USING ARTIFACTORY TO MANAGE BINARIES ACROSS MULTI-SITE TOPOLOGIES

WHITE PAPER
# Table of Contents

## Introduction .................................................................................................................. 3

## On-Demand Proxy ......................................................................................................... 4

## Replicating Repositories ............................................................................................... 5

1. Push Replication ........................................................................................................... 5
   - Scheduled .................................................................................................................. 5
   - Event-Based .............................................................................................................. 5

2. Pull Replication ............................................................................................................ 7
   - Scheduled .................................................................................................................. 8
   - Event-Based .............................................................................................................. 8

   - High Availability .................................................................................................. 9

   - Establishing Replication Relationships with JFrog Mission Control ......................... 10

   - Comparing Replication Types .............................................................................. 11

## Different Ways to Implement Multi-Site Topologies .................................................... 12

- Star Topology ............................................................................................................. 13
  - Event-based multi-push replication ......................................................................... 13
  - Event-based Pull replication ..................................................................................... 13

  - Star topology using Event-based pull replication ..................................................... 15
  - Star topology using Event-based multi-push replication ......................................... 15

- Full Mesh Topology .................................................................................................... 16
  - Single Local Repository Pushed Between Two Sites .............................................. 16
  - Single Virtual Repository Consisting of a Local and Remote Repository ................. 17

  - Single Virtual Repository Consisting of Two Local Repositories ........................... 18

  - Single Virtual Repository Consisting of one Local and Multiple Remote Repositories (Pull Replication) ............................................................... 19

  - Single Virtual Repository Consisting of Multiple Local Repositories
    (Multi-Push Replication) ........................................................................................... 21

  - Full Mesh topology using Event-based pull replication ......................................... 24

  - Full Mesh topology using Event-based multi-push replication ............................... 24

  - Single Local Site with Artifacts Replicated ........................................................... 25

- Geo Synchronized Topology ......................................................................................... 26

## Recommended Configurations ...................................................................................... 27

## Conclusion .................................................................................................................... 28
Increasingly, software development has moved from the realm of highly localized teams to a collaborative endeavour of large teams at global scale. This global collaboration requires an architecture for managing software artifacts and deployable packages that is also global in scale. Three things are key: **locality, reliability, and redundancy**.

**Locality** is an important aspect of modern distributed development. When an organization has development teams across the world, it’s very important to make sure that network latency and bandwidth limitations don’t harm the development productivity. The way to do this is to ensure that all the development needs, such as external and internal dependencies, exist locally. Locality also enhances **reliability** as it ensures that external network failures do not cause development and deployment activities to halt. Additionally, within a region you will want critical services to be highly available so that individual hardware faults within a data center will not limit access to the service.

Reliability is, in turn, enhanced by **redundancy**. Even packages which are only generally used in a specific data center may, in the event of that data center becoming suddenly unavailable, need to be used elsewhere.

This white paper describes best practices in the architecture and use of JFrog Artifactory Pro and Enterprise editions to achieve these three goals.

Artifactory’s unique set of replication capabilities ensure locality in any network topology and for any development methodology. Considering the requirements for establishing your specific distributed pipelines and collaboration, you will have several alternatives to choose from. These include both **push and pull replication topologies**, **remote repositories**, and different scheduling strategies such as **on-demand, on-schedule** or **event-based replication**.

This white paper provides best practices for using these different options, and guidance on which factors should be considered when choosing between them.

This white paper also describes how to use JFrog Mission Control to set up, manage and operate these global topologies.
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.

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

REMOTE REPOSITORIES
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.

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 party commercial components. Using local repositories, all of your internal resources can be made available from a single access point across your organization from one common URL.
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**.

1. **PUSH REPPLICATION**

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](https://en.wikipedia.org/wiki/Cron) 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](https://en.wikipedia.org/wiki/Checksum) 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 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 nodes simultaneously. The alternative is setting up replication chains, which means that each node will propagate artifacts to another node, creating a serial chain of nodes. This solution is more complicated to setup and increases the inconsistency between the nodes, 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.
2. PULL REPPLICATION

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 - on demand. 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.

