Natural

Natural/HA (High Availability)

Version 9.3.2

May 2025

ADABAS & NATURAL

This document applies to Natural Version 9.3.2 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: ATUX-NHA-932-20250505NAT

Table of Contents

Natural/HA (High Availability)

Natural/HA is the feature of Natural that delivers an "always on" to enterprise applications that provide 24/7 availability to their users with negligible downtime.

Natural/HA is intended for production (not development) environments. It is also intended for interactive Natural applications, not Natural programs running in batch mode.

Natural/HA does not require specific binaries. The standard Natural binaries contain a redesigned Runtime that can execute both HA and non-HA applications, depending on the mode they are running in.

This documentation is organized under the following headings:

Why use Natural/HA?	A description of the main menu. Information on the programming modes supported by Natural. An overview of the menus available from the main menu.
Components and Operation	Design overview and principles of operation.
Porting applications to Natural/HA	What needs to be done (or not done) in order to enable applications to run in an HA environment.
Deployment of Updates	Considerations relevant to deployment of updates in and HA environment.
Terminology	Glossary of technical terms used in this documentation.

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

In addition, you can also access the cloud product documentation via **https://www.softwareag.cloud**. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

Product Training

You can find helpful product training material on our Learning Portal at **https://knowledge.softwareag.com**.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Why use Natural/HA?

Previous versions of Natural (prior to version 9.3.1), and in conjunction with Software AG's ApplinX, were able to offer high availability to a certain extent.

This has been improved significantly by Natural/HA in the following areas:

- Natural/HA provides failover capability.

  In case the backend server that is hosting the application fails, with Natural/HA it is possible to transfer the application to a different backend server.

  In previous versions, users received an error in this situation and had to restart the application.

  In contrast, a Natural/HA application supports failover by virtue of being able to implicitly save its state and reload it into a different process that could even run on a different backend server.
- Previous versions of Natural running on a backend server required a permanent connection for the duration of the user session, which could easily be for hours or more in typical real-world scenarios. This meant that if a backend server needed to be removed from the cluster, for example for maintenance reasons, the Natural processes running on the backend server would simply not drain in any reasonable amount of time. As a consequence, the server would go down and take most of the Natural processes with it.

  In contrast, a Natural/HA process for a particular user session only exists on the backend server while it performs an application's function that is triggered by the user (e.g., in response to **<Enter>** or any function key). Since these requests typically return to the user within a few seconds at most, connections can drain very quickly. Once this happens, the backend server can be taken down without any users being impacted.
- Natural/HA processes only exist briefly, to carry out a function requested by the user. This means that comparatively few Natural/HA processes are active at any given time.

  In real-world scenarios of interactive applications, most of the time is spent waiting for user input. Having fewer processes running on the server not only reduces resource demand, but

also results in less potential for negative side-effects, in the event of a system failure with processes not terminated gracefully.

- Because previous versions of Natural did not provide any failover capability, sessions previously had to be "sticky". The load balancer forwarded the requests for a particular user session to the same backend server. With Natural/HA, this is no longer mandatory.

  Furthermore, even when sticky sessions are optionally used with Natural/HA (e.g., to increase application responsiveness), these are sticky sessions *with* failover, rather than (as in previous versions) sticky sessions **without** failover. A huge advantage should the backend server become unavailable.

