USING ARTIFACTORY TO MANAGE BINARIES ACROSS MULTI-SITE TOPOLOGIES

Publish: May 2023

TABLE OF CONTENTS

TABLE OF CONTENTS

INTRODUCTION

The days when applications were created by a small team of developers in one room are long past. Enterprise software development is now a highly collaborative endeavour of packages shared by intersecting teams across multiple sites spread across the world.
Global, multisite collaboration requires an architecture for managing software artifacts and deployable packages that is also global in scale.

COMPUTE LOCALLY, DEVELOP GLOBALLY

Performance over a network degrades steeply as sender and receiver are farther apart. For an organization with development teams across the world, it's critical to make sure that network issues like latency, bandwidth limitations, or connection outages don't harm productivity.
To always build at top speed, all the parts needed to build an application should be physically near to where the build work is done, including shared resources such as external and internal dependencies. When sharing these artifacts across multiple sites, they need to be available locally at each site, either within the local network (on-premises) or in the same region (cloud).
This requires multisite topologies that can locally synchronize a global set of shared artifact repositories, so that every build, at every site, can complete fast, without fail.

ARTIFACTORY FOR MULTISITE DEVELOPMENT

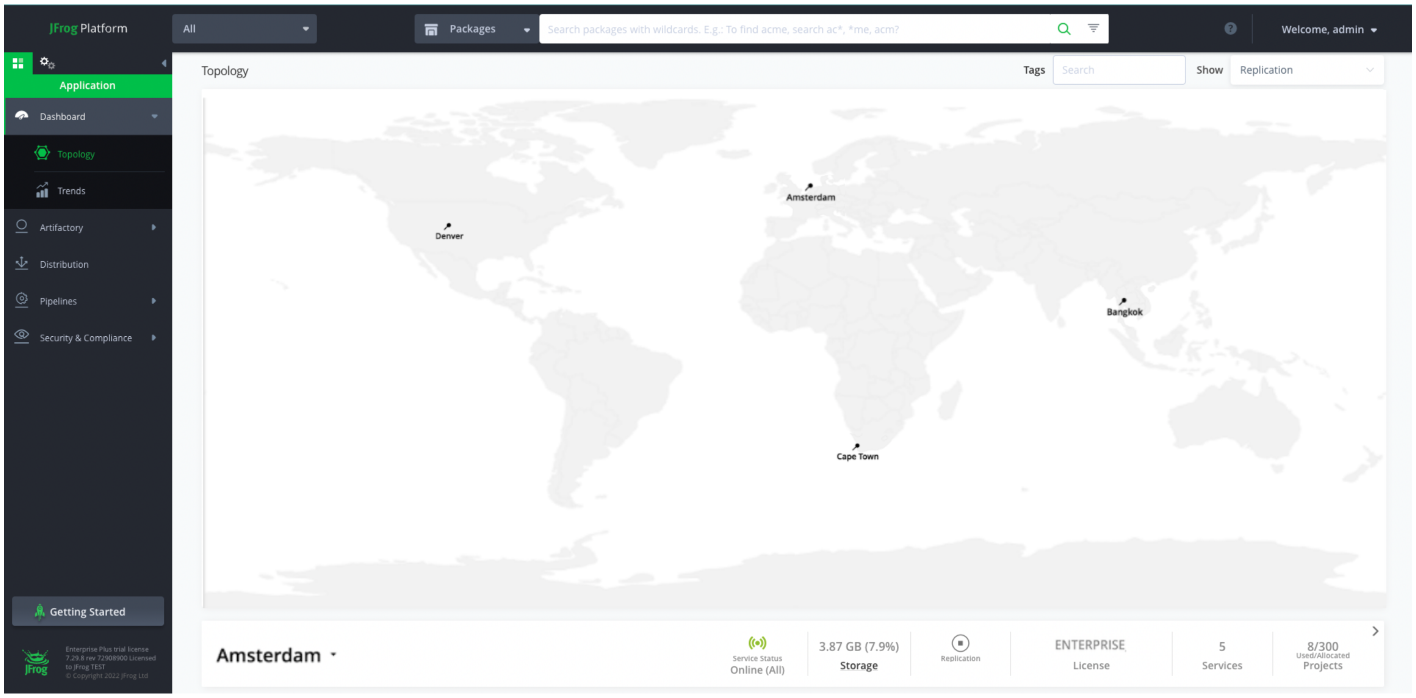
Artifactory's unique set of multisite capabilities ensure locality in any network topology:
Federation of repositories is JFrog's innovative technology for bidirectional mirroring between enterprise repositories in different installations of Artifactory. When an Artifactory repository is joined to a repository in Artifactory at another site, artifacts pushed to one are automatically duplicated in the other. When properly configured, this can fulfill a full mesh topology across multiple sites.
Replication of repositories offers a variety of topology choices, depending on your needs. These include both push and pull replication topologies, as well as scheduling strategies such as on-demand, on-schedule or event-based replication.
We'll describe the best practices in architecture and use for each of these capabilities in JFrog Artifactory Pro and Enterprise editions.

THE JFROG PLATFORM AND MISSION CONTROL

Artifactory is the core component that powers the JFrog DevOps Platform, a unified end-to-end platform for accelerating software development. Through a single pane of glass and central permissions management, you can manage the full lifecycle of your binaries from CI/CD automation, security scanning, to distribution to the edge.

JFrog Mission Control (available with an Enterprise subscription or higher) is the enterprise-scale solution to monitor and manage globally distributed instances of Artifactory from within the JFrog Platform single pane of glass. Replication can be configured by creating new repositories in multiple instances, and then configuring replication between them (one-to-one, or one-to-many).

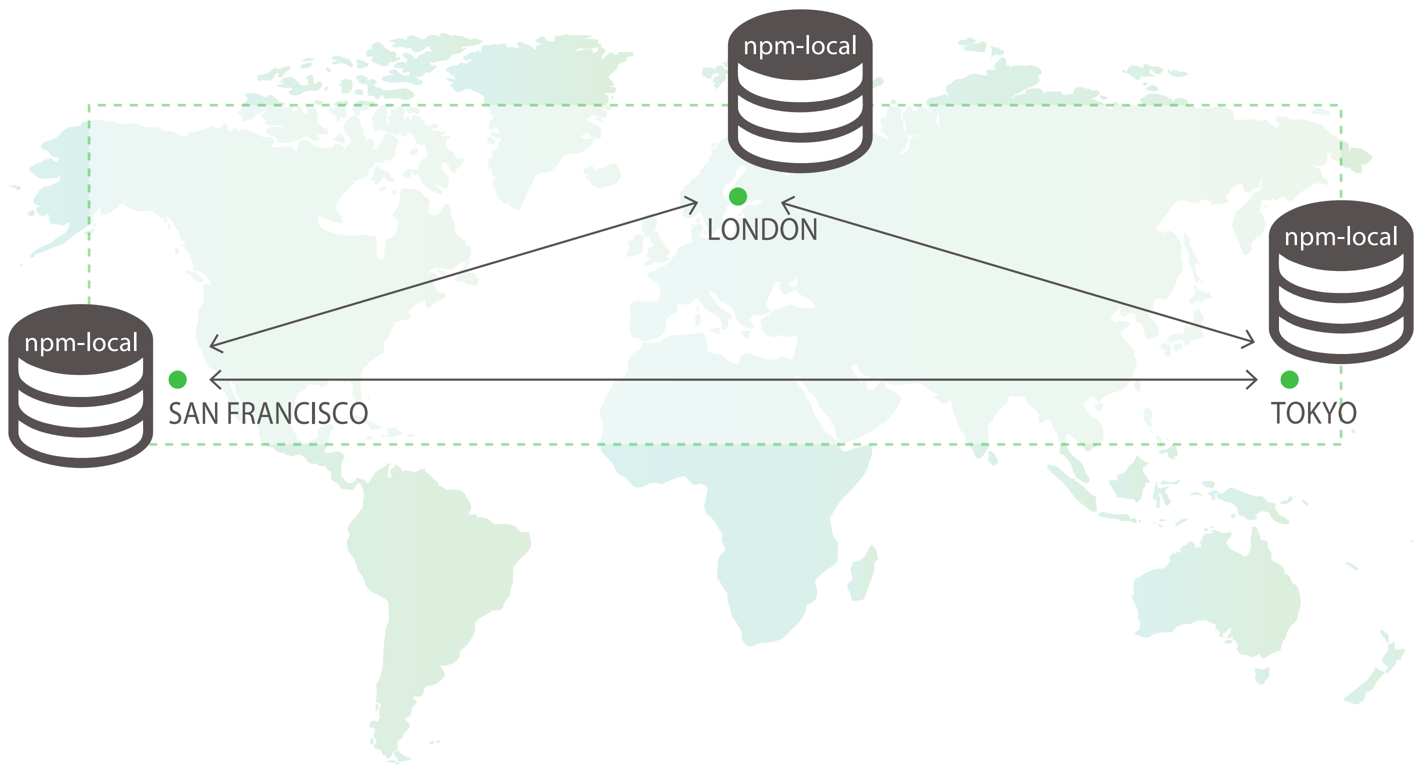
Mission Control displays a map of participating JFrog Platform Deployments.



Once a multisite topology is created, either by federation or replication of repositories, Mission Control will show the network of relationships between the JFrog deployments.

FEDERATING REPOSITORIES

Federated Repositories are JFrog's innovative, bidirectional mirroring technology that provides enterprise administrators an option that is easy to set up and maintain for multisite teams and projects. This repository mirroring technology continuously synchronizes a federated set of Artifactory repositories and their metadata across multiple sites.



