RAPIDS on JFrog Artifactory

Author: Akshit Arora et al.

Summary

This is a how-to guide for making RAPIDS libraries available to air-gapped data centers via JFrog Artifactory. Specifically covers:

1. Setting up remote repositories on Artifactory 7.x
2. Running RAPIDS conda installation.

Background

RAPIDS

RAPIDS enables Python users to take advantage of GPU acceleration in their workflow. It is a suite of open-source libraries and APIs that give you the ability to execute end-to-end data science and analytics pipelines entirely on GPUs. It utilizes NVIDIA CUDA primitives for low-level compute optimization and exposes GPU parallelism and high-bandwidth memory speed through user-friendly Python interfaces. It also includes support for multi GPU and multi-node deployments enabling vastly accelerated processing and training on much larger dataset sizes.

The core parts of the RAPIDS projects includes:

- cuDF: Dataframe manipulation library with pandas-like interface, accelerated on GPUs.
- cuML: Machine learning library with scikit-learn like interface, accelerated on GPUs.
- cuGraph: Graph analytics library with NetworkX like interface, accelerated on GPUs.
- RAPIDS also provides integration with deep learning libraries via array_interface / DLPack. It allows data stored in Apache Arrow format to be seamlessly pushed to deep learning frameworks.

Find out more about the latest RAPIDS features here: Two Years in a Snap — RAPIDS Release 0.16 | by Josh Patterson | RAPIDS AI | Nov, 2020.

JFrog Artifactory

JFrog Artifactory delivers end-to-end automation and management of your binaries and software artifacts. It is a scalable, universal binary repository manager that automatically manages your binaries and artifacts through the application development and delivery process that integrates with your DevOps tools and platforms. Artifactory is the industry standard for

universal binary and artifact repository solutions. It supports all major package types, including Conda, Python and other languages commonly used for data science. In addition, Artifactory supports Kubernetes (K8s), the de facto orchestration tool in the industry, for automating deployment, scaling, and management of microservices and containerized applications.

JFrog Artifactory's hybrid and configurable nature makes it ideal for air-gapped environments. It can run not only on any public cloud (AWS, GCP, Azure, etc) but in virtually any self-managed environment and integrates seamlessly with on-prem Enterprise-grade storage and database solutions.

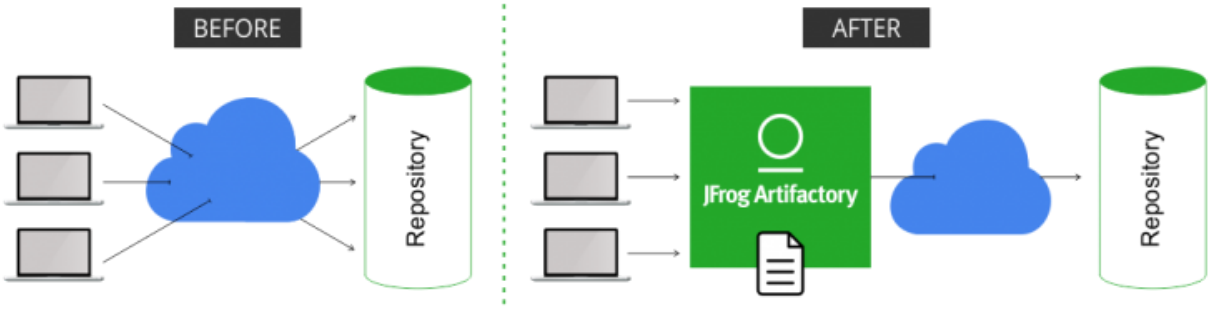
JFrog Artifactory's remote repository capabilities also play a vital role in implementing your air-gap environment. With a remote repository, you can proxy external sources (such as Anaconda), whitelist/blacklist packages and create other criteria for what is and is not allowed. The end-users (developers, CI servers) don't need to change the way they work because JFrog Artifactory behaves just like the original source and works natively with the tools the end-users are already using.

