JFrog

USING ARTIFACTORY TO MANAGE BINARIES ACROSS MULTI-SITE TOPOLOGIES

White Paper

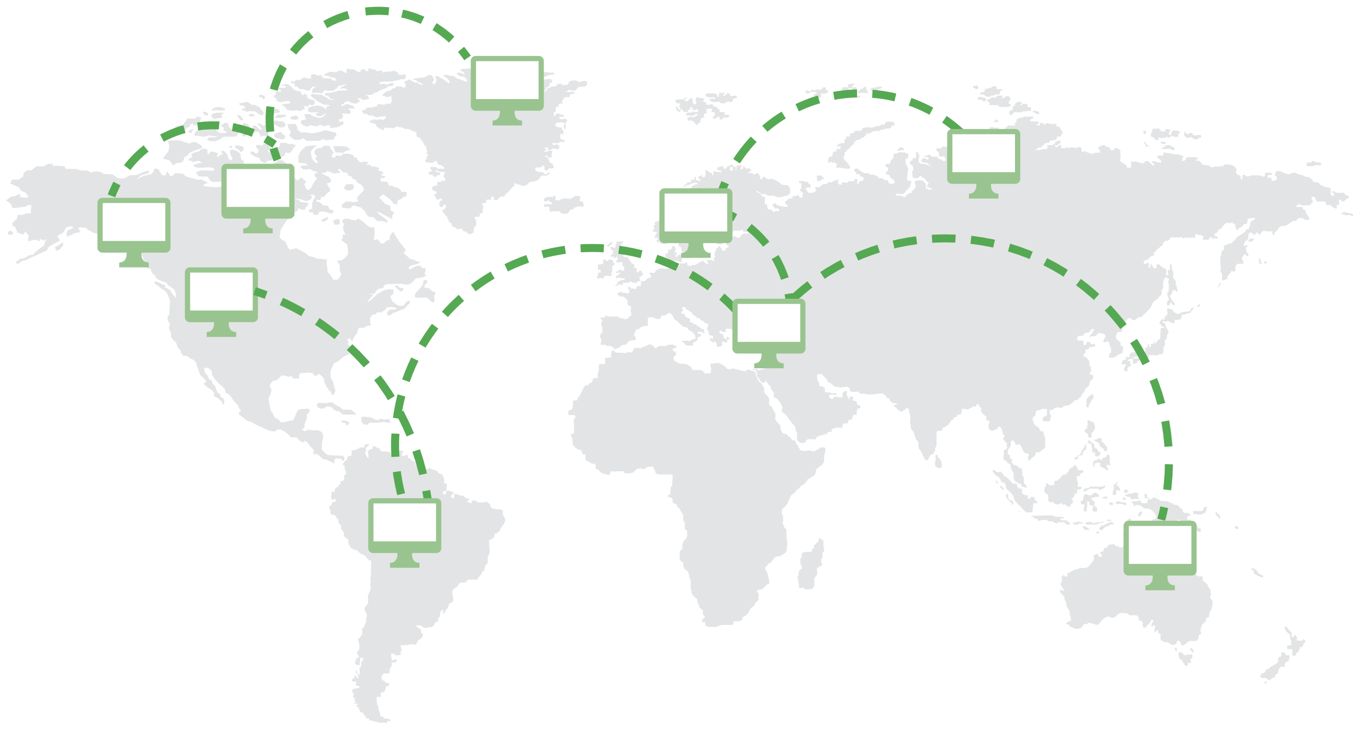
June 2016 | www.jfrog.com

INTRODUCTION

Distributed software development has become commonplace, especially in large enterprises that have several sites in different locations around the globe. This presents many challenges to ensure that all the development teams work on a coherent and synchronized code base. For example:

- Ensuring that developers all work with the same version of remote artifacts
- Ensuring that all build artifacts are shared efficiently between the different teams
- Overcoming connectivity issues such as network stability and latency when accessing remote artifacts
- Accessing specific versions of remote artifacts

To overcome these challenges, Artifactory supports several ways to replicate repositories and accommodate a variety of distributed topologies to meet the needs of any enterprise.



REPLICATING REPOSITORIES

Artifactory supports five ways to replicate repositories:

On-Demand Proxy

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 **Off line** button in the repository configuration. In this case Artifactory will only provide remote artifacts that have already been cached.

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.

Learn more >

Local Push Replication

Push replication is used to synchronize **local repositories**. Pushes are scheduled at regular intervals, and have the unique ability to update the far end asynchronously through events. 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.

Local Repositories

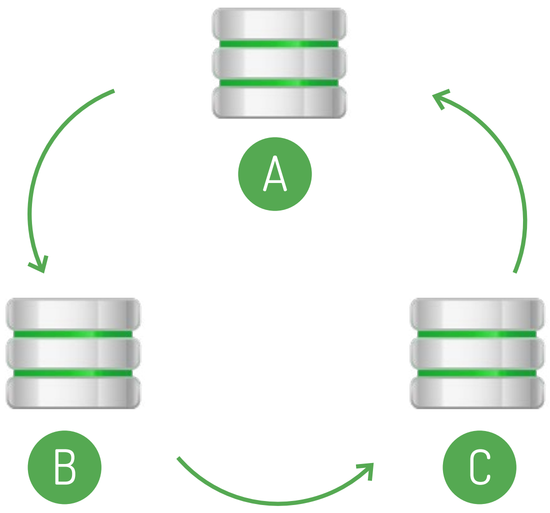
Local repositories are physical, locally-managed repositories into which you can deploy artifacts. Typically, these are used to deploy internal and external releases as well as development builds, but they can also be used to store binaries that are not widely available on public repositories such as 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.