With federated repositories, enterprises can:

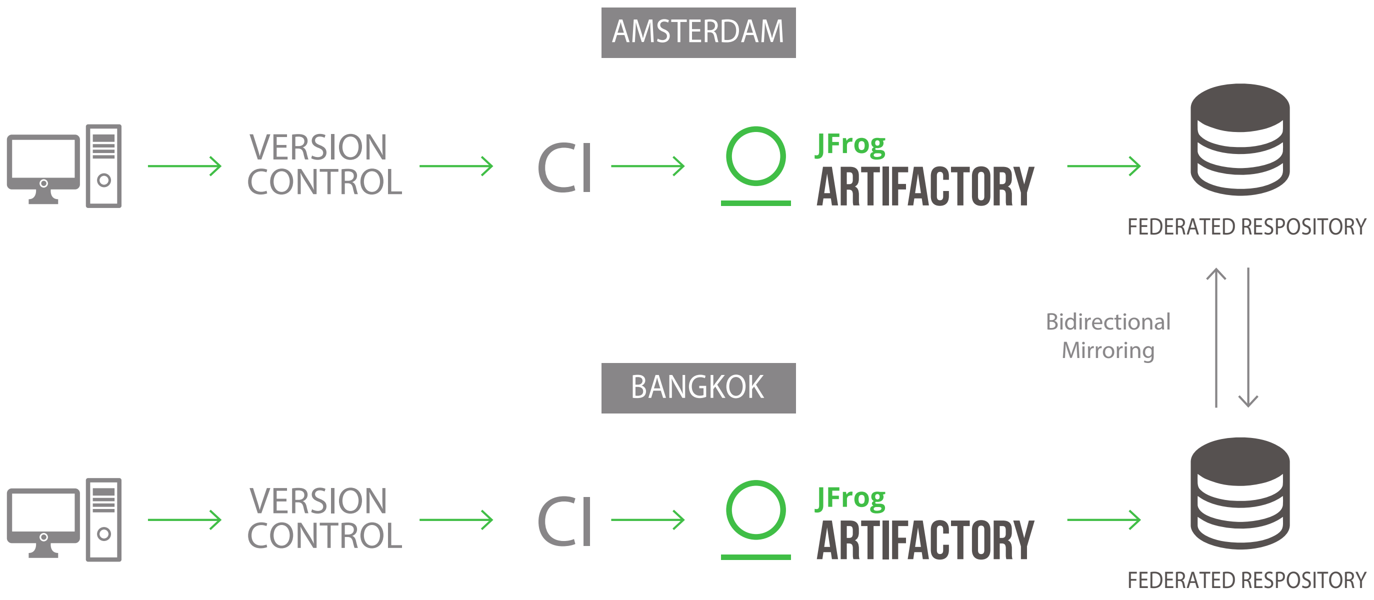
- Collaboratively develop software across geographically distributed teams
- Synchronize frequently updated binaries between sites securely and efficiently
- Scale development globally
  Ensure all shared artifacts and metadata are current to the latest version everywhere
- Replicate for disaster recovery

Federated repositories are supported only in JFrog Enterprise subscriptions or above.

SCALING GLOBALLY WITH FEDERATED REPOSITORIES

Through federation, a local repository in one Artifactory deployment (for example, in Amsterdam) can be logically joined to synchronize with a local repository in another Artifactory deployment (for example, in Bangkok). When joined in this way, artifacts and metadata pushed to the repository in Amsterdam will automatically be replicated to the repository in Bangkok. And because this federation is bidirectional, data pushed to the repository in Bangkok is replicated in Amsterdam as well.

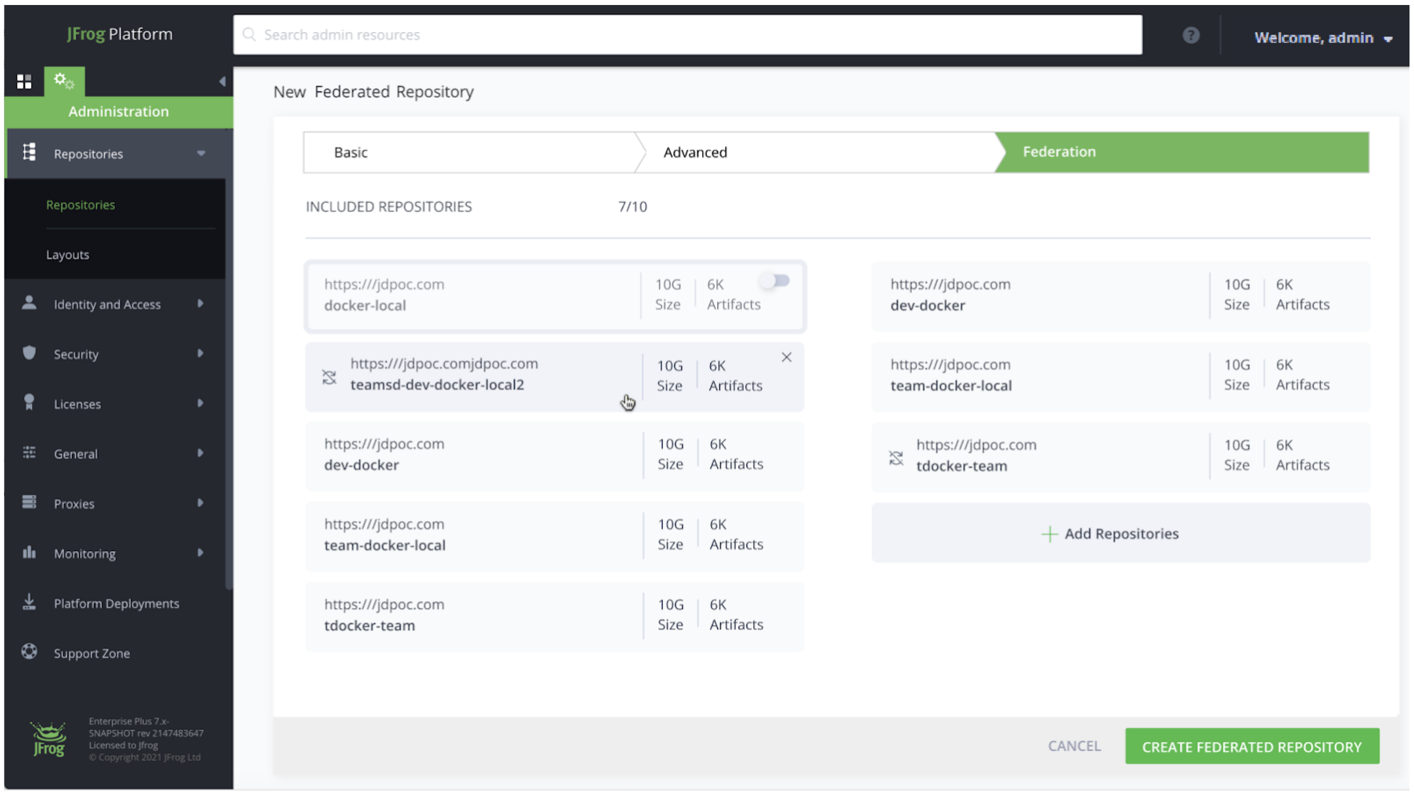
In this way, Artifactory repositories joined in a federation provide each of these sites a unified, locally accessible repository of shared global data.



EASY ADMINISTRATION

With a JFrog federated repository, an authorized administrator at one site can create and join local repositories at several JFrog Platform deployments into a single, multisite federation. There's no need to coordinate setup between platform administrators at different physical sites across time zones to create your federation topology.

Setup and maintenance can be quickly and easily accomplished from any affiliated physical site: all changes to configuration and repository settings are synchronized automatically across all federation members.



INCLUDED REPOSITORIES 7/10

https:///jdpoc.com docker-local 10G Size 6K Artifacts
https:///jdpoc.comjdpoc.com teamsd-dev-docker-local2 10G Size 6K Artifacts
https:///jdpoc.com dev-docker 10G Size 6K Artifacts
https:///jdpoc.com team-docker-local 10G Size 6K Artifacts
https:///jdpoc.com tdocker-team 10G Size 6K Artifacts
https:///jdpoc.com dev-docker 10G Size 6K Artifacts
https:///jdpoc.com team-docker-local 10G Size 6K Artifacts
https:///jdpoc.com tdocker-team 10G Size 6K Artifacts

Administrator View

SCALABLE FOR THE ENTERPRISE

You can federate repositories among any of the 30 repository types supported in Artifactory's universal package management. All repositories in a federation must be of the same type, of course, but you can create multiple federations for different package types. For example, you can have a federation of npm repositories, another of Maven repos, and another for Docker.

Any repository federation can include up to 10 members (local repositories in different JFrog Platform deployments (JPDs) at remote sites) of the same repository type, providing broad geographical coverage.

SECURE CIRCLE OF TRUST

Federated repositories use binary provider tokens to establish a circle of trust among members without having to set up certificates at each site. JFrog Mission Control automatically enables all JPDs for secure federation, or you can manually specify secure JFrog Platform URLs.

Administrators at each site can enable or restrict access to federated repositories by their own users through the permission groups managed on their own JPD.

HOW TO IMPLEMENT FEDERATION TOPOLOGIES

To start working with Federated repositories, you will first need to set up cross-instance authentication (Circle-of-Trust) between the JFrog Platform deployments through a shared public certificate among all participating instances. Once this is configured, the platform administrator may perform CRUD (Create, Remove, Update and Delete) actions on the repositories based on their predefined set of permissions.
All the actions performed on one of the Federated repositories will automatically be synchronized and reflected on all of the Federated members.
It's important to understand that, within a federation, bidirectional mirroring of artifacts and metadata only occurs between repositories that have declared a direct connection between them. Mirroring does not automatically cascade to other, secondarily connected repositories.