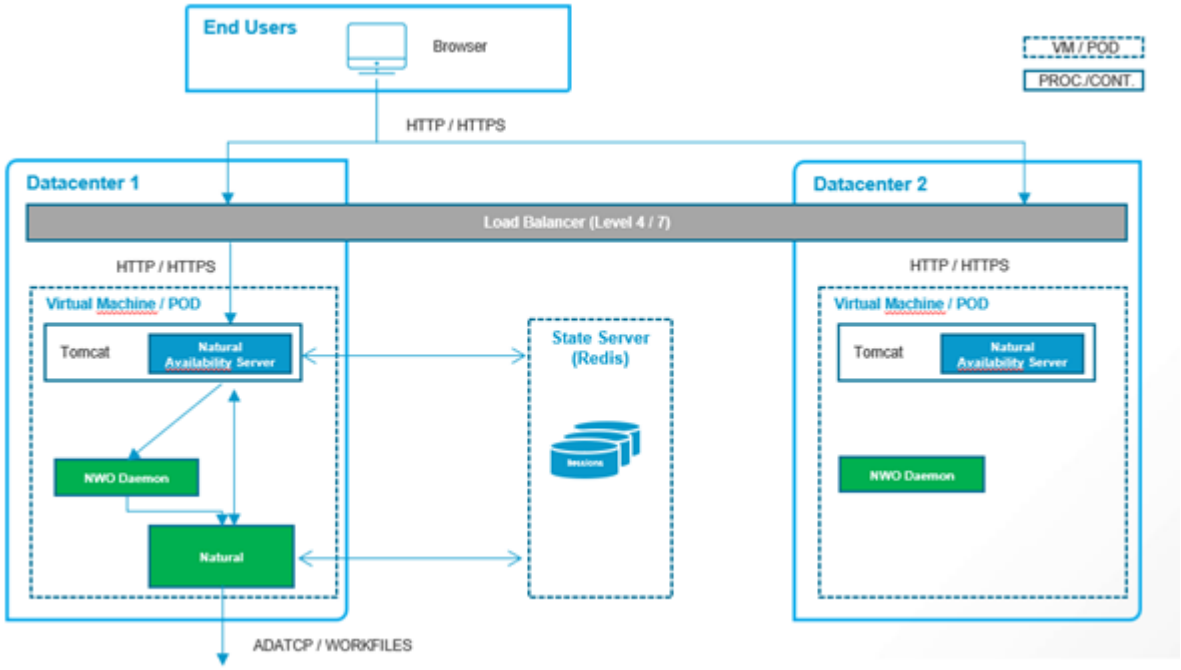
- Natural/HA provides a high degree of flexibility for performing blue-green deployment:

  This version of Natural/HA can take over user sessions from older versions. See the section ***Blue-Green Deployment of Natural/HA System Component Updates*** for more information.

- In addition to blue-green deployment, Natural/HA allows to modify Natural application libraries (FUSER) in place, while leaving the processing of most user sessions unaffected. See the section ***Blue-Green Deployment of Natural/HA Application Updates*** for more information.

3 Components and Operation

The following diagram displays the main components involved in a Natural/HA installation:



Note: The keen eye may have noticed that Natural is missing on the right hand side of the diagram ("*Datcenter 2*"). This illustrates the fact that Natural/HA processes are created on-demand and only exist temporarily. See below for more information.

Components

A Natural/HA installation consists the following components:

- Browser
- Load balancer
- Two or more backend servers containing the following components:

  o Application Server (e.g., Tomcat) / Natural Availability Server (see *Natural Availability Server*)
  Natural Web I/O (NWO) Interface Daemon
  Natural
- State Server (REDIS©)

Operation

The browser runs on the user's client machine and displays the user interface for the Natural application. The user interface is provided by the Natural Availability Server running on the application server (e.g., Tomcat) on one of the backend servers, as chosen by the load balancer.

The Natural Web I/O (NWO) Interface Daemon is responsible for starting Natural processes and must be configured to start in HA mode (see *Installing and Configuring the Natural Web I/O Interface Server*) in order to create Natural/HA processes (i.e., Natural processes running in HA mode).

Unlike a traditional Natural process, a single Natural/HA process does not exist for the lifetime of the Natural application, but terminates itself whenever it needs to perform any interactive I/O. Before doing this, it saves the state of both the Natural system and Natural application on the state server. sTthe next time a Natural/HA process is created for the same Natural/HA session, and potentially running on a different backend server, it can reload the saved state and continue from where it left off.

As a resource-friendly consequence of this design, no corresponding Natural/HA process is running on any backend server while the browser is waiting for user input.

Once the user submits input, the Natural Availability Server starts a new Natural/HA process via the NWO interface daemon and processes the user request. After the user request is processed and screen data is returned to the Natural Availability Server, the Natural/HA process saves its state and terminates. This process continues until the Natural/HA session is completed.

Lingering

The previous paragraph describes the basic operation, which can be optimized via the use of the `LINGER` parameter. If this parameter is set to a non-zero value, the Natural/HA process, after having saved its state, does not terminate itself immediately. Instead, the process waits for any possible new input from the Natural Availability Server for a specified amount of time before terminating.