Learn more >

Event Based Local Push Replication

Event driven push enables repositories to be updated nearly in real time. Each create, copy, move or delete of an artifact is immediately propagated to the far side, but you have the flexibility to disable synchronization of metadata or deleted artifacts if necessary. To ensure that all changes on the near end are propagated to the far end, it is highly recommended to perform scheduled synchronization of repositories.

All artifactory licenses currently support push replication from one repository to another single repository on the far end. But if you need to replicate a repository to multiple nodes you need an Enterprise license which supports multi-push. The alternative, replication chains, can get complicated and therefore do not comply with best practice for repository replication. There is 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 an enterprise license, pull replication is recommended.



Replication loop to be strictly avoided

Pull Replication

Pull replication is invoked by a remote repository, and runs according to a defined schedule to synchronize repositories at regular intervals. 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 >

High Availability

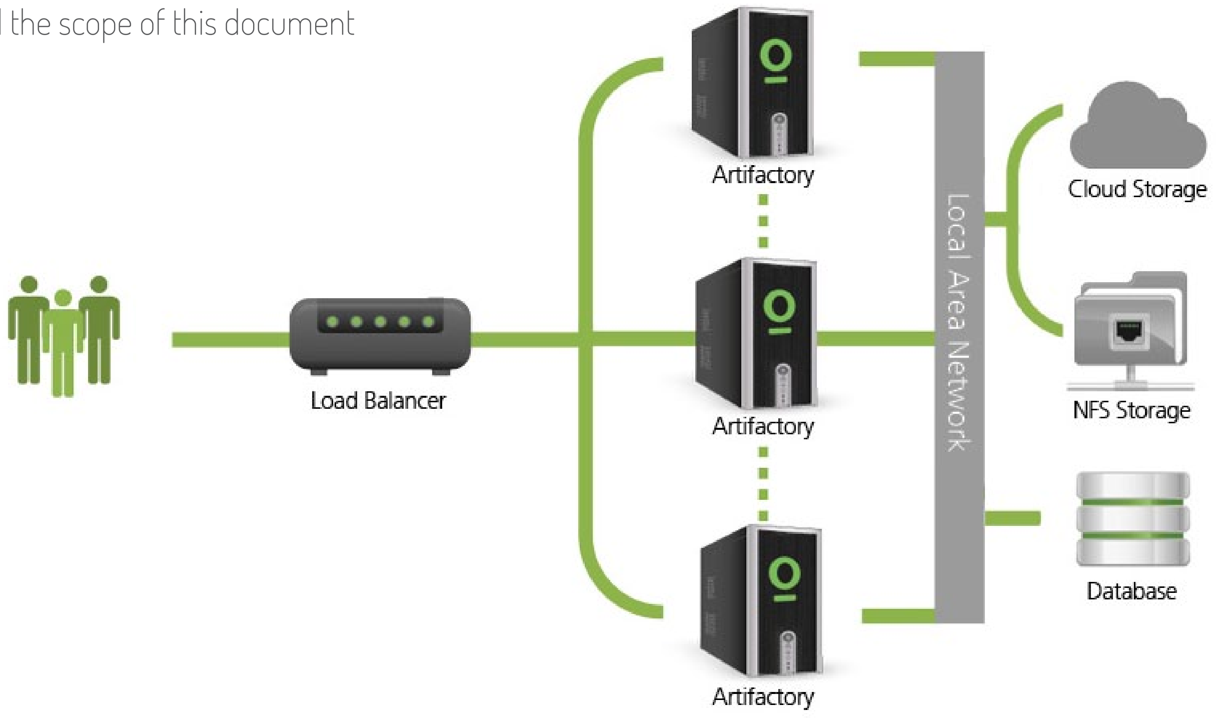
Artifactory supports a **High Availability** network configuration with a cluster of 2 or more Artifactory servers on the same Local Area Network. Replication between the participating servers creates a redundant network architecture that achieves load balancing and failover ensuring there is no single-point-of-failure. 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 10ms 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 Replication 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 multi-push 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 erected and configured, Mission Control displays a geo-location map which shows the global network of participating Artifactory instances along with their replication relationships.

DIFFERENT WAYS TO IMPLEMENT MULTI-SITE TOPOLOGIES

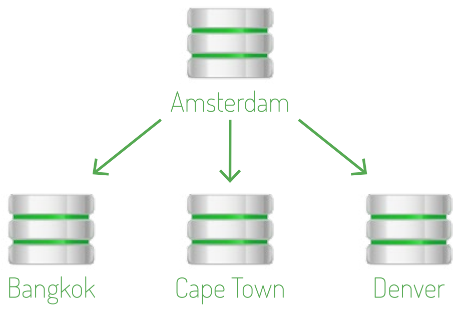
The following sections use the example of an organization with four data centers. One in Amsterdam, one in Bangkok, one in Cape Town and one in Denver.

