IKANALM

IKAN ALM Architecture

Closing the Gap
Enterprise-wide Application Lifecycle Management

Table of contents

Summary

IKAN ALM is a server-agent Web Application Lifecycle Management (ALM) solution with a web-based Administrative console, and consisting of a number of main ALM services and a number of integrations with other ALM tools.

All IKAN ALM actions are defined and executed through the Web application or the Commandline Interface (CLI).

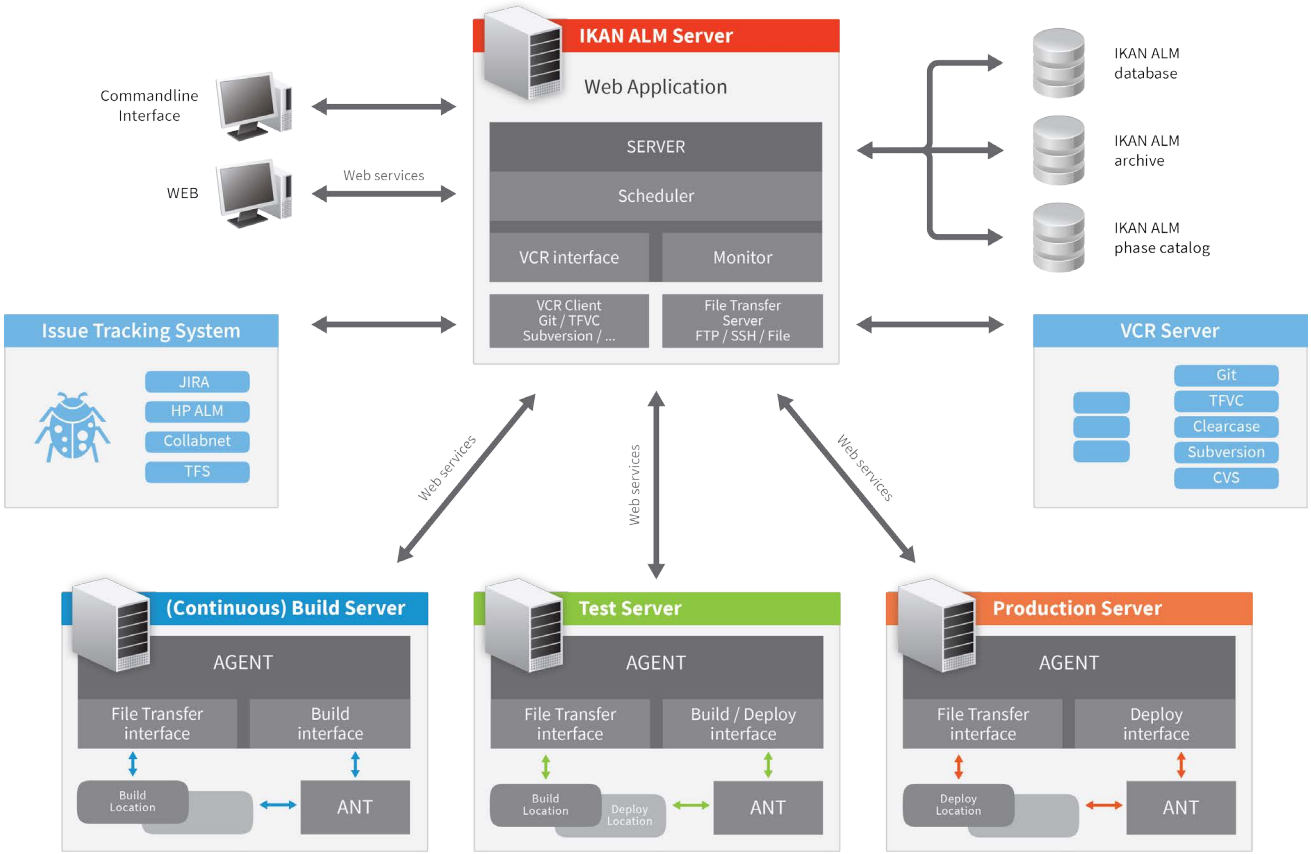
The Web application is used for defining the Global and Project Administration items and the User's personal Desktop

Main ALM services that IKAN ALM provides are:

- Life Cycle
- Build
- Deploy
- Approval
- Notifications

Integrations offered with other ALM tools are for:

- Version Control Repositories
- Requirements, Issue Tracking or Defect Tracking systems
- Build tools
- Deploy tools
- Reporting tools



The Global Administration is used for defining the system settings, version control repository access, user groups, machines, transporters (FTP, FileCopy, Secured Shell),scripting tools (Ant, Maven or NAnt) and issue tracking systems (e.g., JIRA, HP ALM, Collabnet or other).

