use the `set_on_boot` call to change the value of this field to `persist`.

SR_NOT_ATTACHED, SR_HAS_NO_PBDS
: The call cannot find an attached SR.

  Check that there is an SR attached to the host and that the SR is writable. You cannot enable changed block tracking unless the host has access to an SR to which the changed block information can be written.

If you attempt to enable changed block tracking for a VDI that already has changed block tracking enabled, no error is thrown.

4.3. Disabling changed block tracking for a VDI

You can disable changed block tracking for a VDI by using the `disable_cbt` call.

When changed block tracking is disabled for a VDI, the active disks are detached and reattached without the log layer. The associated VM remains in the same state as before changed block tracking was disabled.

4.3.1. Examples

You can use any of our supported language bindings to disable changed block tracking for a VDI. The following examples show how to do it in Python and at the xe command line.

Python:

```
session.xenapi.VDI.disable_cbt(<vdi_ref>)
```

xe command line:

```
xe vdi-disable-cbt uuid=<vdi-uuid>
```

4.3.2. Errors

You might see the same sorts of errors for this call and you might for the `enable_cbt` call.

If you attempt to disable changed block tracking for a VDI that already has changed block tracking disabled, no error is thrown.

4.4. Checking whether changed block tracking is enabled

The value of the boolean `cbt_enabled` VDI field shows whether changed block tracking is enabled for that VDI. You can query the value of this field by using the `get_cbt_enabled` call.

A return value of `true` indicated that changed block tracking is enabled for this VDI.

4.4.1. Examples

You can use any of our supported languages to check whether a VDI has changed block tracking enabled. The following examples show how to do it in Python and at the xe command line.

Python:

```
is_cbt_enabled = session.xenapi.VDI.get_cbt_enabled(<vdi_ref>)
```

xe command line:

```
xe vdi-param-get param-name=cbt-enabled uuid=<vdi-uuid>
```

Chapter 5. Deleting VDI snapshot data and retaining the snapshot metadata

A VDI snapshot is made up of both data and metadata. The data is the full image of the disk at the time the snapshot was taken. The metadata includes the changed block tracking information.

After the snapshot data on the host has been exported to the backup location, you can use the `data_destroy` call to delete only the snapshot data and retain only the snapshot metadata on the host. This action converts the snapshot that is stored on the host or SR into a smaller metadata-only snapshot. The `type` field of the snapshot changes to be `cbt_metadata`.

Metadata-only snapshots are linked to the metadata-only snapshots that precede and follow them in time.

You can use the `data_destroy` call only for snapshots for VDIs that have changed block tracking enabled.

> **Note:**
>
> The API also provides a `destroy` call, which deletes both the data in the snapshot and the metadata in the snapshot.
>
> Do not use the `destroy` call to delete snapshots that are part of a set of changed block tracking backups unless you are sure that you no longer need the changed block tracking metadata.
>
> For example, use `destroy` to remove a metadata-only snapshot that is older than age allowed by your retention policy.

5.1. Examples

You can use any of our supported language bindings to delete the data in a snapshot and convert the snapshot to a metadata only snapshot. The following examples show how to do it in Python and at the xe command line.

Python:

```
session.xenapi.VDI.data_destroy(<snapshot_vdi_ref>)
```

xe command line:

```
xe vdi-data-destroy uuid=<snapshot_vdi_uuid>
```

5.2. Errors

You might see the following errors when using this call:

VDI_MISSING
: The call cannot find the VDI snapshot.

  Check that the reference or UUID you are using to refer to the VDI snapshot is correct. Check that the VDI exists.

VDI_NO_CBT_METADATA
: No changed block tracking metadata exists for this VDI snapshot.

  Check that changed block tracking is enabled for the VDI. You cannot use the `data_destroy` call on VDIs that do not have changed block tracking enabled. For more insformation, see Chapter 4, *Using changed block tracking with a virtual disk image*.

VDI_IN_USE
: The VDI snapshot is currently in use by another operation.

Check that the VDI snapshot is not being accessed by another client or operation. Check that the VDI is not attached to a VM.

5.3. Checking the type of a VDI or VDI snapshot

The value of the `type` VDI field shows what type of VDI or VDI snapshot an object is. The values this field can have are stored in the `vdi_type` enum. You can query the value of this field by using the `get_type` call.

A metadata-only VDI snapshot has the type `cbt_metadata`.

5.3.1. Examples

You can use any of our supported language bindings to query the VDI type of a VDI or VDI snapshot. The following examples show how to do it in Python and at the xe command line.

Python:

```
vdi_type = session.xenapi.VDI.get_type(<snapshot_vdi_ref>)
```

xe command line:

```
xe vdi-param-get param-name=type uuid=<snapshot_vdi_uuid>
```

CITRIX

Chapter 6. Get the list of blocks that changed between VDIs

You can use the `list_changed_blocks` call to get a list of the blocks that have changed between two VDIs. Both VDI snapshots must be taken after changed block tracking is enabled on the VDI.