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 an Enterprise license)



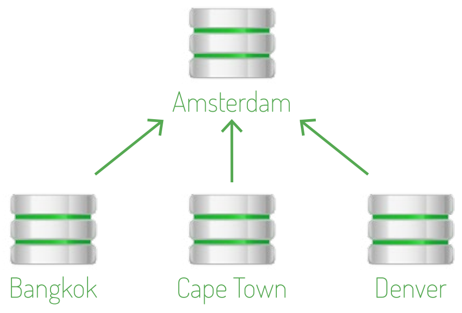
Star topology using multi-push replication

Advantages

- Fast because replication is asynchronous.
- Minimizes time repositories are not synchronized
- Enables smooth geographic failover and disaster recovery
- No replication-based heavy use slowdowns

Pull replication

Bangkok, Cape Town and Denver pull replicate from Amsterdam.



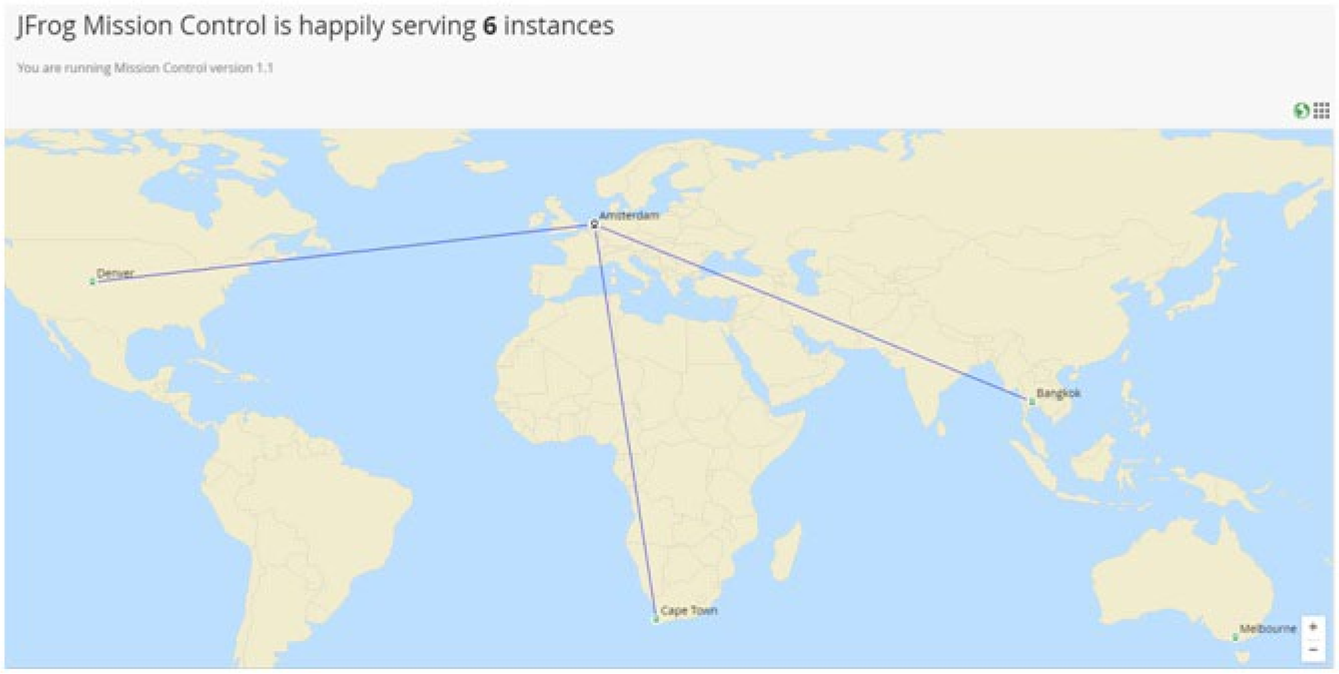
Star topology using pull replication

Advantages

- Pulls packages from remote sites only on request, reducing network load during peak use times
- Reduces traffic on master node
- All data is available even before synchronization completes
- Full synchronization scheduled only for non-peak times

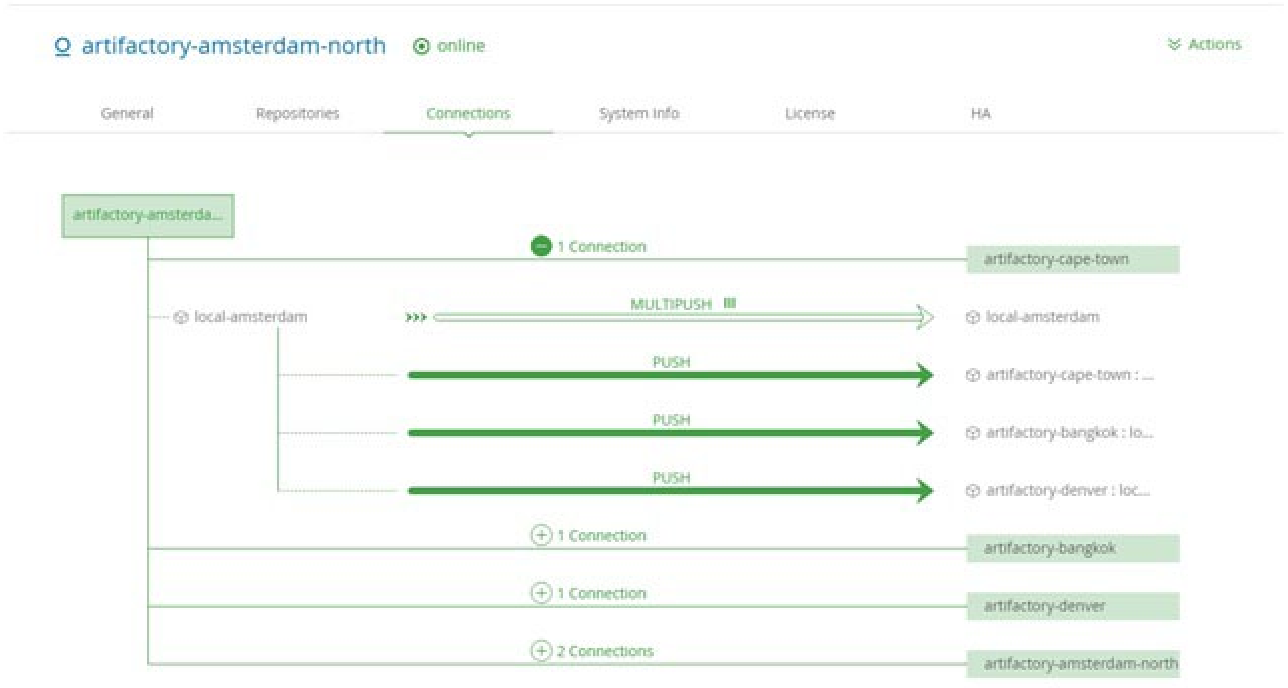
While a star topology presents benefits for both push and pull replication, it also has a significant drawback, in that the central node is potentially a single-point-of-failure.