The Project Administration is used for defining the project with its lifecycle and levels (one or more Build, Test and Production levels).

IKAN ALM actions are called Level Requests. A Level Request consists of a Build Request or a Deploy (to Test or Production) Request. Level Requests are created through the Web Interface or the Commandline Interface.

All Global and Project administration information is stored in a JDBC-compliant database, such as MySQL, Oracle, DB2 or SQL Server. It also contains the log of scripting actions executed while handling Level Requests. The actual Build Results are stored in an archived format (*.zip or *.tar.gz) in the Build Archive on the IKAN ALM Server, so that they can be retrieved for later Deploy Level Requests.

The Web application runs on a Web or application server (preferably Apache Tomcat) on the IKAN ALM server machine. The same machine also runs the SERVER daemon process that handles the Build or Deploy Level Requests. The Level Request handling consists of running different steps, called Phases, such as the communication with the external systems (Versioning, Issue Tracking, …) and the interaction with the IKAN ALM Agent(s) via Web services to handle the actual Build and Deploy actions. As the Server process is OSGi-aware (see IKAN ALM SERVER Architecture) IKAN ALM has the possibility to not only execute Core Phases, but also Custom Phases that it retrieves from the Phase Catalog.

An IKAN ALM agent is also an OSGi-aware daemon process (see IKAN ALM AGENT Architecture) which transfers sources (in case of Build) or a build result (in case of Deploy) from the Server Machine, executes a script by a Scripting Tool (ANT, Maven,…) and (in case of a Build) archives the Build result to the Build Archive on the Server. Next to those Core Phases, it can also run Custom Phases, whereby it retrieves the phases from the phase catalog as defined on the SERVER Machine.

From a user's perspective, the Phase Catalog and the Core and Custom Phases you find there, are a key IKAN ALM feature.
For each phase executed by the SERVER or an AGENT, an entry exists in the Phase Catalog.

IKAN ALM comes with Core SERVER and AGENT Phases and any user can define and customize phases into the Phase catalog.

In the Catalog, each phase consists of a physical JAVA Archive (JAR) on the SERVER file system and related metadata information of the phase. Custom phases are added and managed through the Web interface.

The phase's metadata information is used by both the SERVER and AGENT when a phase is executed. The phase name and version are key parameters to identify a phase, whereby the versioning number allows you to execute different versions of the same phase on the same SERVER or AGENT(S).