This call takes as parameters references to two VDI snapshots:

- `VDI_from`: The earlier VDI snapshot.
- `VDI_to`: The later VDI snapshot. This VDI *cannot* be attached to a VM at the time this comparison is made.

This operation does not require the VM associated with the VDIs to be offline at the time the comparison is made.

The changed blocks are listed in a base64-encoded bitmap. Each bit in the bitmap indicates whether a 64 kB block in the VDI has been changed in comparison to an earlier snapshot. A bit set to 0 indicates that the block is the same. A bit set to 1 indicates that the block has changed.

The bit in the first position in the bitmap represents the first block in the VDI. For example, if the bitmap is 01100000, this indicates that the first block has not changed, the second and third blocks have changed, and all other blocks have not changed.

6.1. Examples

You can use any of our supported languages to get the bitmap that lists the changed blocks between two VDI snapshots. The following examples show how to do it in Python and at the xe command line.

Python:

```
bitmap =
 session.xenapi.VDI.list_changed_blocks(<previous_snapshot_vdi_ref>, <new_snapshot_vdi_ref>)
```

You can convert the base64-encoded bitmap this call returns into a human-readable string of 1s and 0s:

```
from bitstring import BitStream
import base64
data = BitStream(bytes=base64.b64decode(bitmap))
```

xe command line:

```
xe vdi-list-changed-blocks vdi-from-uuid=<previous_snapshot_vdi_uuid> vdi-to-
uuid=<new_snapshot_vdi_uuid>
```

6.2. Errors

You might see the following errors when using this call:

VDI_MISSING
: The call cannot find one or both of the VDI snapshots.

  Check that the reference or UUID you are using to refer to the VDI snapshot is correct. Check that the VDI snapshot exists.

VDI_IN_USE
: The VDI snapshot is currently in use by another operation.

  Check that the VDI snapshot is not being accessed by another client or operation. Check that the more recent VDI snapshot is not attached to a VM. The newer VDI in the comparison cannot be attached to a VM at the time of the comparison.

“Source and target VDI are unrelated”

The VDI snapshots are not linked by changed block metadata.

You can only list changed blocks between snapshots that are taken as part of the same set of incremental backups. For more information, see Section 4.1, “Incremental backup sets”.

CITRIX

Chapter 7. Export changed blocks over a network block device connection

XenServer runs an NBD server on the host that can make VDI snapshots accessible as a network block device to NBD clients. The NBD server listens on port 10809 and uses the "fixed newstyle" NBD protocol. For more information, see the NBD protocol documentation.

NBD connections must be enabled for one or more of the XenServer networks before you can export the changed blocks over NBD. For more information, see Chapter 3, *Enabling NBD connections on XenServer*.

7.1. Getting NBD connection information for a VDI

From a logged in XenAPI session, you can use the `get_nbd_info` call to get a list of connection details for a VDI snapshot made available as a network block device.

These connection details are specific to the session that creates them and the NBD client uses this logged in session when making its connection. Any set of connection details in the list can be used by the NBD client when accessing the VDI snapshot.

Each set of connection details in the list is provided as a dictionary containing the following information:

`address`
: The IP address (IPv4 or IPv6) of the NBD server.

`port`
: The TCP port to connect to the XenServer NBD server on.

`cert`
: The TLS certificate used by the NBD server encoded as a string in PEM format. When XenServer is configured to enable NBD connections in `FORCEDTLS` mode, the server presents this certificate during the TLS handshake and the NBD client must verify the server TLS certificate against this TLS certificate. For more information, see Section 7.2.1, “Verifying TLS certificates for NBD connections”.

`exportname`
: A token that the NBD client can use to request the export of a VDI from the NBD server. The NBD client provides the value of this token to the NBD server using the `NBD_OPT_EXPORT_NAME` option during the NBD option haggling phase of an NBD connection.

  This token contains a reference to a logged in XenAPI session. The XenAPI session must remain logged in for this token to continue to be valid. Because the token contains a reference to a XenAPI session, you must handle the token securely to prevent the session being hijacked.

  The format of this token is not guaranteed and might change in future releases of XenServer. Treat the export name as an opaque token.

`subject`
: A subject of the TLS certificate returned as the value of `cert`. This field is provided as a convenience.

7.1.1. Examples

You can use any of our supported languages to get the list of NBD connection details for a VDI snapshot. The following examples show how to do it in Python.

Python:

```
connection_list = session.xenapi.VDI.get_nbd_info(<snapshot_vdi_ref>)
```

This call requires a logged in XenAPI session that remains logged in while the VDI snapshot is accessed over NBD. This means that this command is not available at the xe command line.