For example, consider an multisite example with these four JFrog Platform deployments:

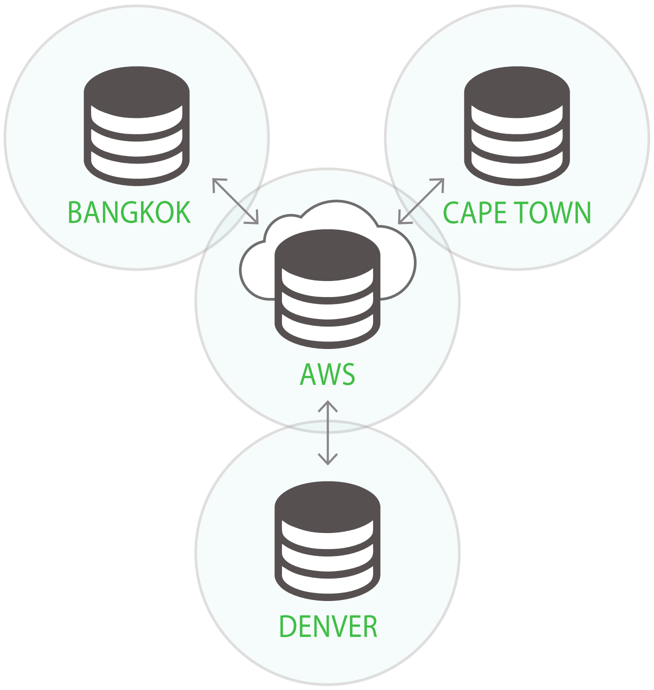
A cloud instance on Amazon Web Services (AWS)
An on-prem site at a branch development office in Bangkok
• An on-prem site at a corporate office in Cape Town
• An on-prem site at the main data center in Denver
•

Let's look at some example topologies to see how this works.

STAR TOPOLOGY

In a star topology, a local repository in a JPD acts as a central hub to other JPD repositories in the federation. The hub JPD repository shares its artifacts and metata with everyone in the federation, and everyone shares theirs with the hub. However, the other sites do not share with each other.

In this example, we configure a federation from the cloud instance on AWS as the central hub to connect to the affiliate sites. No federated repository configuration takes place at the three on-prem sites.



So in this example:

- An artifact pushed to the repository in AWS will be mirrored in Bangkok, Cape Town, and Denver
  An artifact pushed to the repository in Bangkok will be mirrored only on AWS.
- An artifact pushed to the repository in Cape Town will be mirrored only on AWS
- An artifact pushed to the repository in Denver will be mirrored only on AWS

	AWS	BANGKOK	CAPE TOWN	DENVER
AWS	✓	✓	✓	✓
BANGKOK	✓	✓	✗	✗
CAPE TOWN	✓	✗	✓	✗
DENVER	✓	✗	✗	✓

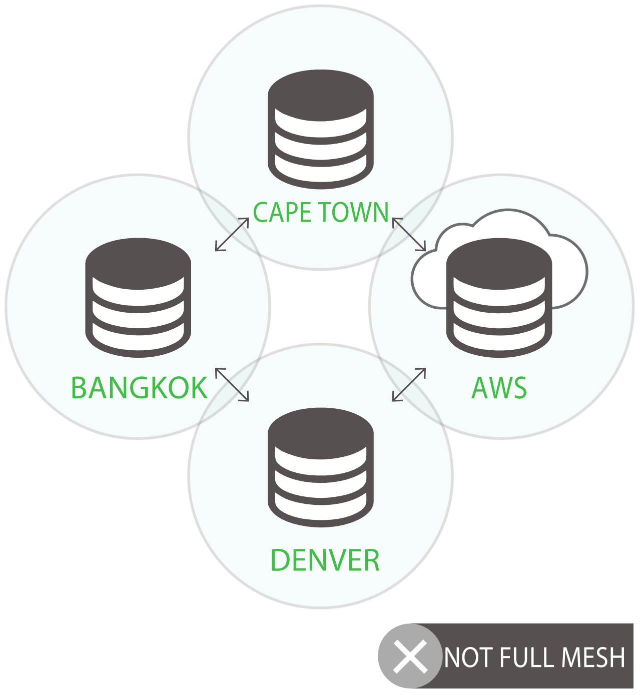
As can be seen, there are significant limitations to this topology. For instance, artifacts created in Bangkok will not be available locally to developers in Cape Town(although all sites can access all artifacts through the cloud instance on AWS).

This type of topology may be useful in certain situations, such as when most artifacts and metadata are expected to be produced at a central site, but need to be available locally to affiliate sites.

FULL MESH TOPOLOGY

In a full mesh topology, all artifacts and metadata in the federation are mirrored in local repositories at all sites. All users at all sites have access to all data according to their locally administered permissions, with minimal network latency.

To accomplish this, platform administrators must be careful to be complete in their federation connections.
For instance, the admins in our example organization might mistakenly configure a federation as shown in this diagram:

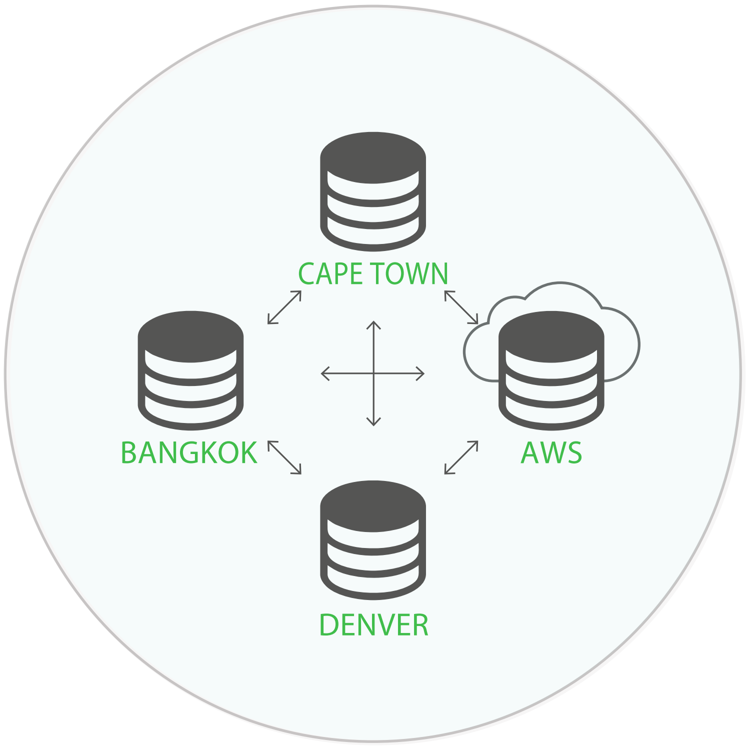


While this topology diagram may appear to create a full mesh network through this circular set of connections, it does not. Bidirectional mirroring of artifacts and metadata only occurs between repositories that are directly connected in the federation.

In this configuration, each site will mirror only two of the others, not all three:

	AWS	BANGKOK	CAPE TOWN	DENVER
AWS	✓	✓	✓	✗
BANGKOK	✓	✓	✗	✓
CAPE TOWN	✓	✗	✓	✓
DENVER	✗	✓	✓	✓

To correctly configure a federated repository for full mesh, each site must be configured to directly federate with the other three.



✓ FULL MESH

In this full mesh configuration, each site will mirror all others:

	AWS	BANGKOK	CAPE TOWN	DENVER
AWS	✓	✓	✓	✓
BANGKOK	✓	✓	✓	✓
CAPE TOWN	✓	✓	✓	✓
DENVER	✓	✓	✓	✓

REPLICATING REPOSITORIES

Replication of repositories can be more complicated to configure and administer than federated repositories, but they may be a better fit in some circumstances:

- You do not wish to synchronize repositories bidirectionally.
- You prefer to synchronize repositories on a scheduled basis.
- You prefer to maintain pull (on-demand) relationships between repositories.
- You only have a Pro-level (not Enterprise) subscription.

To administer replication of repositories, you will need to understand some key Artifactory concepts:

On-demand proxy is the default behavior of all remote repositories, regardless of whether you are proxying another node under control of your organization, or one that belongs to a 3rd party. When a job asks for an artifact from an on-demand remote repository, Artifactory will download this file and cache it for future use. You can suppress this behavior by selecting the Offline button in the repository configuration. In this case Artifactory will only provide remote artifacts that have already been cached.

A remote repository serves as a caching proxy for a repository managed at a remote site such as JCenter or Maven Central. Artifacts are stored and updated in remote repositories according to various configuration parameters that control the caching and proxying behavior.

Learn more >

Replicating artifacts between sites can rely on the on-demand proxy implemented by remote repositories or the different replication options implemented with local repositories.

Local repositories are physical, locally- managed repositories into which you can deploy artifacts. Typically, these are used to deploy internal and external releases as well as development builds, but they can also be used to store binaries that are not widely available on public repositories such as 3rd on public repositories such as 3rd local repositories, all of your internal resources can be made available from a single access point across your organization from one common URL.

Learn more >

Artifactory supports two primary modes of replication: Push and Pull replication. Each mode can be triggered in two ways, either on a regular schedule or by events.

PUSH REPLICATION

Push replication is used to synchronize local repositories, and is implemented by the Artifactory server on the near end invoking a synchronization of artifacts to the far end.
Push replication is useful when an artifact producer wants to distribute their artifact to other sites, which will use this artifact as a dependency.

Other than rare exceptions, a user should never have “write” access to the far end, and there should be only one master site with other sites slaved to it.
There are two ways to invoke a push replication: Scheduled and Event-Based.

SCHEDULED

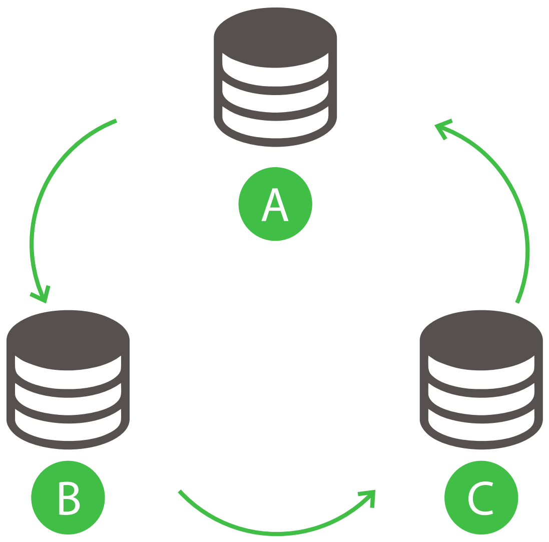
Pushes are scheduled asynchronously at regular intervals using a Cron expression that determines when the next replication will be triggered. Even if the plan is to use an event-based replication, the Cron expression is still required. The scheduled replication will serve as a backup for the event-driven replication, ensuring that all artifacts are synced, even if the event-driven replication of one of the artifacts failed for some reason, for example a network error.
Because of the checksum-based nature of Artifactory’s storage and replication mechanism, no artifacts will be transferred if they already exist on the other side, even under a different name or path, so no harm is done when you configure overlapped replications such as both event-driven AND scheduled.