This allows the Natural Availability Server to re-use the lingering Natural/HA process instead of starting a new one via the NWO interface daemon if the lingering process handled the last user request. This is always the case if the same backend server is selected by the load balancer for any particular Natural/HA session.

Lingering implies the use of sticky sessions. As upside of this constraint, lingering processes may improve application responsiveness significantly. Although lingering increases the number of processes running on the server side, in the case of Natural, these lingering processes do not consume significant CPU resources. For the application, the performance advantage may outweigh the penalty on the CPU's resource.

Note: The state server is used by both Natural and the Natural Availability Server and needs to be configured separately for both components, although both can refer to the same location (e.g., REDIS© Enterprise database).

4 Porting applications to Natural/HA

To start using a Natural application in an HA environment, the following aspects must be considered:

- Adabas must be accessed by configuring the TCP/IP as described in the *Adabas ADATCP Access* section.
- Transactions for other databases (e.g., SQL) cannot cross I/O boundaries. This is because it is not possible to save and restore the state for such databases before and after an I/O occurs.
- If the application performs printing or sorting, or uses work files that are read or written across I/O boundaries, the *HA_FILESHARE* path (see under *Local Configuration File > Installation Assignments*) should be defined to a highly available directory that is accessible from all backend servers. Both, printing and sorting use the *HA_FILESHARE* directory (if defined) implicitly when Natural is running in HA mode. The same applies to work files without any path information. However, for work files used across I/O boundaries, it is necessary to adapt any explicit paths to point to the shared location. Note that the internal environment variable `$HA_FILESHARE` can be used to represent the *HA_FILESHARE* directory when defining work file locations (e.g., *$HA_FILESHARE/wrk001.txt*). Work files that are **not** used across I/O boundaries can continue to use non-shared locations.
- Natural/HA requires a state server. By default, if a *HA_FILESHARE* path has been defined (see above), Natural/HA sessions will be saved there. Otherwise, it is possible to define one or more explicit state servers (e.g., REDIS© enterprise databases) and select which one of them should be used via the `STATESRV` profile parameter.
- If setting up a read-only buffer pool, the object *NATURAL.CXSD* in the *RES* subdirectory of the [FNAT] SYSTEM library should be included in the preload list. This file contains the schema (i.e., data structure) common to all saved Natural/HA sessions and is needed to be able to both read and write them.

Note: Because work on this topic is continuing, with improvements being frequently backported to this version, it is strongly recommended to keep up with any product updates (e.g., fix levels) when making use of this functionality.

5 Deployment of Updates

Once a Natural/HA application is up and running, one of the most relevant considerations is how to apply updates with minimal impact to end users. We can distinguish between two types of updates:

- Natural product updates, i.e., deploying newer versions of Natural
- Natural application updates, i.e., deploying new versions of the Natural application

The following subsections offer strategies related to the above topics.

Blue-Green Deployment of Natural/HA System Component Updates

Saved Natural/HA sessions created by older Natural versions/fix levels can be taken over by newer Natural versions/fix levels. The same applies to Natural Availability Server sessions. The NWO daemon is stateless.

Therefore, updates to any of these three principal Natural/HA system components can be applied via a traditional blue-green deployment scenario. In this case, the updates are "hot-swapped", meaning that Natural/HA sessions will pick up the changes without having to be restarted.

Note: The state server must be the same in the blue and green environments. This ensures that any sessions saved by the blue version can be found by the green version.

Blue-Green Deployment of Natural/HA Application Updates

The approach fow system components updates (as described above) can also be used to a limited extent for Natural application (FUSER) updates. A common scenario would be a continuous integration (CI) process, where application updates are applied frequently (e.g., every day). In the cae of a CI process, potentially involving thousands of application objects, only a small fraction is typically modified during any one CI interval.

Only the Natural/HA sessions that happen to use one of the modified objects at precisely the time of being saved are likely to be lost. All other sessions will continue unchanged, which will often be the vast majority. The worst-case scenario would be when one of the objects that is always active (e.g. the main program object) is changed, resulting in the loss of all running sessions. Nevertheless, this approach may represent a good trade-off between ease of deployment and degree of session takeover in many real-world scenarios.

Blue-Green Deployment with Connection Draining

This deployment strategy is a modification of traditional blue-green deployment where data traffic over an existing connection continues to be forwarded to the original ("blue") backend server for a grace period during a blue-green deployment, even though it is no longer part of the new ("green") backend pool.