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.


JFrog Mission Control is happily serving **6** instances

You are running Mission Control version 1.1

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

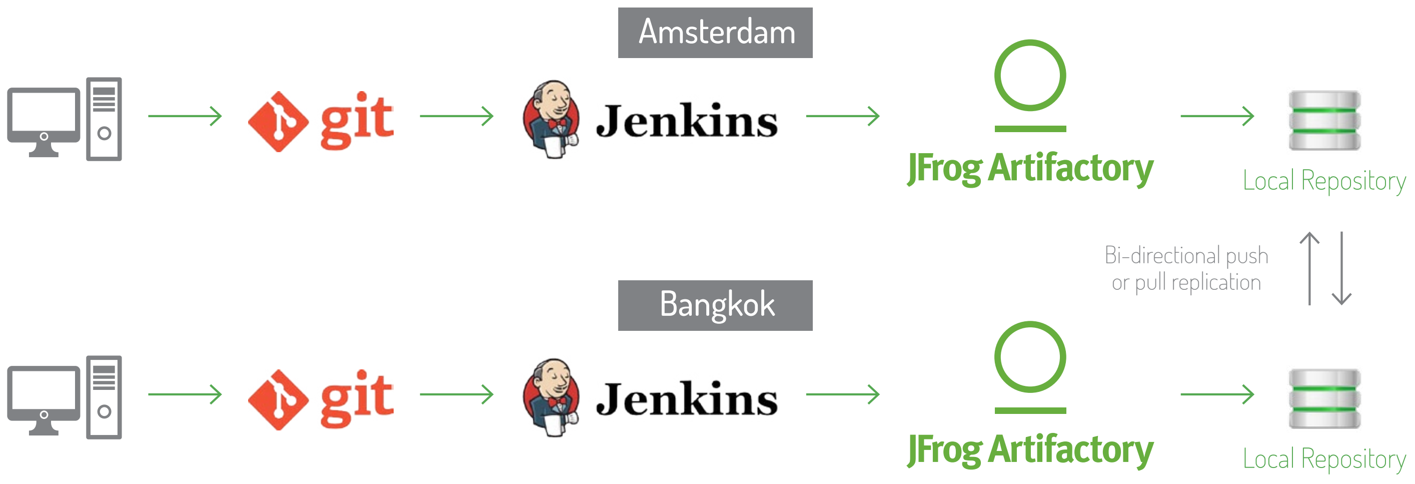

artifactory-amsterdam-north ⊙ online

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

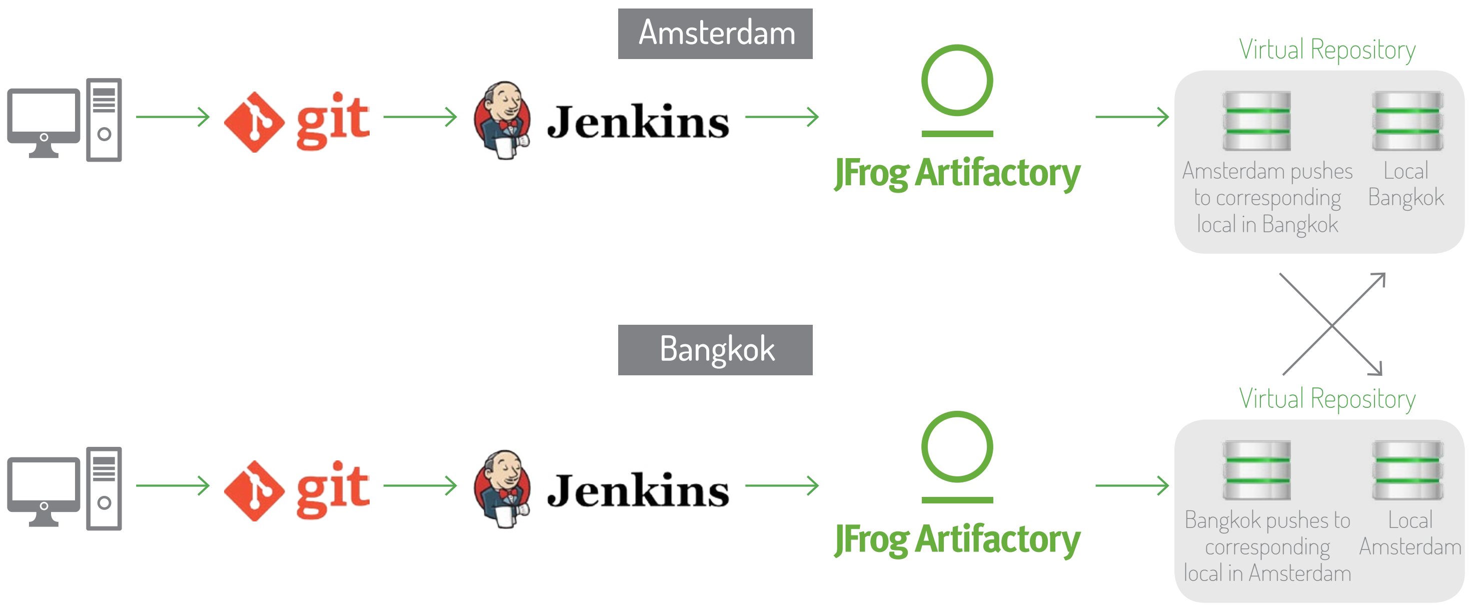
If there are modules that are developed on multiple sites, each site may deploy them to a local repository, and then the sites synchronize between them.