EVENT-BASED

Pushes occur nearly in real-time since each create, copy, move or delete of an artifact and its metadata is immediately propagated to the far end.

Artifactory supports event-based push replication from one repository to another single repository on the far end. Artifactory Enterprise supports multi-push which allows you to replicate a repository to multiple JFrog deployments simultaneously. The alternative is setting up replication chains, which means that each node will propagate artifacts to another node, creating a serial chain of deployments. This solution is more complicated to setup and increases the inconsistency between the deployments, and therefore does not comply with best practice for repository replication.
There is also the risk of creating a replication loop (A pushes to B, B pushes to C, C pushes back to A) which can have disastrous effects on your system and must be strictly avoided. If you need to replicate to multiple repositories, and don’t have Artifactory Enterprise edition, pull replication is recommended.

To ensure that all changes on the near end are propagated to the far end, a scheduled replication is mandatory in conjunction with event based replication.



REPLICATION LOOP TO BE STRICTLY AVOIDED

PULL REPLICATION

Pull replication is a scheduled pre–population of a remote repository cache. It's useful for remote sites that need to get the artifacts they require not by the first request, as in the normal operation of remote repository, but even earlier. For example, artifacts produced on one day in the remote site can be replicated during the night and sit in cache waiting to be consumed as dependencies the next morning.

Pull replication is invoked by a remote repository in two ways: Scheduled and Event-Based.

The remote repository invoking the replication from the far end can pull artifacts from any type of repository - local, remote or virtual.
Synchronized deletion can be configured in both push and pull replication repositories. This is optional and not enabled by default. On-demand proxy replication does not support deletions.

SCHEDULED

Pull replication is invoked through a schedule that's defined by a Cron expression to synchronize repositories at regular intervals.

EVENT-BASED

Pull replication is invoked by a remote repository from the far end and can pull artifacts from any type of repository - local, remote or virtual of the source Artifactory server. Pulls occur nearly in real-time since each create, copy, move or delete of an artifact and its metadata is immediately propagated to the far end from the source Artifactory server.

When an event triggers a replication, artifacts that are in the process of being replicated to the far end are already available for use through Artifactory's remote-proxy mechanism of remote repositories, even if the replication process is not yet complete. As a result, requests for these artifacts will not fail.

With event-based pull replication, many target servers can pull from the same source server efficiently implementing a one-to-many replication, thus reducing the traffic on target servers since they do not have to pass on artifacts in a replication chain.

Event-based pull replication allows leveling the network throughput bursts involved with scheduled replications. It also reduces the need for computing resources on the source node, distributing the replication computation logic to the target nodes.
Support for event-based pull replication is available only with the Artifactory Enterprise edition.
To ensure that all changes on the near end are propagated to the far end, a scheduled replication is mandatory in conjunction with event based replication.

COMPARING REPLICATION TYPES

The following tables summarize the difference between the different replication options and triggering methods:

PUSH REPLICATION	PULL REPLICATION
Network Topology The source initiates the network communication with the target.	Network Topology The target initiates the network communication with the source (For event-based pull, two-way communication is required).
Replication configuration at the source. Centralized control.	Replication configuration at the remote. Distributed control.
Replicated artifacts are indexed on the target repository, keeping the repository locally consistent.	Index is based on SOURCE repository, but in an offline scenario it may not be locally consistent.
	Artifacts are available immediately even before synchronization completes.
	Reduces compute overhead by not recalculating index on remotes.

EVENT-BASED REPLICATION	SCHEDULED (CRON) REPLICATION	ON-DEMAND PROXY
Shortest time to Global consistency. Minimizes time repositories are not synchronized.	Replication traffic managed to low traffic periods.	Only artifacts that are resolved on the far end are replicated (and cached).
The replication traffic is spread out. No 'lump jobs'.	Guarantees full synchronization in case of a missed event or error provided.	Reduces the network traffic since artifacts are only fetched and stored on demand.
Enables smooth geographic failover and disaster recovery.		Reduces storage at remote sites due to on-demand caching.
		Proxying available only via the remote repository.

HOW TO IMPLEMENT REPLICATION TOPOLOGIES

The following sections use the example of an organization with four data centers. One in Amsterdam, one in Bangkok, one in Cape Town and one in Denver.

STAR TOPOLOGY

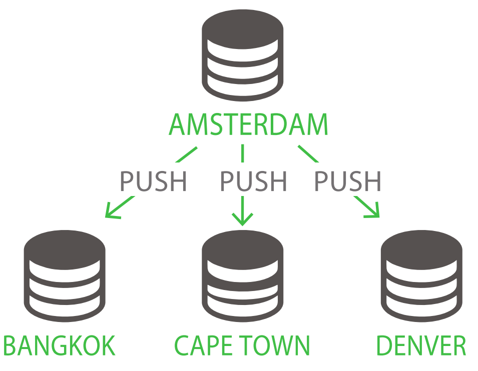
Star topology is recommended when you have a main facility doing development (say, Amsterdam), however additional development is managed at multiple remote sites (Bangkok, Cape Town, and Denver). In this case, both push and pull replication could be used, each with its own set of advantages.

Event-based multi-push replication

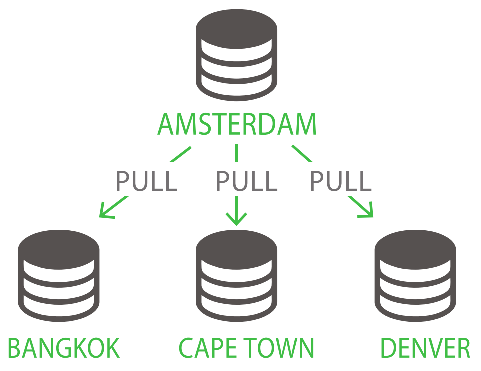
Amsterdam pushes to Bangkok, Cape Town and Denver. (Note: all sites must have Artifactory Enterprise)

Event-based Pull replication

Bangkok, Cape Town and Denver pull replicate from Amsterdam. (Note: all sites must have Artifactory Enterprise)



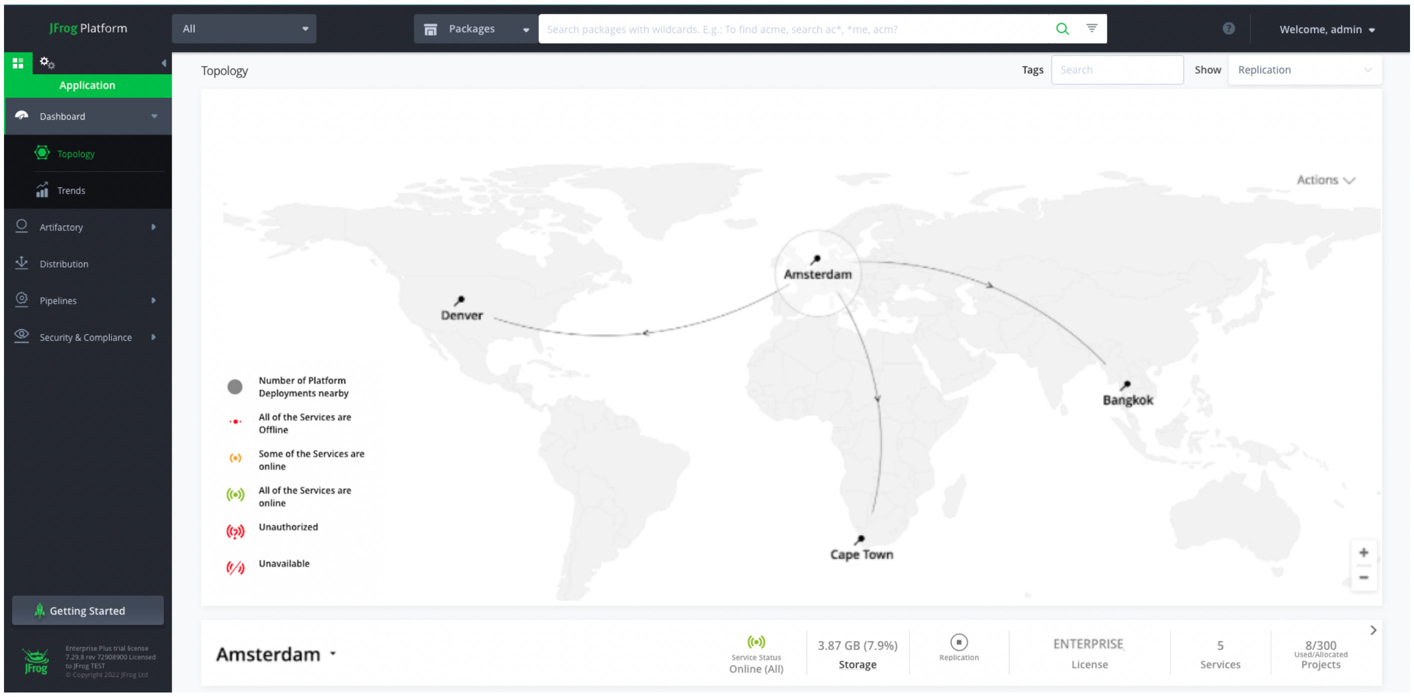
STAR TOPOLOGY USING MULTI-PUSH REPLICATION



STAR TOPOLOGY USING PULL REPLICATION

While a star topology presents benefits for both push and pull replication, it also has a significant drawback, in that the central node is potentially a single-point-of-failure.