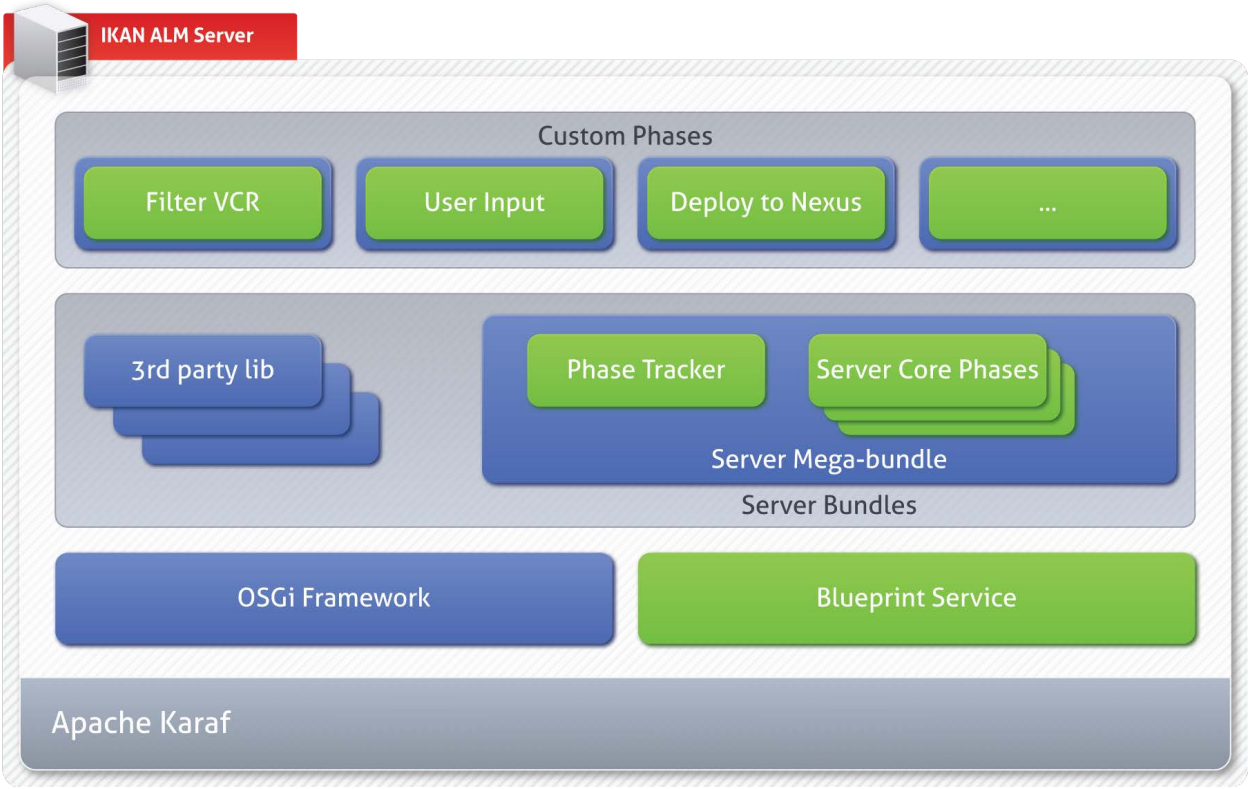
IKAN ALM SERVER Architecture

The SERVER is a Java process (Daemon) with a Monitor and Scheduler sub process. It is executed in a JAVA Virtual Machine (JVM) and it handles Level Requests given by a user through the Web interface or initiated through the Commandline Interface.

The monitor process steers the interaction with the Version Control Repositories when a Level Request is executed (check-out of source code, tagging of source code) and it will communicate with the AGENTS to make sure that the Build and Deploy processes are executed correctly (see IKAN ALM AGENT Architecture).

Support for Continuous Integration, Nightly Builds or Continuous Delivery is handled by the Scheduler process. The Scheduler will trigger the Version Control Repository to see if source code has been changed. This project-based triggering is based on pre-defined moments (every night at 8PM) or time intervals (every hour, every 5 minutes, …)[1].

[1] Note that Continuous Integration may also be realized by a hook script that calls the IKAN ALM Commandline interface when a commit is done in the versioning system.



IKAN ALM Server OSGI

The SERVER is OSGi-aware and runs within Apache Karaf, a light weight OSGI runtime container. Karaf supports the Blueprint Service[2] Component Frameworks, that we selected and use. This Blueprint Service gives us the ability to make our Phases pluggable.

Other benefits the Karaf container brings are the integrated JAAS security framework, the Feature Service, i.e., the provisioning system that allows us to install and update Phases automatically (see Interaction between the IKAN ALM SERVER/AGENT and the Phase Catalog), the OS integration (runs as a Windows service or Linux daemon) and the possibility of hot deployment of OSGi bundles.

The SERVER code is packaged in an OSGi "Server Mega-bundle" wherein a number of key Blueprint Components are declaring: the Phase Tracker and the Server Core Phases. The Core phases are the basic building blocks for handling Level Requests: checking-out of source code from a Version Control Repository, the notification and management of Agents in order to ensure proper execution of Builds and Deploys, tagging of source code in the VCR.

Each of these phases is registered as a Blueprint Service when the Server mega-bundle is started and is kept by the Phase Tracker that traces the Blueprint Phase services.

The Server mega-bundle imports about 60 third party libraries that all are translated to correct OSGi Bundles. Most of them come from the SpringSource Enterprise Bundle Repository[3]; others have been transformed by us as they were not available in a public repository.

Finally, the Phase Tracker will also register the Custom Phases: these are the Phases that execute a specific script (Ant, NAnt or Maven), which are defined in the Phase Catalog by the user himself and that are used to execute specific processes (e.g., to interact with an external archive like Nexus, Jenkins or TFS) on the SERVER for a Level Request.