Since no two air-gapped environments are the same, JFrog Artifactory offers several implementation options. Here are two common implementations:

JFrog Artifactory in the DMZ

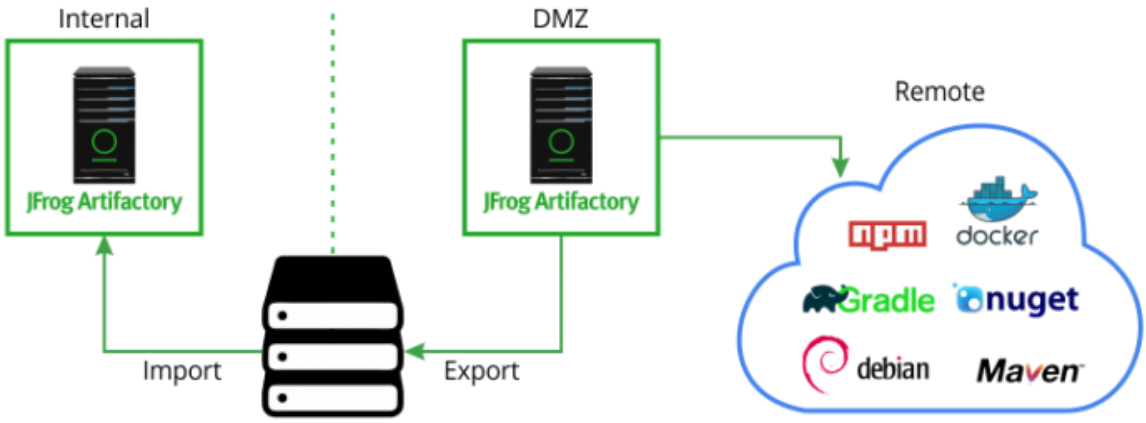
In this air-gapped implementation, you place JFrog Artifactory in the DMZ. Network rules prevent end-users from going out to the internet directly but allow JFrog Artifactory to reach out to the internet. When a user requests a package, if JFrog Artifactory does not have it stored locally (yet), it will reach out to the internet to the set of remote resources you have approved and download the requested package, caching it for future use and returning it to the user.

This is fairly straightforward to implement and guarantees no unauthorised external sources are used.



Fully air-gapped

The implementation above is air-gapped to the end-users but does allow the end-users to reach a machine that is not air-gapped. Certain extra secure environments restrict this type of communication and enforce a 100% air-gapped environment where no machine can reach any machine that has internet access. In these scenarios, you can use a two-step implementation with an offline sync. JFrog Artifactory's import/export capabilities allow you to download into one JFrog Artifactory with internet connection (and optionally manually approve) and then export to an archive that can then be placed on media to be imported into an internal JFrog Artifactory that does not have internet access. End-users would then pull from the internal JFrog Artifactory.



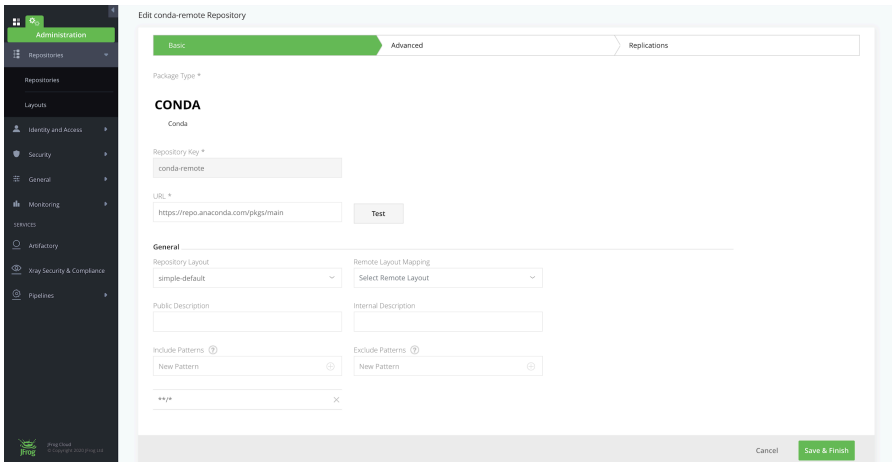
Setup

These instructions were written as a result of a specific end-user use case. The customer needed RAPIDS libraries but was restricted from accessing public sources since they were in an air gapped environment. Fortunately, the end-user was already a JFrog Artifactory user and used JFrog Artifactory in the DMZ. With this, the end-user simply upgraded to the appropriate version and was able to follow this setup to utilize the RAPIDS libraries in-house while still complying with corporate security requirements.