In the following diagram, we can see an example of star topology with an instance in Amsterdam replicating to several global instances in Bangkok, Cape Town and Denver.



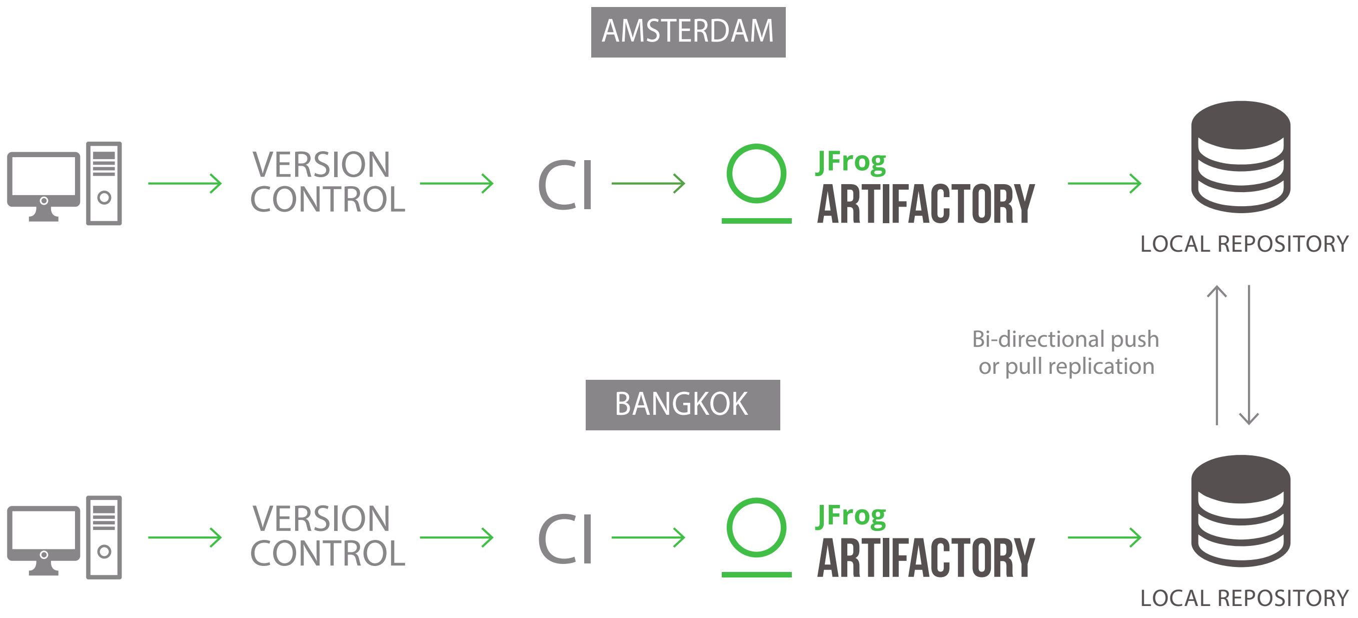
Once replication is configured we can see the replication status and schedules of all managed instances.

FULL MESH TOPOLOGY

Full mesh topology is recommended when development is more equally distributed between the different sites, however, the term is somewhat of a misnomer. A true full mesh topology implies that each side would implement a complete bi-directional synchronization (whether by push or by pull), however this is usually not considered best practice. What we are recommending is actually a star topology, but implemented per project instead of having everything centralized. There are different ways to do this as described in the sections below.

SINGLE LOCAL REPOSITORY PUSHED BETWEEN TWO SITES

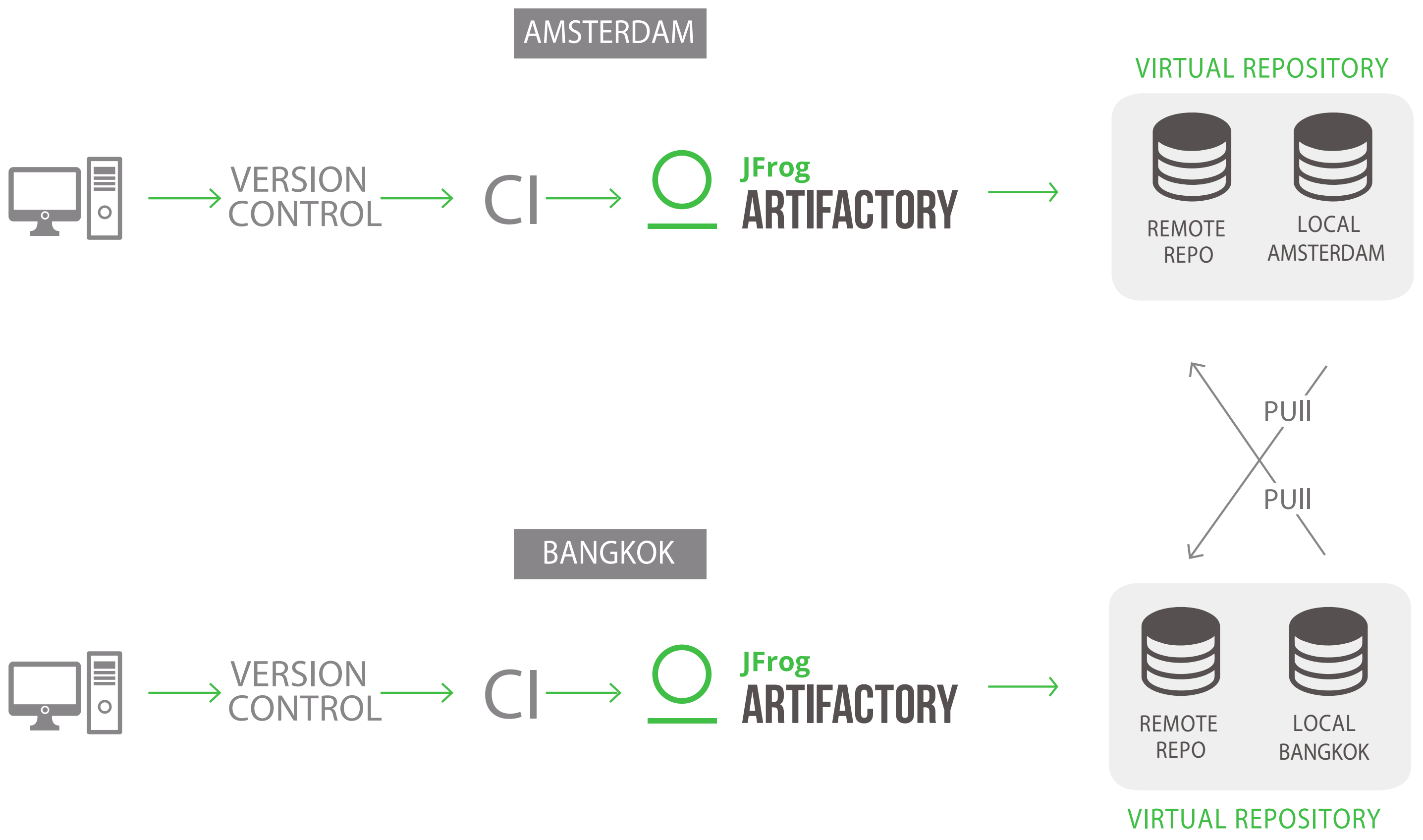
If there are modules that are developed on multiple sites, each site may deploy them to a local repository, and then the sites synchronize between them either using event-based push or pull replication.



While this solution is technically possible, pushing updates in both directions is very risky and poses a significant risk that data will be lost, especially if delete synchronization is enabled during the event-based push replication. Consider if Amsterdam is updated with a set of artifacts. If Bangkok now runs its scheduled synchronization process before Amsterdam manages to push over the update, Bangkok will delete those files from Amsterdam. This solution is therefore not recommended.

