

Migrating only specific repositories from Nexus3 to Artifactory is explained in this document.

In order to do so, you would have to run the migration in multiple steps:

[Run the migration tool in multiple stages](#)

This document assumes the latest version of the tool is [jfrog-nexus-migrator-0.9.1.sh](#) (check for the latest version in [Migrating from Sonatype Nexus Repository Manager to Artifactory](#))

Let's go over these steps one by one:

1)Run

```
C/C++  
./jfrog-nexus-migrator-0.9.1.sh config
```

This will ask how to connect to Artifactory (and saves the details to /opt/migration/nexus-migrator/jfrog-cli.conf.v5) and Nexus (and saves the details to /opt/migration/nexus-migrator/migrationConfig.yaml) .

Note: In the migrationConfig.yaml the default threads is 12. You can monitor the migration and increase the threads if monitoring shows everything is fine in Nexus and Artifactory during the migration .

2) Run

```
C/C++  
./jfrog-nexus-migrator-0.9.1.sh getConfig
```

3) After running (1) and (2), a migration folder would be created (under the path where the script is located), you should navigate into the "migration" folder -> and then into the nexus-migrator folder.

```
varunm@varunm-nexus3:/opt/migration/nexus-migrator$ ls -ltr  
total 296  
-rw-r--r-- 1 root root 868 Apr 24 13:54 migrationConfig.yaml  
-rw-r--r-- 1 root root 266671 Apr 24 13:54 nexus.json  
-rw-r--r-- 1 root root 570 Apr 24 13:54 repositories.list  
-rw-r--r-- 1 root root 3 Apr 24 13:54 securityRegexMap.list  
-rw-r--r-- 1 root root 214 Apr 24 13:54 security.json  
drwxr-xr-x 6 root root 4096 Apr 24 14:11 artifact-cache  
-rw-r--r-- 1 root root 1484 Apr 24 14:11 repositories.yaml  
-rw-r--r-- 1 root root 15 Apr 24 14:11 assetmap.json  
-rw-r--r-- 1 root root 2 Apr 24 14:11 artifact.list  
varunm@varunm-nexus3:/opt/migration/nexus-migrator$
```

4) In the nexus-migrator folder, you should edit the file named "repositories.yaml" and delete any repository which you do not want to be created in Artifactory, an example from my end:

```
Repositories:
  localRepositories:
    docker-local:
      repoLayout: simple-default
      type: docker
    maven-releases:
      repoLayout: maven-2-default
      type: maven
      handleReleases: "true"
    maven-snapshots:
      repoLayout: maven-2-default
      type: maven
      handleReleases: "true"
    nuget-hosted:
      repoLayout: simple-default
      type: nuget
  remoteRepositories:
    maven-central:
      type: maven
      repoLayout: maven-2-default
      url: https://repo1.maven.org/maven2/
      handleReleases: "true"
    nuget.org-proxy:
      nuget:
        downloadContextPath: api/v2/package
        feedContextPath: api/v2
      type: nuget
      repoLayout: simple-default
      url: https://api.nuget.org/v3/index.json
  virtualRepositories:
    maven-public:
      type: maven
      repositories:
        - maven-releases
        - maven-snapshots
        - maven-central
      repoLayout: maven-2-default
    nuget-group:
      type: nuget
      repositories:
        - nuget-hosted
        - nuget.org-proxy
      repoLayout: simple-default
Blobstores:
- name: default
  type: File
  path: default
```

The picture above shows the repositories.yaml file after running the getConfig step.

If you wish to only migrate the "docker-local" repository, delete everything else in the repositories.yaml :

```
Repositories:
  localRepositories:
    docker-local:
      repoLayout: simple-default
      type: docker
Blobstores:
- name: default
  type: File
  path: default
```

Note: if you want the target repository in artifactory to be a different name change that in the /opt/migration/nexus-migrator/repositories.list where we map the source and target repos.

For example here is mapping of a source repo npm (in Nexus) which will be migrated to target repo npm (in Artifactory):

C/C++

```
- source: npm  
  target: npm
```

[Slack](#) , [slack](#) repositories.list reponame syntax,

```
repoName:  
  - source: <repo name in nexus>  
    target: <repo name to be created in artifactory>
```

repositories.yaml reponame syntax,

```
Repositories:  
  localRepositories:  
    <repo name in nexus>:  
      repoLayout: simple-default  
      type: generic
```

Note: the value of target (in repositories.list) is used to indicate the name in artifactory.