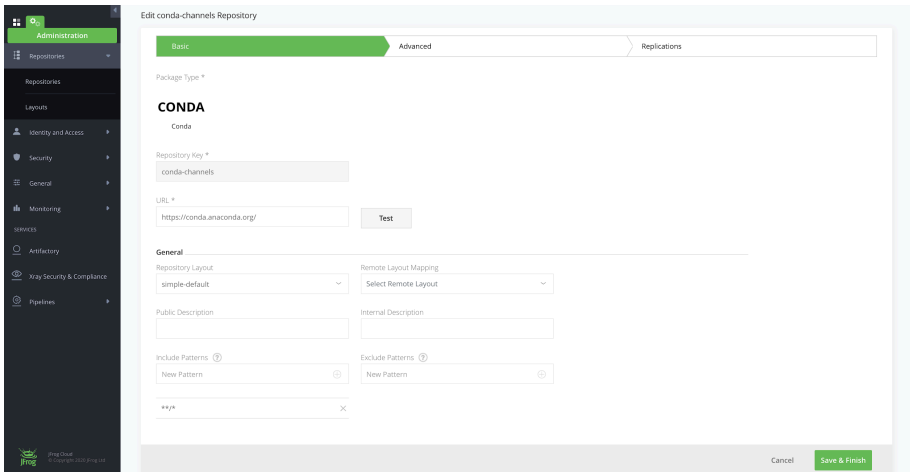
The following instructions have been verified on Artifactory 7.10.5 but will work on Artifactory 7.4.0+ or Artifactory 6.19.0+, though the UI menu navigation will be slightly different in 6.x. This setup assumes the JFrog Artifactory in the DMZ air gapped implementation.

1. Create a remote repository to the official Conda site

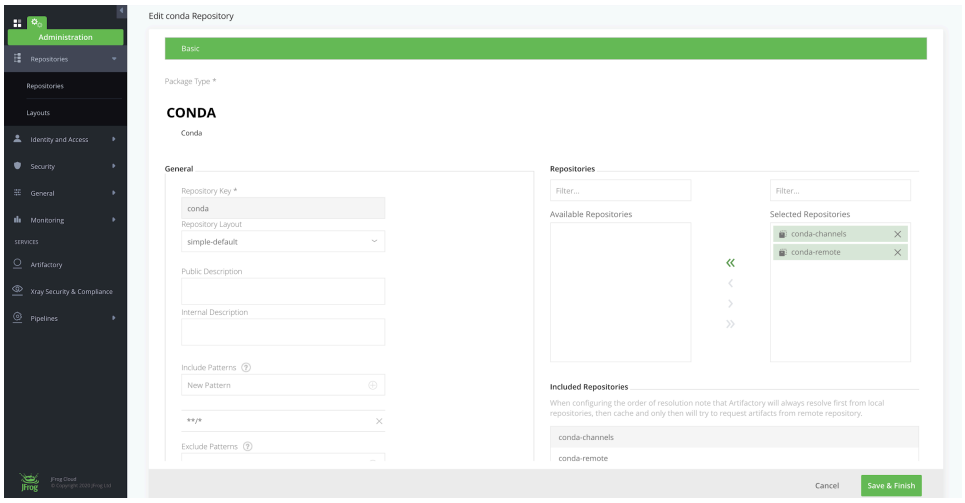
a. Using the UI, navigate to **Administration** -> **Repositories** -> **Repositories** -> **Remote** -> **New Remote Repository** and select **Conda** as the package type
b. Fill out the repository name, for example ‘conda-remote’
c. Fill out the URL: https://repo.anaconda.com/pkgs/main



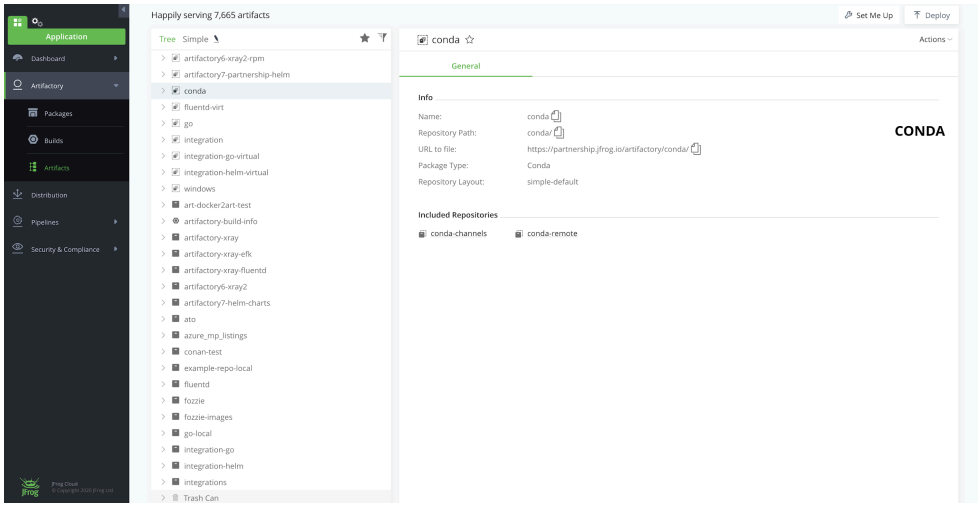
2. Create a second remote repository to the extra Conda channels
    a. Using the UI, navigate to **Administration** -> **Repositories** -> **Repositories** -> **Remote** -> **New Remote Repository** and select **Conda** as the package type
    b. Fill out the repository name, for example ‘conda-channels’
    c. Fill out the URL: https://conda.anaconda.org