7.1.2. Errors

You might see the following errors when using this call:

VDI_INCOMPATIBLE_TYPE
: The VDI is of a type that does not support being accessed as a network block device.

  Check that the type of the VDI is not `cbt_metadata`. You can use the `get_type` call to find out the type of a VDI. If your VDI is `cbt_metadata`, you cannot access it as a network block device. For more information, see Section 5.3, “Checking the type of a VDI or VDI snapshot”.

An empty list of connection details
: The VDI cannot be accessed.

  Check that the XenServer host that runs the NBD server has a PIF with an IP address.

  Check that you have at least one network in your pool with the purpose `nbd` or `insecure_nbd`. For more information, see Chapter 3, *Enabling NBD connections on XenServer*.

  Check that storage repository the VDI is on is attached to a host that is connected to one of the NBD-enabled networks.

7.2. Exporting the changed blocks using an NBD client

An NBD client running in the backup location can connect to the NBD server that runs on the XenServer host and access the VDI snapshot by using the provided connection details.

The NBD client that you use to connect to the XenServer NBD server can be any implementation that supports the “fixed newstyle” version of the NBD protocol.

When choosing or developing an NBD client implementation, consider the following requirements:

- The NBD client must support the “fixed newstyle” version of the NBD protocol. For more information, see the NBD protocol documentation.
- The NBD client must request an export name returned by the `get_nbd_info` call that corresponds to an existing logged in XenAPI session. The client makes this request by using the `NBD_OPT_EXPORT_NAME` option during the NBD option haggling phase of the NBD connection.
- The NBD client must verify the TLS certificate presented by the NBD server by using the information returned by the `get_nbd_info` call. For more information, see Section 7.2.1, “Verifying TLS certificates for NBD connections”.

  Note:

  If you are using the Linux upstream NBD client, a minimum version of 3.15 is required to support TLS.

  Note:

  XenServer supports up to 16 concurrent NBD connections.

After the NBD client has made a connection to the XenServer host and accessed the VDI snapshot, you can use the bitmap provided by the `list_changed_blocks` call to select which blocks to read. For more information, see Chapter 6, *Get the list of blocks that changed between VDIs*.

7.2.1. Verifying TLS certificates for NBD connections

When connecting to the NBD server using TLS, the NBD client must verify the certificate that the server presents as part of the TLS handshake.

We recommend that you use one of the following methods of verification depending on your NBD client implementation:

- Verify that the server certificate matches the certificate returned by the `get_nbd_info` call.
- Verify that the public key of the server certificate matches the public key of the certificate returned by the `get_nbd_info` call.

7.2.1.1. Alternative approach

As a less preferred option, it is possible for the NBD client to verify the certificate that the server presents during the TLS handshake by checking that the certificate meets all of the following criteria:

- It is signed by a trusted Certificate Authority
- It has an `Alternative Subject Name` (or, if absent, a `Subject`) that matches the subject returned by the `get_nbd_info` call.

Chapter 8. Coalescing changed blocks onto a base VDI

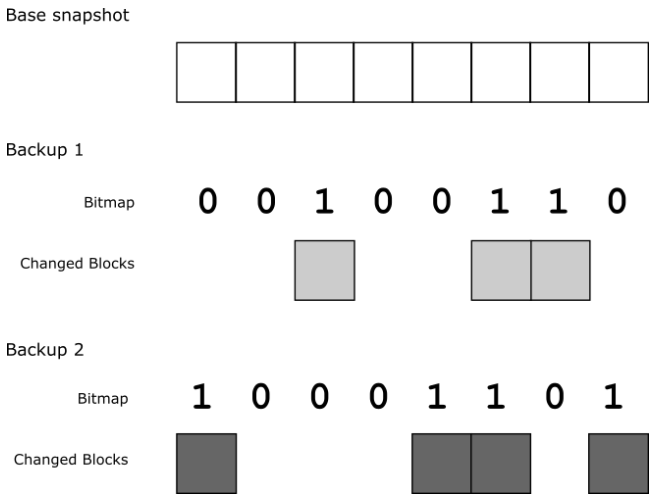
When using backed up data to restore the state of a VDI, you must import a full VDI into XenServer. You cannot import only the sets of changed blocks. To import incremental backups created with changed block tracking data into XenServer, you must first use these incremental backups to create a full VDI.

A set of incremental backups created with changed block tracking can be used to create a full VDI whose data is identical to the source VDI at the time an incremental backup was taken.

For more information about incremental backup sets, see Section 4.1, “Incremental backup sets”.