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. 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 Two Local Repositories

A better way 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.



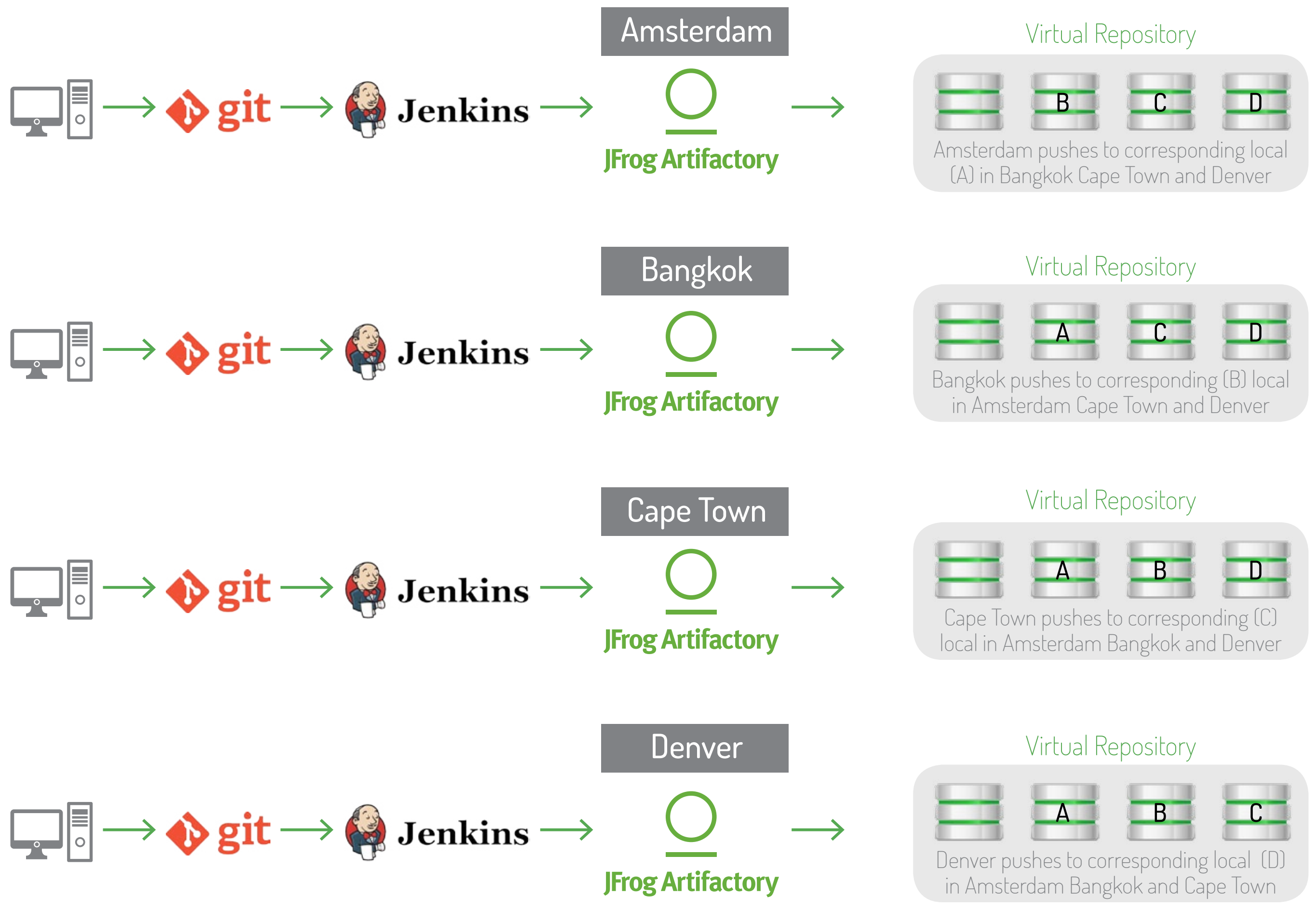
This configuration works well for continuous integration between geographically distant sites because it minimizes the time taken for an artifact to become available, although since replication is not instant, there is no guarantee that the same artifact will always be available at each site. For example, 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 Multiple Local Repositories