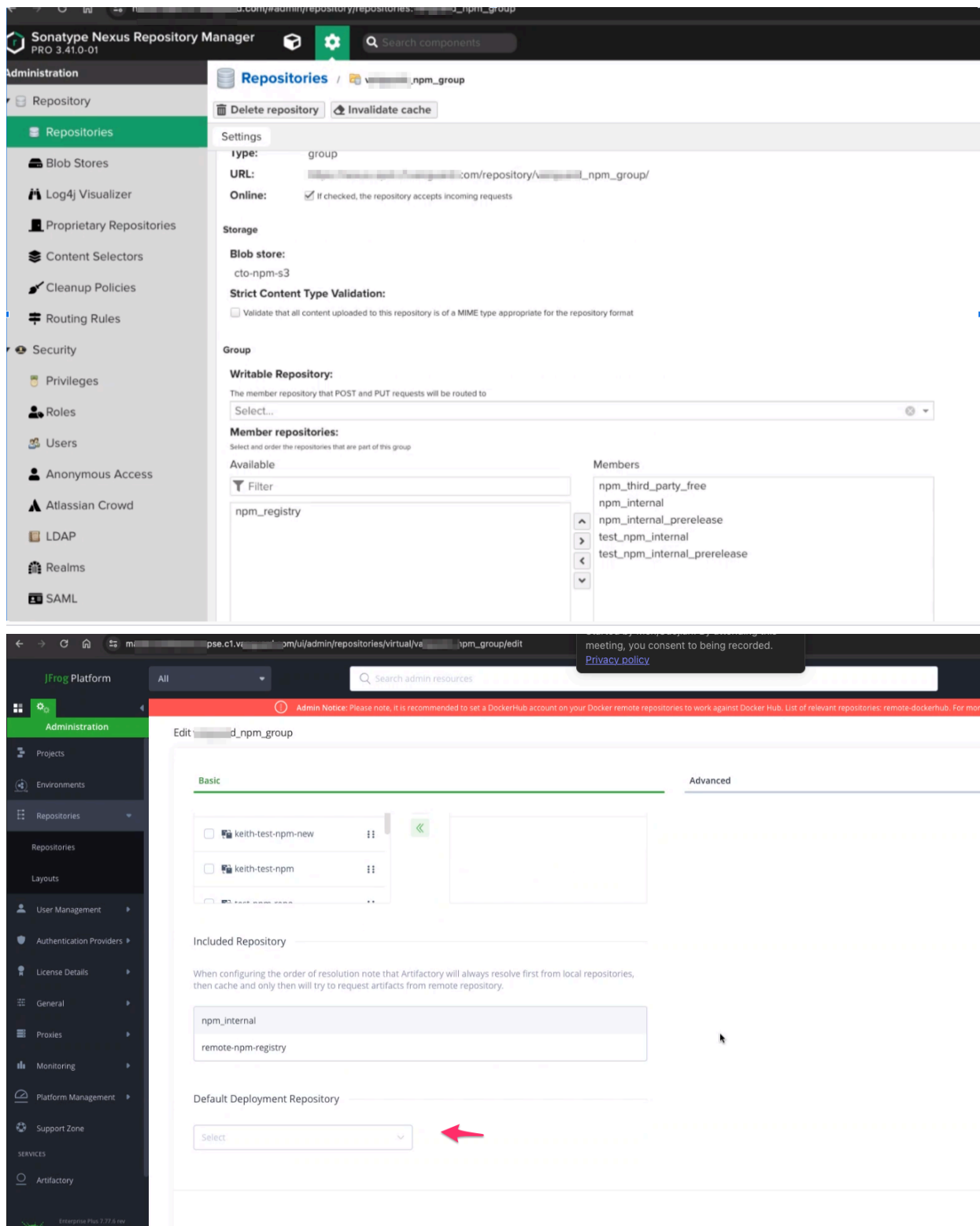
repositories.list file is the only place where new reponame of artifactory is used . In other files such as repositories.yaml the original repo name in Nexus should be used for reference.