In addition, and as an exception, subsequent connections for sticky sessions are typically also routed to the original backend server during this grace period. The default value for the grace period is usually long enough to allow any active Natural/HA processes in the blue environment to run to completion without the connection being closed.

The exception for sticky sessions ensures that any sticky sessions closed by the user within the grace period are not affected, even if they would not have been taken over by the green environment.

This, together with a potential improvement in responsiveness via the `LINGER` parameter, are two considerations for wanting to use sticky sessions in a Natural/HA environment, despite this not being strictly necessary.

For non-sticky sessions, connection draining does return any benefits, because a new connection is opened for each subsequent user operation. All of these operations will be directed to the "green" environment, regardless of whether the grace period has expired.

6 Terminology

The following terminology is used in the Natural/HA documentation.

Natural/HA
Natural/High Availability. The "/HA" suffix intends to emphasize features only relevant to a high availability (HA) context.

HA Mode
The mode of operation of Natural required to support high availability. This is set in the configuration of the Natural Web I/O (NWO) Interface Daemon responsible for starting Natural processes.

Natural/HA Application
A Natural application running in high availability (HA) mode.

Natural/HA Session
A running instance of a Natural/HA application. Each session has its own unique session ID.

Active Natural/HA Session
A Natural/HA session that is currently processing an operation requested by the user.

Inactive Natural/HA Session
A Natural/HA session that has finished processing the last requested operation and is currently waiting for the next user input.

Saved Natural/HA Session
An object representing the state of an inactive Natural/HA session residing on the state server. These saved objects have a base name containing the session ID and the extension *.nssd* (Natural saved session data). These objects expire by default, after a user-configurable period determined by the `SSTTL` parameter. Expired objects are eligible for automatic removal at any time for state servers supporting this functionality. This prevents obsolete (e.g., orphaned) sessions from continually building up on the state server over time.

Natural/HA Session Schema

An object representing the data structure of a saved Natural/HA session. Although the actual data values and their number of occurrences can vary wildly from session to session, the structure of the data is constant for a particular Natural/HA version and fix level.

Storing this information in a separate schema object avoids repeating this information in each saved Natural/HA session object, allowing the latter to become much more compact.

Session schemas are stored in objects of proprietary format with the extension *.CXSD* (Compiled XML Schema Description). The native session schema (*NATURAL.CXSD*) is stored in the resource (*RES*) directory of the FNAT SYSTEM library and is cached and accessed via the buffer pool. For this reason, it should be present in the preload list when setting up a read-only buffer pool for a Natural/HA application.

Note: Session schemas are also stored on the state server, under a versioned name, depending on the Natural version and fix level to which it corresponds. This allows saved Natural/HA sessions to be taken over by a different version of Natural than the version that created them, facilitating blue-green deployment. Unlike saved Natural/HA sessions, Natural/HA session schemas never expire on the state server.

Natural/HA Process

The operating system's process that is currently hosting an active Natural/HA session for the purpose of processing an operation requested by the user.

After the processing of this operation has completed, the state of the Natural/HA session is saved, and the process terminated after a user definable `LINGER` time has expired. When the next user operation is requested, the lingering process (if still available) is re-used, or a new process (potentially on a different backend server) is instantiated and the previously saved Natural/HA session restored into it.

Note: While the Natural/HA session is inactive (see above), it is not being hosted by any process on any backend server. This differs from the traditional non-HA behavior, where a Natural/HA session is hosted by a single process for its entire lifetime.

Backend Server

A single node belonging to a cluster (or pool) of server nodes accessed by a browser-based frontend via a load balancer. Each backend server typically contains an application server hosting the Natural Availability Server, the NWO daemon (used for instantiating Natural/HA processes), and Natural/HA itself. In Natural/HA production environments, at least two backend servers are required to continue to serve the Natural/HA application should one of the servers become unavailable for any reason.

State Server

The configured location where the state of inactive Natural/HA sessions or Natural Availability Server sessions are saved. This location must be accessible to all backend servers. In addition, this location must provide high availability when used in a production environments, so as

not to itself become a single point of failure (SPOF). For example, a REDIS© Enterprise database in the Cloud fulfills these requirements.