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, 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
git
Jenkins
JFrog Artifactory
Virtual Repository
B
C
D
Amsterdam pushes to corresponding local (A) in Bangkok Cape Town and Denver
Bangkok
git
Jenkins
JFrog Artifactory
Virtual Repository
A
C
D
Bangkok pushes to corresponding (B) local in Amsterdam Cape Town and Denver
Cape Town
git
Jenkins
JFrog Artifactory
Virtual Repository
A
B
D
Cape Town pushes to corresponding (C) local in Amsterdam Bangkok and Denver
Denver
git
Jenkins
JFrog Artifactory
Virtual Repository
A
B
C
Denver pushes to corresponding local (D) in Amsterdam Bangkok and Cape Town

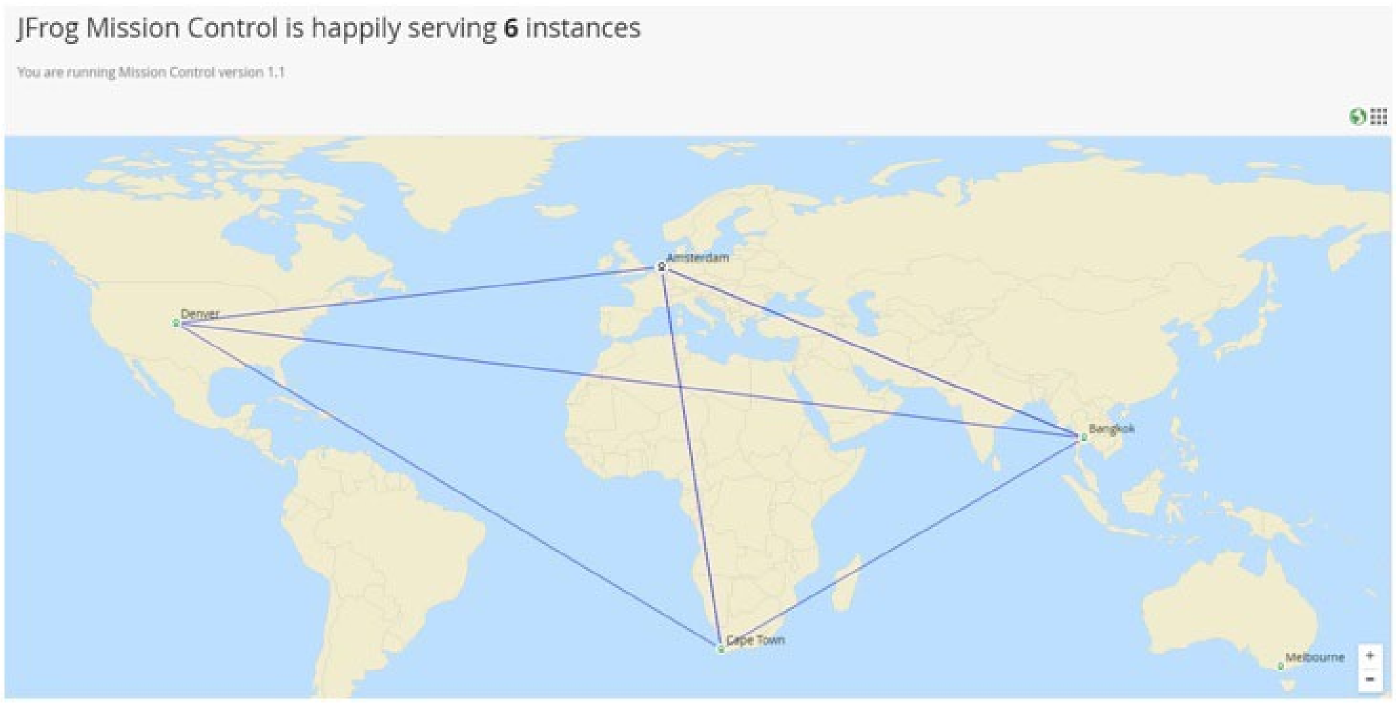
Again, this configuration only works for Enterprises users