VIRTUAL REPOSITORIES

A virtual repository encapsulates any number of local and remote repositories, and represents them as a unified repository accessed from a single URL. It gives you a way to manage which repositories are accessed by developers since you have the freedom to mix, match and modify the actual repositories included within the virtual repository. You can also optimize artifact resolution by defining the underlying repository order so that Artifactory will first look through local repositories, then remote repository caches, and only then Artifactory will go through the network and request the artifact directly from the remote resource. For the developer it's simple. Just request the package, and Artifactory will safely and optimally access it according to your organization's policies.

Learn more >
**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 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.
Artifactory supports a **High Availability** network configuration with a cluster of 2 or more Artifactory servers on the same Local Area Network. Sharing resources between the participating servers creates a redundant network architecture that achieves load balancing and failover ensuring there is no single-point-of-failure. However, to maintain high availability, the participating Artifactory servers must be installed in geographically close locations (preferably the same data center) with a network latency of 1ms or less. Higher latency will cause a rapid deterioration of system performance essentially making it unsuitable for high availability systems. **As a result, an Artifactory HA configuration is not suitable to implement replication between geographically distant locations.**

There are different techniques to achieve load balancing and failover in geographically distributed systems, however this is not high availability, and is beyond the scope of this document.

---

**HIGH AVAILABILITY SYSTEMS**

Systems that are considered mission-critical to an organization can be deployed in a High Availability configuration to increase stability and reliability. This is done by replicating nodes in the system and deploying them as a redundant cluster to remove the complete reliability on any single node. In a High Availability configuration there is no single-point-of-failure. If any specific node goes down, the system continues to operate seamlessly and transparently to its users through the remaining, redundant nodes, with no down time or degradation of system performance as a whole.

[Learn more >](#)
ESTABLISHING REPPLICATION RELATIONSHIPS
WITH JFROG MISSION CONTROL

The easiest and most efficient way to establish replication relationships between different Artifactory instances is through JFrog Mission Control. Mission Control provides centralized control over any number of Artifactory instances enabling enterprises to monitor and manage globally distributed instances of Artifactory through a single application. As such, it allows enterprises to create replication relationships using simple DSL scripts from a single command and control center without the need to create and configure repositories and replication in each instance individually. Replication can be configured either by creating new repositories in multiple instances, and then configuring replication between them (one-to-one, or one-to-many), or by updating an existing repository and applying a replication DSL script to it.

Once a multi-site topology is created and configured, Mission Control displays a map which shows the network of participating Artifactory services and their replication relationships.
# Comparing Replication Types

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

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

<table>
<thead>
<tr>
<th>Event-Based Replication</th>
<th>Scheduled (Cron) Replication</th>
<th>On-Demand Proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest time to Global consistency. Minimizes time repositories are not synchronized.</td>
<td>Replication traffic managed to low traffic periods.</td>
<td>Only artifacts that are resolved on the far end are replicated (and cached).</td>
</tr>
<tr>
<td>The replication traffic is spread out. No ‘lump jobs’.</td>
<td>Guarantees full synchronization in case of a missed event or error provided.</td>
<td>Reduces the network traffic since artifacts are only fetched and stored on demand.</td>
</tr>
<tr>
<td>Enables smooth geographic failover and disaster recovery.</td>
<td></td>
<td>Reduces storage at remote sites due to on-demand caching.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proxying available only via the remote repository.</td>
</tr>
</tbody>
</table>
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 REPPLICATION

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

EVENT-BASED PULL REPPLICATION

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

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 service in Amsterdam replicating to several global services in Bangkok, Cape Town and Denver.

Once replication is configured using Mission Control we can see the replication status and schedules of all managed services.
### Star Topology Using Event-Based Pull Replication:

<table>
<thead>
<tr>
<th>Type</th>
<th>Source</th>
<th>Destination</th>
<th>Status</th>
<th>Last Run</th>
<th>Start Time</th>
<th>Duration</th>
<th>Next Run</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>remote-amsterdam</td>
<td>remote-amsterdam</td>
<td>Active</td>
<td>21-11-17 12:00 UTC</td>
<td>00000000</td>
<td>00000000</td>
<td>21-11-17 12:00 UTC</td>
<td>Active</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Source</th>
<th>Destination</th>
<th>Status</th>
<th>Last Run</th>
<th>Start Time</th>
<th>Duration</th>
<th>Next Run</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>remote-amsterdam</td>
<td>remote-amsterdam</td>
<td>Active</td>
<td>21-11-17 12:00 UTC</td>
<td>00000000</td>
<td>00000000</td>
<td>21-11-17 12:00 UTC</td>
<td>Active</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Source</th>
<th>Destination</th>
<th>Status</th>
<th>Last Run</th>
<th>Start Time</th>
<th>Duration</th>
<th>Next Run</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>remote-amsterdam</td>
<td>remote-amsterdam</td>
<td>Active</td>
<td>21-11-17 12:00 UTC</td>
<td>00000000</td>
<td>00000000</td>
<td>21-11-17 12:00 UTC</td>
<td>Active</td>
</tr>
</tbody>
</table>