3. Create a virtual repository that aggregates both remote repositories
    a. Using the UI, navigate to **Administration** -> **Repositories** -> **Repositories** -> **Virtual** -> **New Virtual Repository** and select **Conda** as the package type
    b. Fill out the repository name, for example ‘conda’
    c. Add both Conda remote repositories we created to the Selected Repositories



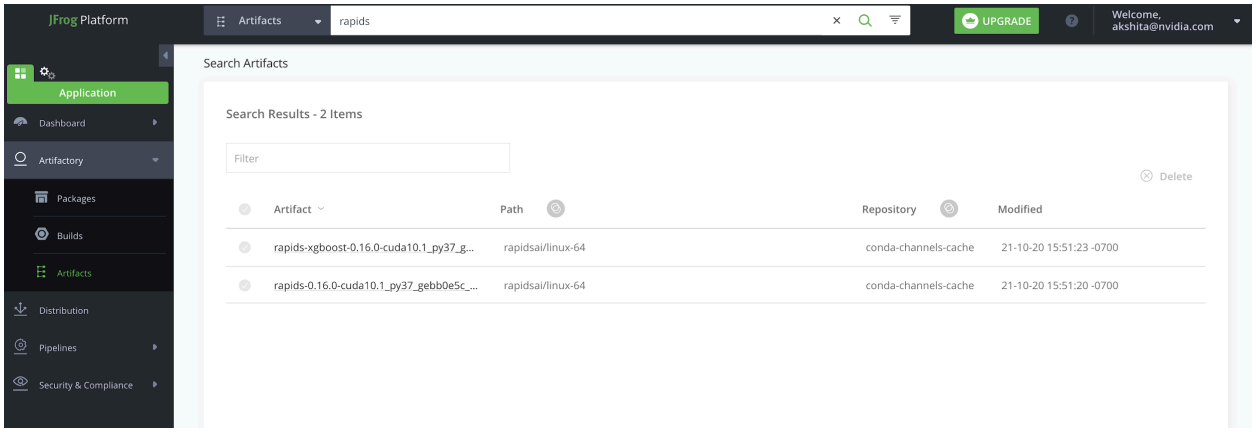
4. Set up your Conda client to work with Artifactory
    a. Using the UI, navigate to **Application** -> **Artifactory** -> **Artifacts,** click on the Conda virtual repository you created in step 3 and select **Set Me Up**
    b. Follow the **Set Me Up** instructions to configure your Conda client



5. Add the appropriate RAPIDS packages via RAPIDS release selector to confirm the set up is working correctly. Example:

```
conda create -n rapids-0.16 -c rapidsai -c nvidia -c conda-forge \
    -c defaults rapids=0.16 python=3.7 cudatoolkit=10.1
```

If this install is successful, your Artifactory instance will have the libraries cached for future use (you can see them in the artifacts browser, see screenshot below), speeding up any future build, and protecting you from the package you need being removed from the remote site or connections issues.



You're all set! Checkout [RAPIDS sample notebooks] to get started.

References

1. Explanation about why RAPIDS chose conda over pip. https://medium.com/rapids-ai/rapids-0-7-release-drops-pip-packages-47fc966e9472
2. RAPIDS 0.16 updates https://medium.com/rapids-ai/two-years-in-a-snap-rapids-0-16-ae797795a5c4
3. SaaS Artifactory: https://jfrog.com/artifactory/start-free/
4. https://www.jfrog.com/jira/browse/RTFACT-21908
5. https://www.jfrog.com/jira/browse/RTFACT-19267
6. Links to RAPIDS DLIs?

Authors

Akshit Arora, Subhan Ali, Arturo Aparicio, Mark Galpin, ...

Publish target: https://medium.com/rapids-ai