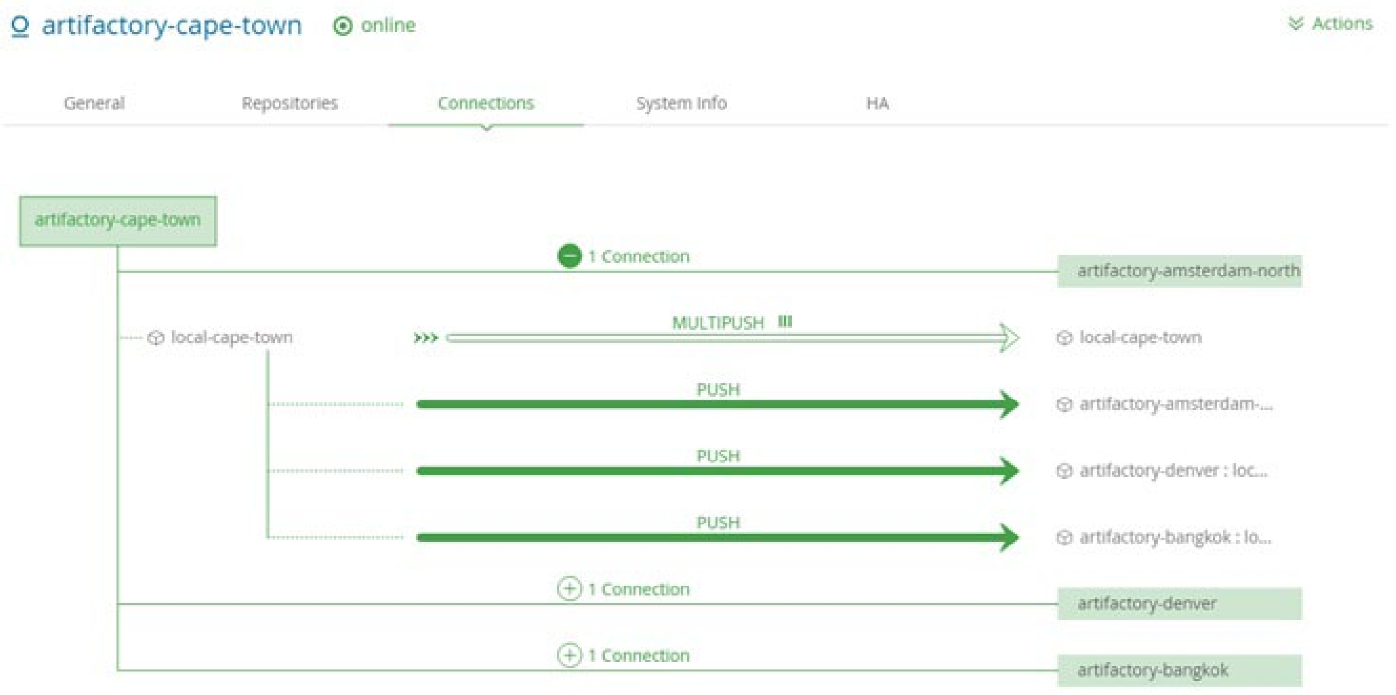
The full mesh topology described can be configured using JFrog Mission Control. It can easily be implemented by applying a configuration script for replication to each instance specifying its local repository as the source and the corresponding repository at each of the other destinations as the target.

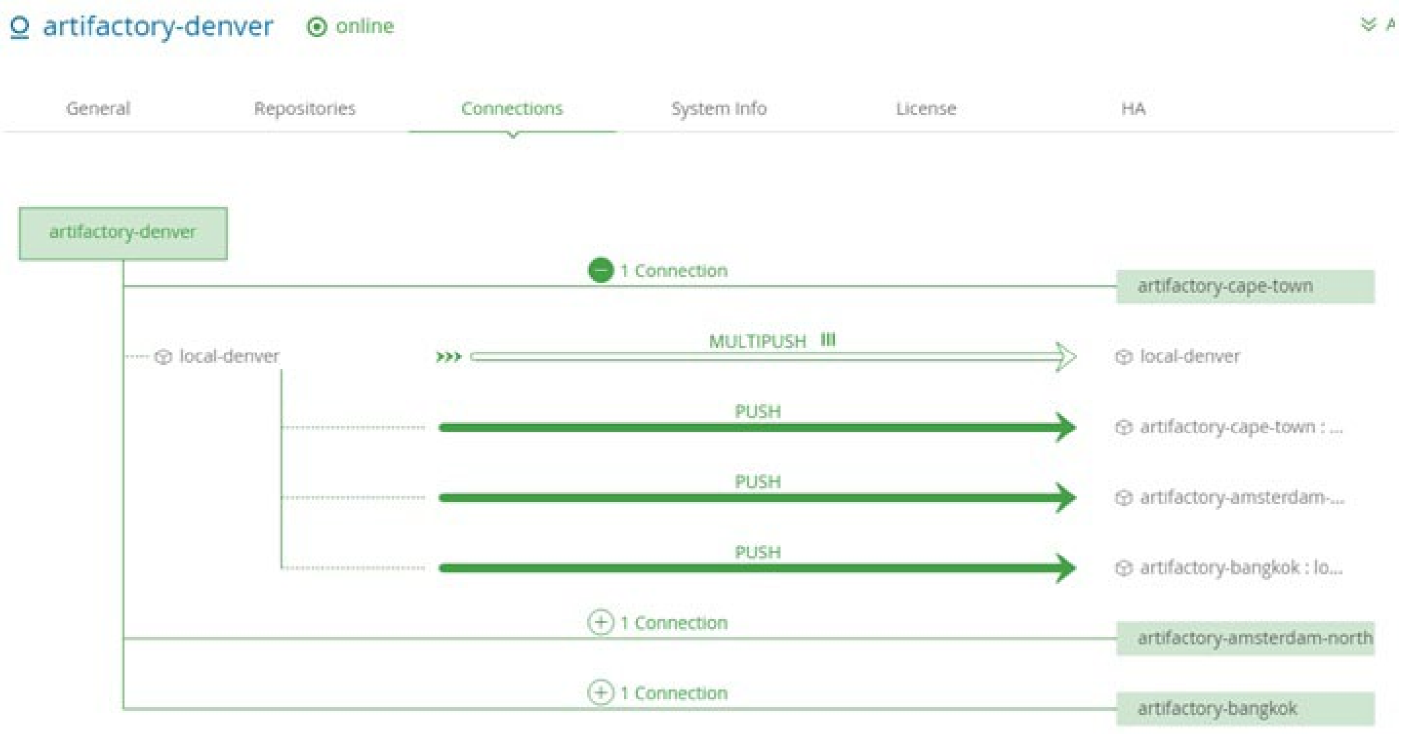
Download

You can download reusable configuration scripts to implement a full mesh topology from JFrog's [Mission Control Configurations Scripts]() project on GitHub.