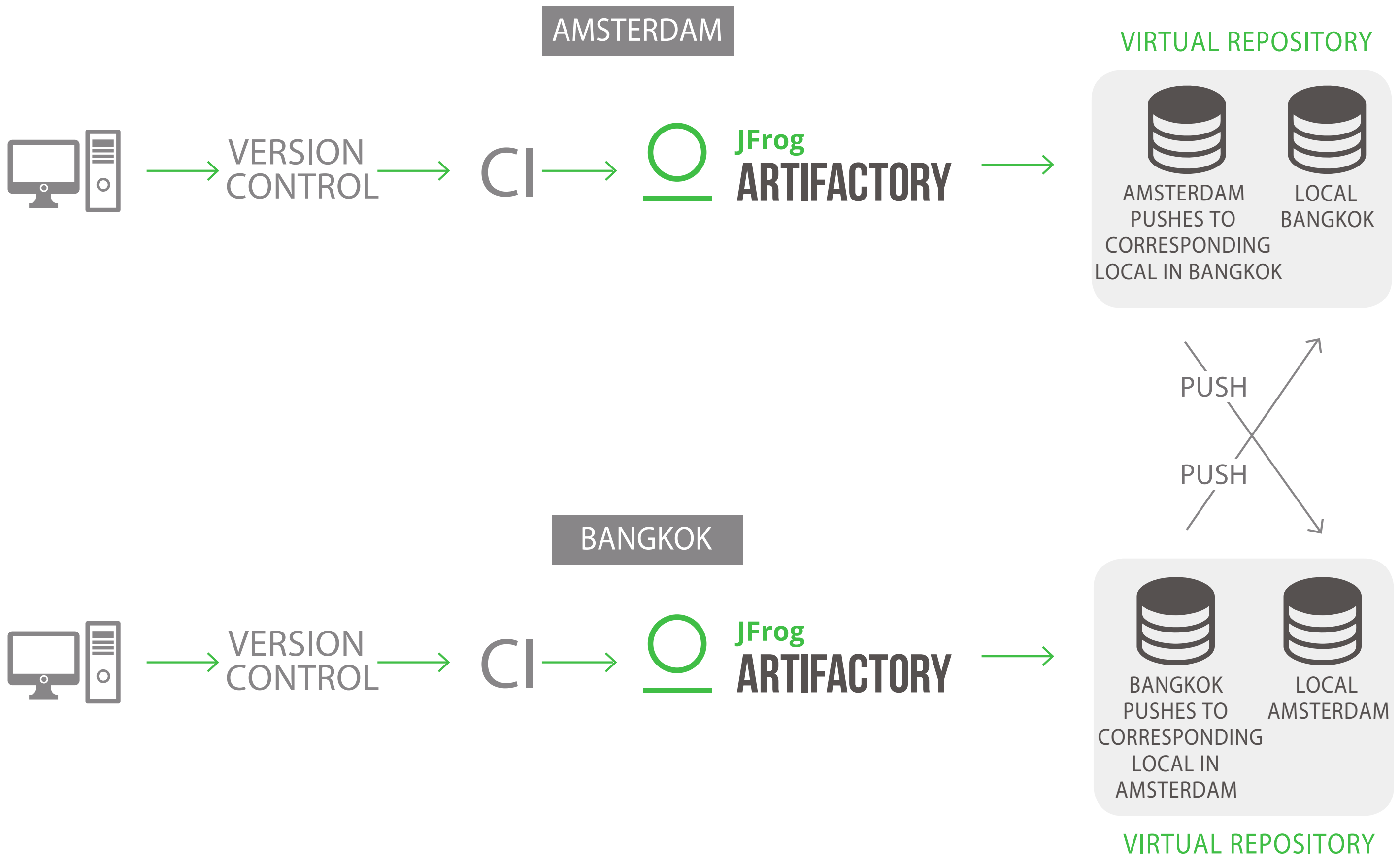
SINGLE VIRTUAL REPOSITORY CONSISTING OF A LOCAL AND REMOTE REPOSITORY

A better way to implement full mesh topology is to have each site manage a local and a remote repository. Each site can only write to its own local repository, while the remote repository is populated by pull replicating the local repository on the other site. In other words, Artifactory in Bangkok pull replicates from the local repository in Amsterdam, to its own corresponding remote repository, and vice versa. This can be done with one default deployment target which is the virtual repository that will point to the local repository ('local- amsterdam' for Amsterdam and 'local-bangkok' for Bangkok) for deployment.



SINGLE VIRTUAL REPOSITORY CONSISTING OF TWO LOCAL REPOSITORIES

Another alternative to implement full mesh topology is to have each site manage two local repositories. Each site can only write to its own local repository, while the second one is populated by being push replicated by Artifactory from the distant repository (which is local to the other site). In other words, Artifactory push replicates from the local repository in Amsterdam, to the corresponding repository in Bangkok, and vice versa. This can be done with one default deployment target which is the virtual repository that will point to the local repository ('local- amsterdam' for Amsterdam and 'local-bangkok' for Bangkok) for deployment.



This configuration works well for continuous integration between geographically distant sites because it minimizes the time taken for an artifact to become available, although since replication is not instant, there is no guarantee that the same artifact will always be available at each site. For example, if source code is updated simultaneously at both sites it is possible that each site could create a build that only contains its own updates. The sites would eventually synchronize; however, each site may create a build that does not match any build at the other site.

SINGLE VIRTUAL REPOSITORY CONSISTING OF ONE LOCAL AND MULTIPLE REMOTE REPOSITORIES (PULL REPLICATION)

Enterprise users can implement full mesh topology by having each site manage a single local repository and multiple remote repositories (that represent the other sites' local repositories). The configuration is as follows: Each site can only write to its own local repository, while the other remote repositories are populated by pull replicating from local repository on the other sites. In the example diagram below: Local repositories in the Artifactory instances in Bangkok, Cape Town and Denver are pull replicated to the corresponding remote repositories in Amsterdam. Local repositories in Amsterdam, Cape Town and Denver are pull replicated to the corresponding remote repositories in Bangkok. Local repositories in Amsterdam, Bangkok, and Denver are pull replicated to the corresponding remote repositories in Cape Town. Local repositories in Amsterdam, Bangkok, and Cape Town are pull replicated to the corresponding remote repositories in Denver.

In this environment, a solid naming convention can be crucial for two reasons: first, it reduces confusion, and second it allows for easier disaster recovery if a single node goes down. We recommend that at each site, the local nodes be named something like:

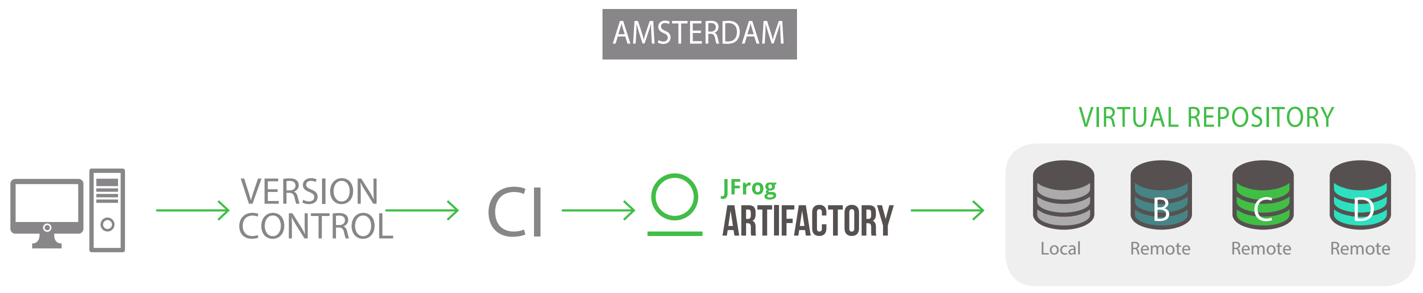
- "libs-release-amsterdam"
- "libs-release-bangkok"
- "libs-release-cape-town"
- "libs-release-denver"

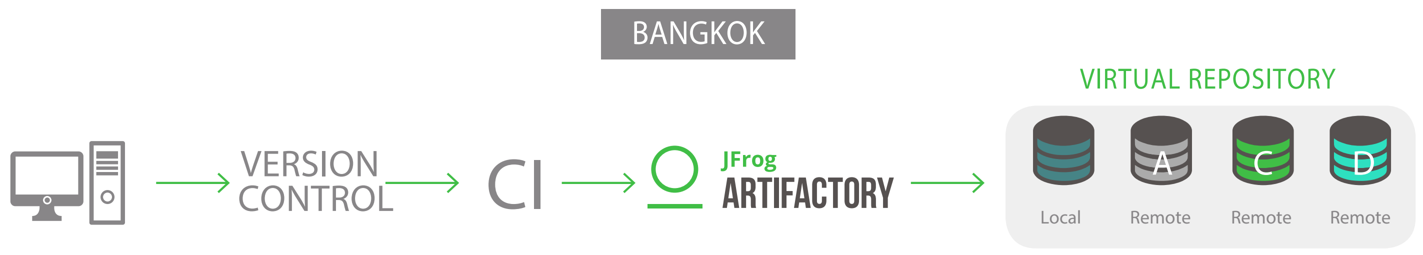
and at all the sites the remote nodes be named like:

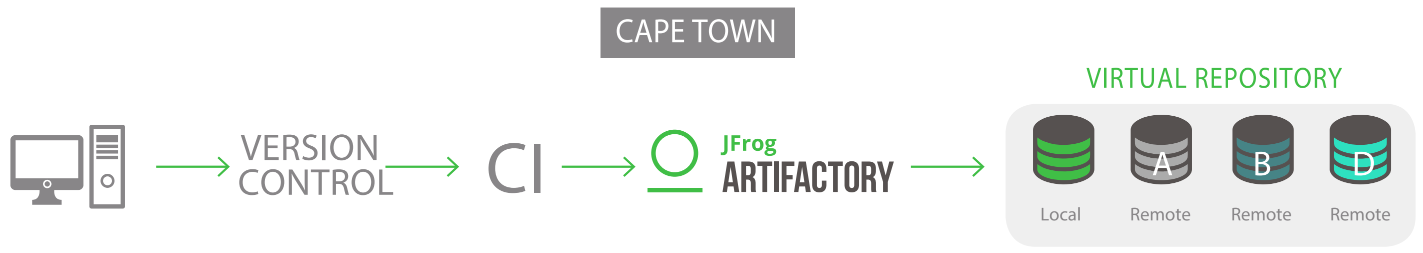
- "libs-release-amsterdam-remote"
- "libs-release-bangkok-remote"
- "libs-release-cape-town-remote"
- "libs-release-denver-remote"

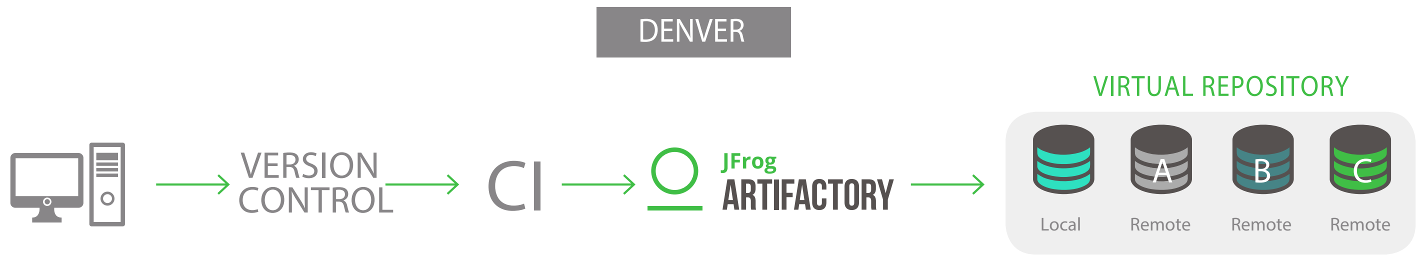
The Amsterdam CI environment writes only to "libs-release-amsterdam", the Bangkok CI environment writes only to "libs-release- bangkok" etc. If the Amsterdam Artifactory fails, the Amsterdam CI environment can be designed to fail-over to any other Artifactory in the mesh with minimal reconfiguration.