For example, you have a set of incremental backups that comprises:

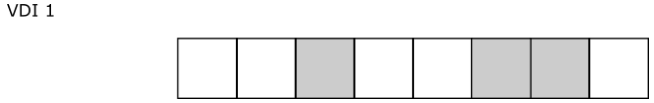
- A base snapshot that captures the data for the full VDI.
- Backup 1: The first incremental backup, which consists of a bitmap list of blocks changed since the base snapshot and the data for only those changed blocks.
- Backup 2: The second incremental backup, which consists of a bitmap list of blocks changed since backup 1 and the data for only those changed blocks.



If you want to restore a VDI on XenServer to the state it was at when backup 2 was taken, you must create a VDI that takes blocks from the base snapshot, changed blocks from backup 1, and changed blocks from backup 2. To do this, you can apply each set of changed blocks in sequence to the base snapshot of the VDI.

First build up a coalesced VDI by taking unchanged blocks from the base snapshot and changed blocks from those exported at backup 1. The bitmap list of changed blocks that was used to create backup 1 defines which blocks are changed.

After coalescing the base snapshot with the changed blocks exported at backup 1, you have a full VDI whose data is identical to that of the source VDI at the time backup 1 was taken. Call this coalesced VDI "VDI 1".



Next, use this coalesced VDI, VDI 1, to create another coalesced VDI by taking unchanged blocks from VDI 1 and changed blocks from those exported at backup 2. The bitmap list of changed blocks that was used to create backup 2 defines which blocks are changed.

After coalescing VDI 1 with the changed blocks exported at backup 2, you have a full VDI whose data is identical to that of the source VDI at the time backup 2 was taken. Call this coalesced VDI "VDI 2".

VDI 2

This coalesced VDI, VDI 2, can be used to restore the state of the VDI on XenServer at the time that a snapshot was taken for backup 2.

When creating a coalesced VDI, ensure that you work with your VDIs and changed blocks as binary.

Ensure that you verify the integrity of the backed up and restored VDIs. For example, you can do this by computing the checksums of the data.

8.1. Examples

The following example shows how to create a coalesced VDI. The example shows applying a single set of changed blocks to the base VDI snapshot. To apply multiple sets of changed blocks, you must repeat this process for each set of changed blocks in order from oldest to most recent, using the output from the previous iteration as the base VDI for the next iteration.

Python:

```
def write_changed_blocks_to_base_VDI(vdi_path, changed_block_path, bitmap_path,
 output_path):
    bitmap = open(bitmap_path, 'r')
    vdi = open(vdi_path, 'r+b')
    blocks = open(changed_block_path, 'r+b')
    combined_vdi = open(output_path, 'wb')

    try:
        bitmap_r = bitmap.read()
        cb_offset = 0
        for x in range(0, len(bitmap_r)):
            offset = x * changed_block_size
            if bitmap_r[x] == "1":
                blocks.seek(cb_offset)
                blocks_r = blocks.read(changed_block_size)
                combined_vdi.write(blocks_r)
                cb_offset += changed_block_size
            else:
                vdi.seek(offset)
                vdi_r = vdi.read(changed_block_size)
                combined_vdi.write(vdi_r)
```

CITRIX

Appendix A. Constraints

The following section lists advisories and constraints to consider when using changed block tracking.

- Changed block tracking is available only to customers with an Enterprise license for XenServer. If a customer without an Enterprise license attempts to use an incremental backup solution for XenServer that uses changed block tracking, they are prevented from enabling changed block tracking on new VDIs. However, if the customer has existing VDIs with changed block tracking enabled, they can still perform other changed block tracking actions on these VDIs.
- Changed block tracking information is lost on Storage XenMotion. If you attempt to migrate a VM that has VDIs with changed block tracking enabled, you are prevented from doing so. You must disable changed block tracking before Storage XenMotion is allowed.
- If a host or an SR crashes, XenServer disables changed block tracking for all VDIs on that host or SR. Before taking a VDI snapshot, we recommend that you check whether changed block tracking is enabled. If changed block tracking is disabled and this is not expected, this can indicate that a crash has occurred or that a XenServer user has disabled changed block tracking.

  To continue using changed block tracking, you must enable changed block tracking again and create a new baseline by taking a full VDI snapshot. Subsequent changed block tracking metadata uses this snapshot as a new baseline.

  The set of snapshots and changed block tracking data captured before the crash cannot be used as a baseline or comparison for any snapshots taken after the crash. However, the set of incremental backups taken before the crash can be used to create a VDI image to use to restore the VDI to a previous state.

  For more information, see Section 4.1, “Incremental backup sets”.
- XenServer supports a maximum of 16 concurrent NBD connections.

Appendix B. Additional Resources

The following resources provide additional information:

- GitHub repository of sample code
- Citrix XenServer Management API Guide
- Citrix XenServer Software Development Kit Guide
- Citrix XenServer documentation
- NBD protocol documentation
- nbd-client manpage