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

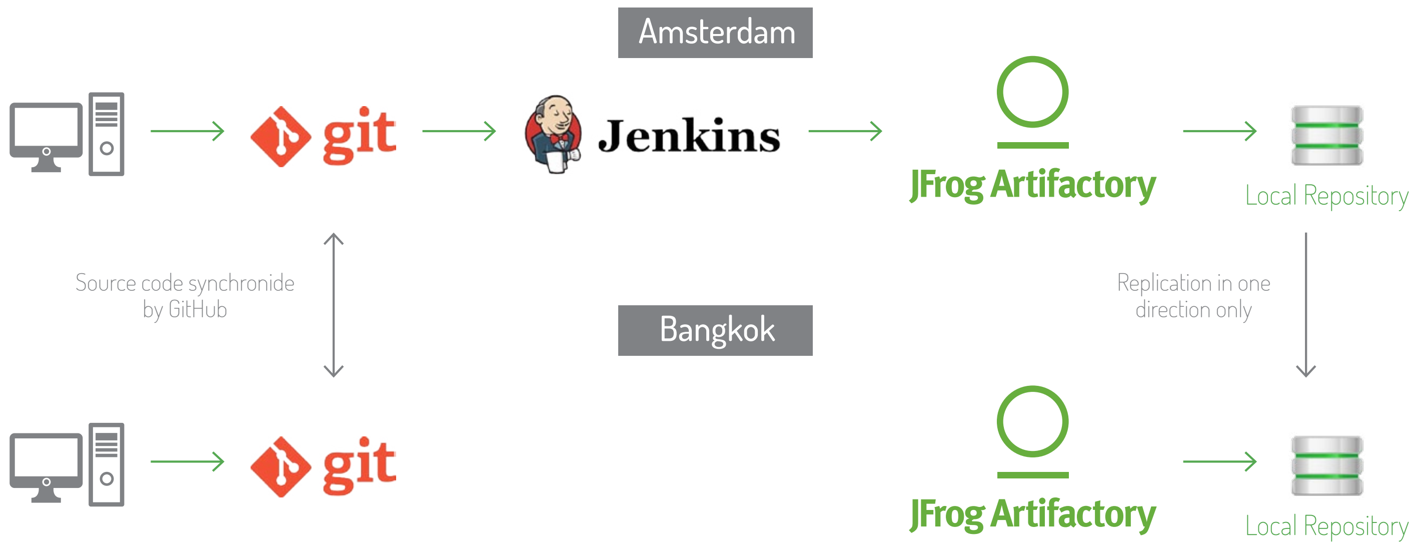

JFrog Mission Control is happily serving 6 instances
You are running Mission Control version 1.1
Amsterdam
Denver
Bangkok
Cape Town
Melbourne


artifactory-cape-town
online
Actions
General
Repositories
Connections
System Info
HA
artifactory-cape-town
1 Connection
artifactory-amsterdam-north
local-cape-town
MULTIPUSH
local-cape-town
PUSH
artifactory-amsterdam-...
PUSH
artifactory-denver : loc...
PUSH
artifactory-bangkok : lo...
1 Connection
artifactory-denver
1 Connection
artifactory-bangkok


artifactory-denver
online
General
Repositories
Connections
System Info
License
HA
artifactory-denver
1 Connection
artifactory-cape-town
local-denver
MULTIPUSH
local-denver
PUSH
artifactory-cape-town : ...
PUSH
artifactory-amsterdam-...
PUSH
artifactory-bangkok : lo...
1 Connection
artifactory-amsterdam-north
1 Connection
artifactory-bangkok

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 makes use of the source code multi-site replication inherent in GitHub.



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

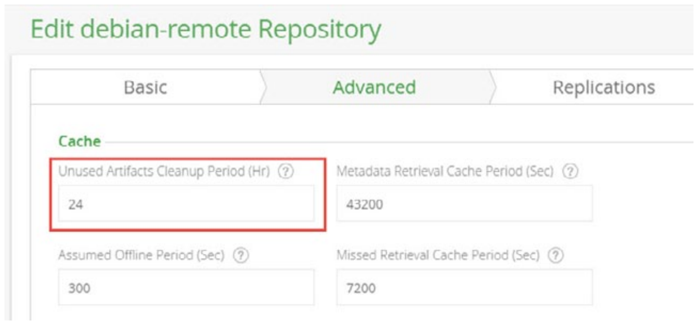
Another topology which is an extension of the full mesh topology is a Geo Synchronized topology. This is a situation where several Artifactory instances are connected to a GeoDNS. Using event based push replication we can have multiple instances in different geographical locations serving different global 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 been routing to (this can be done for DR purposes as well as for dividing a load in multiple locations to different instances). For the users everything is behind the scenes and they just connect to Artifactory through one URL.

This can be hard 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 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.

SETUP/GOAL	RECOMMENDATION
One central CI server Your 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.
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 delete any files that no longer exist on the far node. The best way to do this 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.  Edit debian-remote Repository Cache Unused Artifacts Cleanup Period (Hr): 24 Metadata Retrieval Cache Period (Sec): 43200 Assumed Offline Period (Sec): 300 Missed Retrieval Cache Period (Sec): 7200

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 paper has shown how Artifactory supports distributed programming by supporting a variety of network topologies.

With advanced features of remote repositories, virtual repositories, push / multi-push replication and 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.