In the following diagram, we can see a full mesh topology with local repositories in each site being pull replicated by corresponding remote cache in other sites.


AMSTERDAM
VERSION CONTROL
CI
JFrog ARTIFACTORY
VIRTUAL REPOSITORY
Local
B Remote
C Remote
D Remote


BANGKOK
VERSION CONTROL
CI
JFrog ARTIFACTORY
VIRTUAL REPOSITORY
Local
A Remote
C Remote
D Remote


CAPE TOWN
VERSION CONTROL
CI
JFrog ARTIFACTORY
VIRTUAL REPOSITORY
Local
A Remote
B Remote
D Remote


DENVER
VERSION CONTROL
CI
JFrog ARTIFACTORY
VIRTUAL REPOSITORY
Local
A Remote
B Remote
C Remote

SINGLE VIRTUAL REPOSITORY CONSISTING OF MULTIPLE LOCAL REPOSITORIES (MULTI-PUSH REPLICATION)

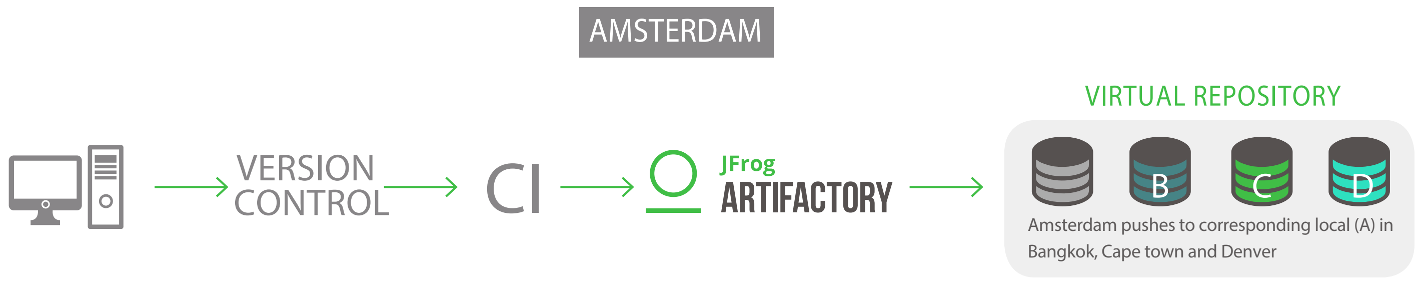
Enterprise users can implement full mesh topology by having each site manage multiple local repositories. Each site can only write to its own local repository, while the other ones are populated by being push replicated by Artifactory from the distant repository (which is local to the other site). In other words, Artifactory multi-push replicates from the local repository in Amsterdam, to the corresponding repositories in Bangkok, Cape Town and Denver, Artifactory in Bangkok multi-push replicates to Amsterdam, Cape Town and Denver, Artifactory in Cape Town multi-push replicates to Amsterdam, Bangkok and Denver and Artifactory in Denver multi-push replicates to Amsterdam, Bangkok and Cape Town.

In this environment too, a solid naming convention can be crucial for two reasons: first, it reduces confusion, and second it allows for easier disaster recovery if a single node goes down. We recommend that at all the sites, the nodes be named something like:

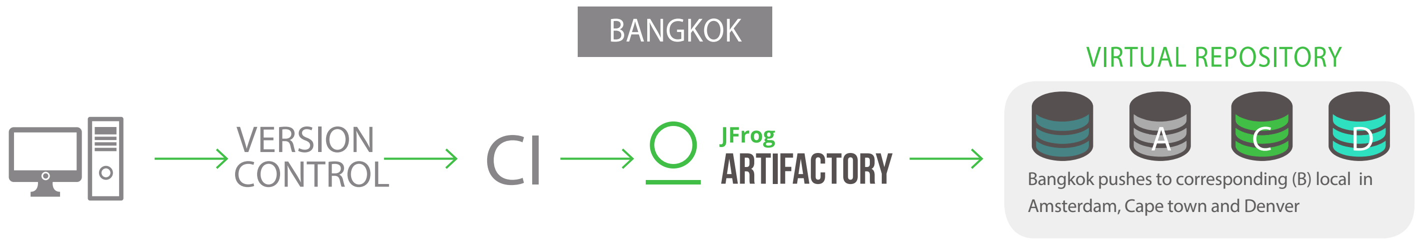
- “libs-release-amsterdam”
- “libs-release-bangkok”
- “libs-release-cape-town”
- “libs-release-denver”

In this architecture, Artifactory considers all the repositories to be local, even though several of them are actually replicated duplicates of remote repositories. Then, the Amsterdam CI environment writes only to “libs-release-amsterdam”, the Bangkok CI environment writes only to “libs-release- bangkok” etc. All other CI environments should treat the respective repositories which have been push replicated to them by others, as read-only and their user accounts should not have write access, to prevent replication-based issues. This also means that if the Amsterdam Artifactory fails, the Amsterdam CI environment can be designed to fail-over to any other Artifactory in the mesh with minimal reconfiguration.

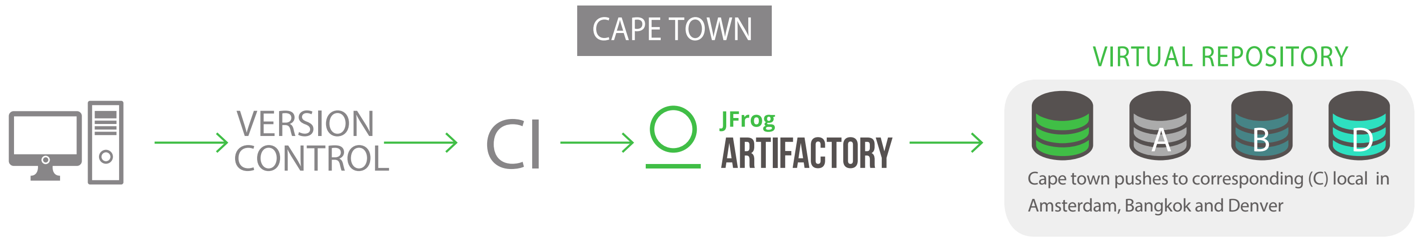
In the following diagram, we can see a full mesh topology with an instance in Amsterdam replicating repository `local-amsterdam’ to corresponding repositories in instances in Bangkok, Cape Town and Denver. In the same fashion, Bangkok, Cape Town and Denver replicate their own local repository to the corresponding one in all the other instances.



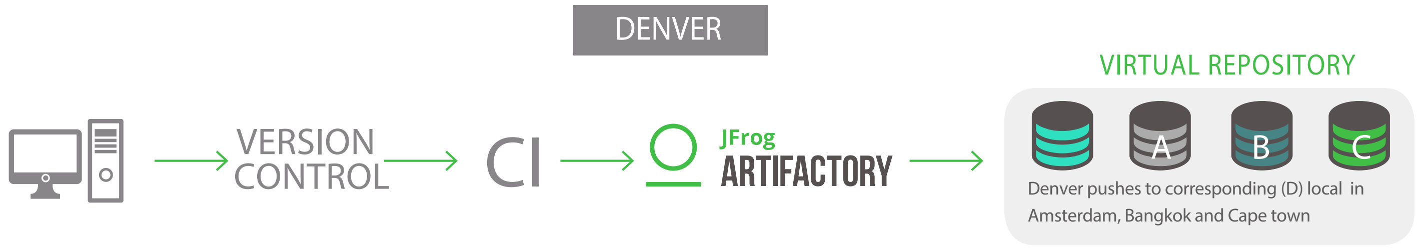
Amsterdam pushes to corresponding local (A) in Bangkok, Cape town and Denver