### Star Topology Using Event-Based Multi-Push Replication:

<table>
<thead>
<tr>
<th>Type</th>
<th>Source</th>
<th>Destination</th>
<th>Status</th>
<th>Last Run</th>
<th>Start Time</th>
<th>Duration</th>
<th>Next Run</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>remote-amsterdam</td>
<td>remote-amsterdam</td>
<td>Active</td>
<td>21-11-17 12:00 UTC</td>
<td>00000000</td>
<td>00000000</td>
<td>21-11-17 12:00 UTC</td>
<td>Active</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Source</th>
<th>Destination</th>
<th>Status</th>
<th>Last Run</th>
<th>Start Time</th>
<th>Duration</th>
<th>Next Run</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>remote-amsterdam</td>
<td>remote-amsterdam</td>
<td>Active</td>
<td>21-11-17 12:00 UTC</td>
<td>00000000</td>
<td>00000000</td>
<td>21-11-17 12:00 UTC</td>
<td>Active</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Source</th>
<th>Destination</th>
<th>Status</th>
<th>Last Run</th>
<th>Start Time</th>
<th>Duration</th>
<th>Next Run</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>remote-amsterdam</td>
<td>remote-amsterdam</td>
<td>Active</td>
<td>21-11-17 12:00 UTC</td>
<td>00000000</td>
<td>00000000</td>
<td>21-11-17 12:00 UTC</td>
<td>Active</td>
</tr>
</tbody>
</table>
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.

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.
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.
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.
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” and “libs-release-denver” respectively and at all the sites the remote nodes be named like “libs-release-amsterdam-remote” “libs-release-bangkok-remote” “libs-release-cape-town-remote”, and “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.
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” and “libs-release-denver” respectively. 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 local (D) in Amsterdam Bangkok and Cape Town
The full mesh topology described above can be configured using JFrog Mission Control. It can easily be implemented by applying a configuration script for replication to each instance specifying its local repository as the source and the corresponding local or remote repository at each of the other destinations as the target.

The diagrams below illustrate how the full mesh topology looks in JFrog Mission Control.

**DOWNLOAD**
You can download reusable configuration scripts to implement all the topologies presented in this white paper from JFrog’s Mission Control configuration scripts on GitHub.
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.

This topology can be tedious to implement without Mission Control. Using Mission Control, you can either apply the same configuration using DSL scripts to multiple instances or import configuration from one instance and apply it to a few other instances. Then you can easily create event push or pull replication between all instances as mentioned above.
# RECOMMENDED CONFIGURATIONS

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

<table>
<thead>
<tr>
<th>SETUP/GOAL</th>
<th>RECOMMENDATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>One central CI server</td>
<td>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.</td>
</tr>
<tr>
<td>You have only one CI server in a central location where you build artifacts, and you want to replicate those to satellite locations.</td>
<td></td>
</tr>
<tr>
<td>Multiple CI servers</td>
<td>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.</td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Replicating over limited bandwidth</td>
<td>You should invoke a pull replication during times of low traffic.</td>
</tr>
<tr>
<td>You are a satellite site without a CI server. You need to replicate a repository from the main site, but you have limited bandwidth.</td>
<td></td>
</tr>
<tr>
<td>Replicating with limited data transfer</td>
<td>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.</td>
</tr>
<tr>
<td>You need to replicate a repository, but want to limit the amount of data transferred.</td>
<td></td>
</tr>
<tr>
<td>Replicating but limiting data storage</td>
<td>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.</td>
</tr>
<tr>
<td>You want to replicate a repository at another site, however, you also want to limit the amount of data stored at your site.</td>
<td></td>
</tr>
</tbody>
</table>
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, push / multi-push replication and pull / event-based pull replication, Artifactory allows organizations to customize their multi-site topology and support their distributed development environment by replicating data between sites.

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