5) After editing the repositories.yaml file, you may run the next step:

[./jfrog-nexus-migrator-0.9.1.sh](#) createRepo

This will only create the repositories specified in the [repositories.yaml](#) file in your target Artifactory instance (based on the repo name you want to use for the target repo in [repositories.list](#))

Note: The nexus group will map to a virtual repo in Artifactory. You can choose to configure the “Default Deployment Repository” in the virtual repo as that is not configured in the [repositories.yaml](#)



6) Now we can migrate the Artifacts to the specific repositories that you created with the createRepo step, in my example, I only created a "docker-local" repository, hence on this step -> I will only migrate the "docker-local" repository content:

```
./jfrog-nexus-migrator-0.9.1.sh migrateArtifact --repos="docker-local"
```

This is explained in detail in [Run the migration tool in multiple stages](#)

Note: The syntax for `--repos` with `ma` is,

```
./jfrog-nexus-migrator-0.9.2.sh ma --repos="<repo name in nexus>"
```

Another option is to use the asset file as explained in later steps .

7) Now, you may finish the migration by running the last step to migrate the users , permissions etc::

```
./jfrog-nexus-migrator-0.9.1.sh migrateSecurity
```

8) If you want to redo the migration for any repo (say repo named npm) or do a delta migration then change the status of the repo in

/opt/migration/nexus-migrator/repositories.yaml

```
C/C++
From
  npm:
    repoLayout: simple-default
    type: npm
    status: completed

to

  npm:
    repoLayout: simple-default
    type: npm
    status: pending
```

And do a transfer using the asset file which just has the files you want to transfer to the target npm repo:

Note: An asset file with name as `<repo-name>_assetmap.json` should be created in /opt/migration/nexus-migrator folder . See steps in [nexus_asset_file_creator/blob/master/README.md](#) . Here the **repo-name** (in the `<repo-name>_assetmap.json`) is name of your **target** repo (based on log below) .

C/C++

```
2024/05/21 18:56:10 [DEBUG] : Source repository name - npm, Target repository
name - sv-npm-local
2024/05/21 18:56:10 [DEBUG] : Getting the repository migration status for
sv-npm-local
2024/05/21 18:56:10 [INFO] : Starting artifact migration for sv-npm-local [5/6]
2024/05/21 18:56:10 [ERROR] :sv-npm-local_assetmap.json does not exist in
migration/nexus-migrator/ directory.
```

When running migrate with the "--repos" option internally we generate the asset file using the Nexus public API to get the file list -

<https://help.sonatype.com/en/assets-api.html>

For example , if you run the API against any npm repos with scoped packages in it we could see it.

example :

C/C++

```
curl -u admin:Jfrog@2022 -X GET
'http://localhost:8081/service/rest/v1/assets?repository=npm' >
npm_assetmap.json
```

You will get:

C/C++

```
{
  "items" : [ {
    "downloadUrl" :
"http://localhost:8081/repository/npm/@parcel/markdown-ansi",
    "path" : "@parcel/markdown-ansi",
    "id" : "bnBtOmQ4MzMwMDVkdNDljMzQ4ZThjZmUyOTNiM2EwYzZiOGJi",
    "repository" : "npm",
    "format" : "npm",
    "checksum" : {
      "sha1" : "f732ac870b0887f6788aa9ab43e611ad2358d2a3"
    },
    "contentType" : "application/json",
    "lastModified" : "2024-05-21T16:29:20.030+00:00",
```

```

    "lastDownloaded" : null,
    "uploader" : "admin",
    "uploaderIp" : "172.16.1.6",
    "fileSize" : 832,
    "blobCreated" : null,
    "blobCreated" : "2024-05-21T16:29:20.030+00:00"
  }, {
    "downloadUrl" :
"http://localhost:8081/repository/npm/@parcel/markdown-ansi/-/markdown-ansi-2.1
2.0.tgz",
    "path" : "@parcel/markdown-ansi/-/markdown-ansi-2.12.0.tgz",
    "id" : "bnBt0jI3YjkwNGJhNDMxZD1lNWVkdExZDYyZjRlZjI4NTIw",
    "repository" : "npm",
    "format" : "npm",
    "checksum" : {
      "sha1" : "a4301321fa784a28ba817e65e41432fe8b3b3192",
      "sha512" :
"599cf7af32fc9341f7591e2a4c75fa21cf03944b35ee478ef60b43e0c346c8c3506ea404bd0eb5
95625a207d2702d8047adbb448e213895a9d66b6fccf1fccef",
      "sha256" :
"7bdc6c5f379e9f5b257a3856a52be90be4e335018f805a156f6ad382d5d1b050",
      "md5" : "6d3c05d7040971e18ab1804f6edb9a9d"
    },
    "contentType" : "application/octet-stream",
    "lastModified" : "2024-05-21T16:29:20.051+00:00",
    "lastDownloaded" : null,
    "uploader" : "admin",
    "uploaderIp" : "172.16.1.6",
    "fileSize" : 2056,
    "blobCreated" : null,
    "blobCreated" : "2024-05-21T16:29:20.051+00:00",
    "npm" : {
      "name" : "@parcel/markdown-ansi",
      "version" : "2.12.0"
    }
  } ],
  "continuationToken" : null
}