Bangkok pushes to corresponding (B) local in Amsterdam, Cape town and Denver

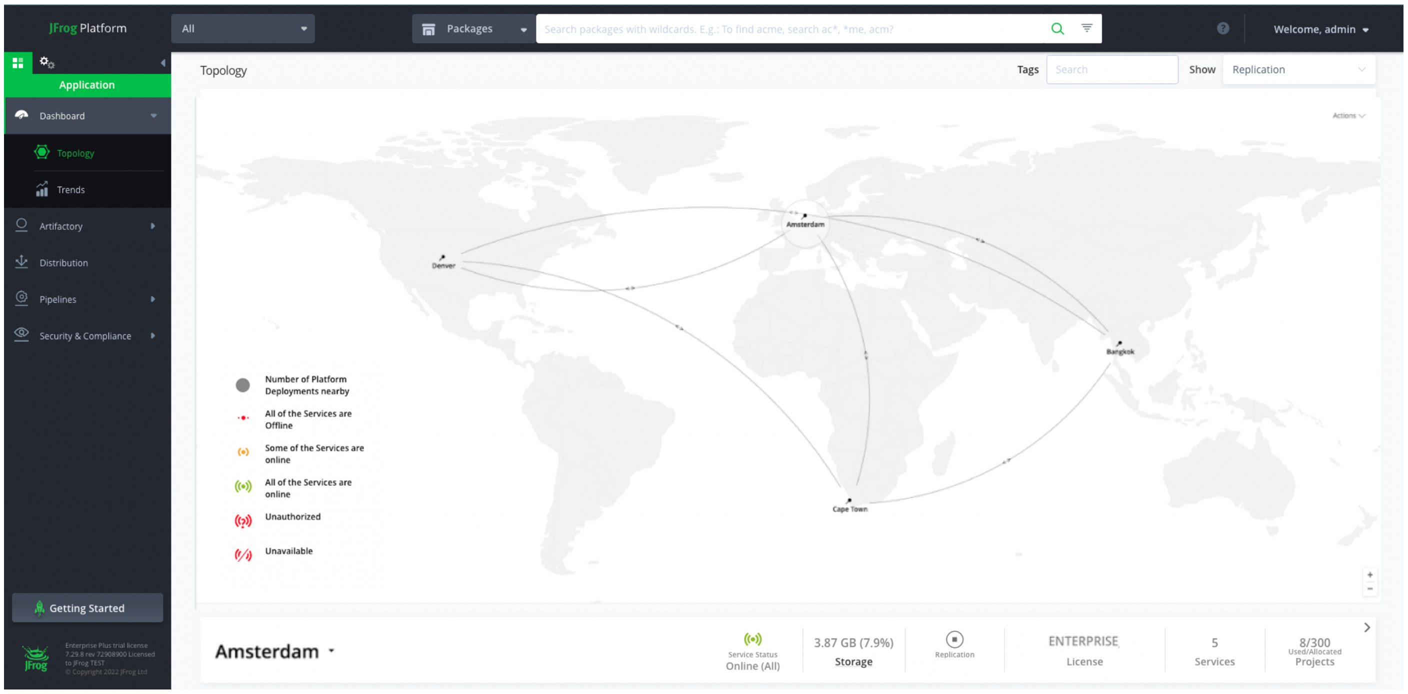


Cape town pushes to corresponding (C) local in Amsterdam, Bangkok and Denver



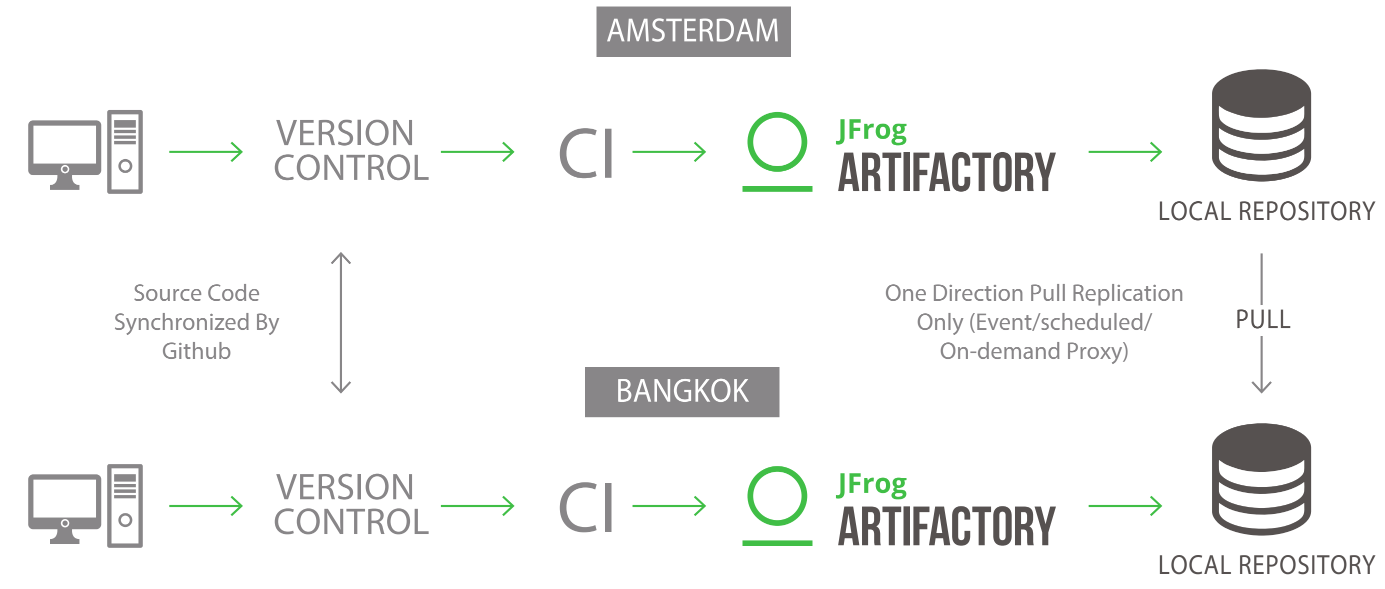
Denver pushes to corresponding (D) local in Amsterdam, Bangkok and Cape town

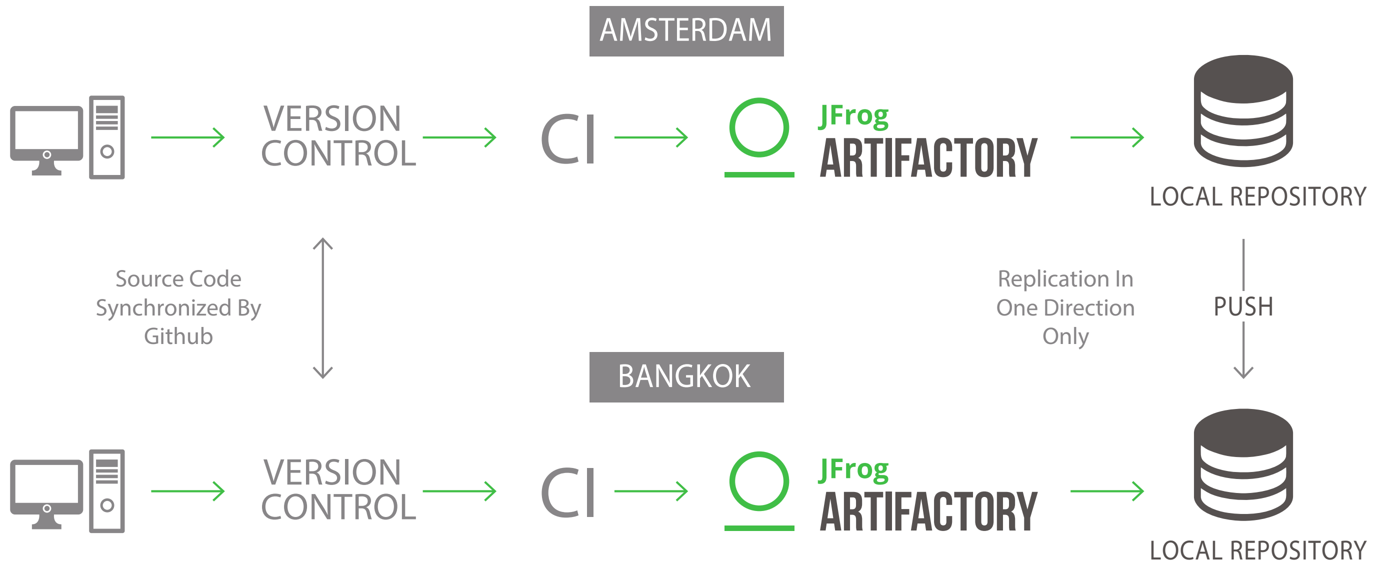
The screenshot below shows how the full mesh topology looks in JFrog Mission Control.



SINGLE LOCAL SITE WITH ARTIFACTS REPLICATED

This is the most conservative configuration, and makes the most sense if you don't want to have redundant CI servers, so only one site actually builds artifacts for distribution.





This configuration provides the strongest guarantee that artifacts are synchronized between the two sites, however this comes at the cost of adding load and build time to the CI server at the near end (Amsterdam in the above example).

GEO SYNCHRONIZED TOPOLOGY

Another topology which is an extension of the full mesh topology is a Geo Synchronized topology. This is a situation where several Artifactory instances are connected to a Geolocation Routing. Using event based push or pull replication we can have multiple instances in different geographical locations serving different global teams while each instance contains the same artifacts at any given time by replicating immediately when changes occur.

In this use case the desired outcome is to have the exact same configuration (repository names, users, groups, permission targets etc.) in all of the instances connected to the routing server so that users can deploy and resolve from the same repositories without the need to change configuration in their build tool according to the server they are being routing to (this can be done for DR purposes as well as for dividing a load in multiple locations to different instances). From an end user perspective, interactive or build server, everything is behind the scenes and they just connect to Artifactory through a single URL.

RECOMMENDED CONFIGURATIONS

The following table provides recommendations for configurations depending on your setup and other limitations you may have to address.

SETUP/GOAL	RECOMMENDATION
One central CI server You have only one CI server in a central location where you build artifacts, and you want to replicate those to satellite locations.	Use a Star Topology. Whether you use multi-push or pull replication depends on whether you have an enterprise license, and which advantages are most important to you.
Multiple CI servers You have several sites, and each has its own CI server. Each site builds a subset of all the artifacts needed by all the other sites.	Use a Full Mesh topology with event based pull replication so that all data is available even before synchronization completes. Alternatively, event based push replication can be implemented when network topology requires.
Replicating over limited bandwidth You are a satellite site without a CI server. You need to replicate a repository from the main site, but you have limited bandwidth.	You should invoke a pull replication during times of low traffic.
Replicating with limited data transfer You need to replicate a repository, but want to limit the amount of data transferred.	Use on-demand proxy by defining a remote repository to proxy the repository on the far side that you need to replicate. It is recommended NOT to synchronize deletions.
Replicating but limiting data storage You want to replicate a repository at another site, however, you also want to limit the amount of data stored at your site.	Use on-demand proxy by defining a remote repository to proxy the repository on the far side that you need to replicate. In addition, you should let Artifactory clean up artifacts that are no longer in use. The best way to do it is to set the Unused Artifacts Cleanup Period field to a non- zero value to modify and control the amount of storage that is consumed by caches.

CONCLUSION

There are several ways to set up your distributed network to support development at multiple geographically distant sites. The optimal setup depends on the number of sites, availability of CI servers at each site and different optimizations for data storage or data transfer that each organization may prefer.

This white paper has shown how Artifactory supports distributed development by supporting a variety of network topologies.

With advanced features of remote repositories, virtual repositories, federated repositories, push / multi-push replication and pull / event-based pull replication, Artifactory allows organizations to customize their multisite topology and support their distributed development environment by replicating data between sites.

For questions on how to configure your own multisite setup, please contact us at support@jfrog.com.