```

But when running the `migrateasset` command using ``ma --use-existing-asset-file="true"`` the `<repo-name>_assetmap.json` the asset list should be in below format:

Unset

```
{
  "assets": [
    {
      "source": "SOURCE_REPO_NAME_IN_NEXUS/testgp/testad/v12/testad-v12.jar",
      "fileblobRef": ""
    },
    {
      "source": "SOURCE_REPO_NAME_IN_NEXUS/testgp/testad/v12/testad-v12.jar.sha256",
      "fileblobRef": ""
    },
    ...
  ]
}
```

Note: the "fileblobRef": "" and SOURCE_REPO_NAME_IN_NEXUS
Example:

C/C++

```
{
  "assets": [
    {
      "source": "npm/@parcel/markdown-ansi/-/markdown-ansi-2.9.3.tgz",
      "fileblobRef": "",
      "lastDownloaded": "null",
      "lastUpdated": "1693320901754",
      "createdDate": "1693320901754"
    }
  ]
}
```

You can run it as:

C/C++

```
./jfrog-nexus-migrator-0.9.2.sh ma --use-existing-asset-file="true" && tail -F migration.log
```

You should see "Success - 1" as shown in migration.log snippet below:

C/C++

```
2024/05/21 23:00:47 [DEBUG] : Source repository name - npm, Target repository
name - npm
2024/05/21 23:00:47 [DEBUG] : Getting the repository migration status for npm
2024/05/21 23:00:47 [INFO] : Starting artifact migration for npm [5/6]
2024/05/21 23:00:47 [INFO] : Reading assets from
migration/nexus-migrator/npm_assetmap.json
2024/05/21 23:00:47 [DEBUG] : Creating temporary directories for artifacts
migration
2024/05/21 23:00:47 [INFO] : Generated final artifact list for migration
2024/05/21 23:00:47 [INFO] : Starting artifacts migration
2024/05/21 23:00:47 [DEBUG] : Setting the number of threads for migration
2024/05/21 23:00:47 [INFO] : [Thread 3] [1/1] Fetching artifact :
npm/@parcel/markdown-ansi/-/markdown-ansi-2.9.3.tgz
2024/05/21 23:00:47 [INFO] : [Thread 3] [1/1] Deploying artifact :
npm/@parcel/markdown-ansi/-/markdown-ansi-2.9.3.tgz
2024/05/21 23:00:47 [INFO] : Artifact migration got completed for npm
```

Artifact Migration Report

npm : Failed - 0 | Success - 1

Troubleshooting1 (Nexus mlgrator tool) :

[298077](#) We're only planning to migrate a subset of repositories from Nexus, so it's easier for us to just generate repositories.yaml and repositories.list for those ones. I think getConfig is trying to generate configs for all the repos in Nexus, which we don't really need and that takes house if you have 1800+ repos in Nexus as per [getConfig_hanging_in_Vanguard](#)

With command :

C/C++

```
./jfrog-nexus-migrator-0.9.2.sh [jfrog-nexus-migrator-0.9.2.sh] ma
--repos="<repo name in nexus>"
```

It looks like if we use this, we might not need to create the assetmap.json. But when I tried it with a large repository, it seemed to take "longer than" the steps where we generate assetmap.json and use [./jfrog-nexus-migrator-0.9.2.sh](#) ma --use-existing-asset-file="true".

So, I'm wondering if there's a performance issue when we use --repos=<nexus-repo-name>? I don't mind creating assetmap.json and using it, even if it's a few extra steps, if it helps avoid any performance problems.

Let me clarify the steps involved in the migration process and provide instructions for your reference:

1. The "getConfig" step retrieves configuration information from Nexus Repository, including repository information (repositories.yaml and repositories.list) and security entities (security.json and securityRegexMap.list) such as users, groups, and permissions.
2. Since our focus is primarily on migrating repositories and their contents, we can disregard the security entity files.
3. Regarding the "use-existing-asset-file" option, please follow these steps:
 - Keep the asset file with the name "<repo-name>_assetmap.json" inside the "migration/nexus-migrator" folder.
 - Set the flag "--use-existing-asset-file="true"" while executing the "migrateArtifacts" step.

```
./jfrog-nexus-migrator-<version>.sh ma --use-existing-asset-file="true"
```

4. For each repository, create an assetmap.json file with the name "<repo-name>_assetmap.json" and follow the mentioned steps to run the "migrateArtifacts" step.

Troubleshoot2

For your convenience, please send all the following requested information to further investigate this:

1. Confirm the Nexus Version you are using
2. The migration/nexus-migrator/**repositories.yaml** file
3. The exact command that was used to trigger the Migration tool
4. The Artifactory [Support Bundle](#) from the last migration time frame attempt (I guess one hour around 2023/08/02 19:32:45 UTC) for further analysis
5. Resource allocated to Nexus server:
 - > RAM.
 - > CPU.
 - > Free memory on the nexus server.
 - > Storage.

-> What is the filestore type.

6. Please share the migration.log from the last time it failed

7.If possible you can increase the debug level of the tool using the following before

trying the migration again to see if it gives additional details as mentioned in

<https://jfrog.com/help/r/jfrog-installation-setup-documentation/edit-migration-configuration-optional>:

- a) modify the migrationConfig.yaml file and update the value of "logging: INFO" under migrator in the YAML file to "logging: DEBUG" i.e set the config.migrator.logging to DEBUG
- b) Make changes to the repositories.yaml file located in the working directory >/migration/nexus-migrator folder to limit migration to just this 1 repo which fails and rerun the test.
- c) Share with us the newly migration.log for further inspection.

Troubleshoot3

The getconfig command hangs.

Check if there is any connectivity issue to nexus from the instance where you run the next migrator script using APIs in [Nexus api fetch.txt](#)

Troubleshoot4

Q1:

The migration is being done in batches, and sometimes the migration tool can time out. I'd like to know if this batch process applies only to the ``ma --repos`` option. How does it work when migrating using ``assetmap.json``?

If we do have a large number of assets in the asset json file how long it will take before the timeout happens. Based on the current speed for 3706 npm assets will take around 5 mins to complete.

Please correct me if I misunderstand.

>> If you are running the migrateasset command using ``ma --use-existing-asset-file="true"`` then there should not be any timeouts. The Timeout can occur only when running migrateasset command using ``ma --repos`` option where internally the tool generates the asset file using the Nexus public API to get the file list - <https://help.sonatype.com/en/assets-api.html> and that may timeout. So with ``ma --use-existing-asset-file="true"`` option you should be fine.

Q2:

During the migration, I noticed that the newly created repository requires permissions for anonymous downloads and read-only access. We have a 'read_only' permission that is applicable to most repositories. However, during the migration, as we create a new repository, I added the repository to the permission manually via the User Interface (UserManagement->Permission->select the permission->edit repositories and add the repository from filter).

I would like to know is there is a JFrog CLI command or an option during the migration stage that can automatically add the new repository to the permission list. This would ensure that we do not miss this step during the migration.

>> To add a new repository to permission list use the [add_or_remove_repos_in_permission_target.sh](#) as mentioned in [add_or_remove_repos_in_permission_target.md](#)

Q3:

Also, would like to know the impact of “Client did not publish a checksum value. If you trust the uploaded artifact you can accept the actual checksum by clicking the 'Fix Checksum' button.”. Is it blocking any artifacts to download Or its just a warning.

>> It is only a warning shown in UI. Does not block artifact downloads from Artifactory.

Q4:

I ran the SQL query to find assets with missing checksums based on the KB article [Change Checksum Policy](#) . However, I haven't run the query to fix them yet. Is this the best approach to update a large number of assets after migration? I believe we should take a database backup before running this query and assess the potential impact of the error before proceeding with the database update.

>> Yes, running the update SQL after taking a DB backup is recommended.