[2] http://wiki.osgi.org/wiki/Blueprint
[3] http://ebr.springsource.com/

Each of these phases resides in a separate OSGi bundle that declares one Blueprint Service. At installation or update, the Phase Tracker will pick it up and the Server mega-bundle will consume it when it handles a Level Request that uses that specific Custom Phase.

For the installation and update of the Custom Phases (and also of the Core phases) from within the Phase Catalog, the Karaf feature mechanism is used (see Interaction between the IKAN ALM SERVER/AGENT and the Phase Catalog).

IKAN ALM AGENT Architecture

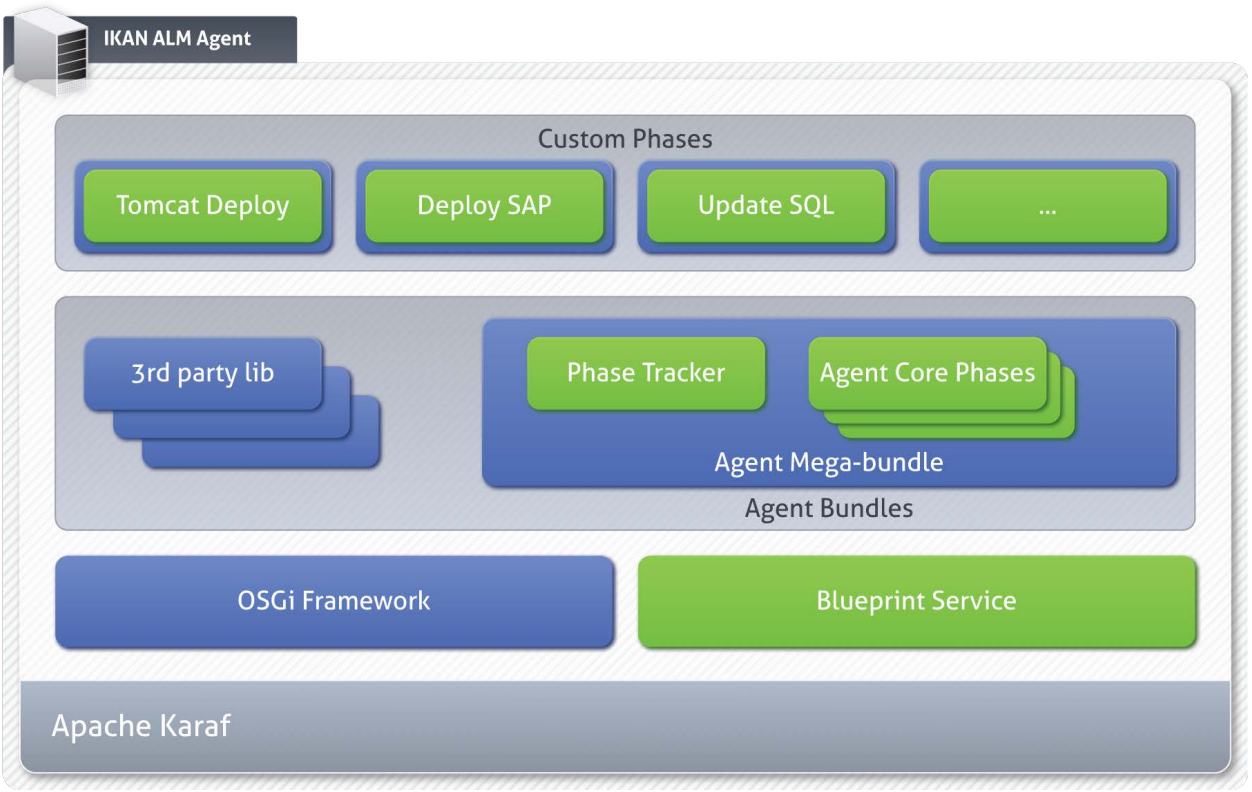
The AGENT is a Java process (Daemon) with a Build and Deploy sub process. It is executed in a JAVA Virtual Machine (JVM) and it handles Builds and Deploys that are initiated by the SERVER.

This process is so-called ”local” when it is executed on the same physical machine as the SERVER. When it runs on a different physical machine, it is called a “Remote” agent. The AGENT interacts remotely with the Monitor process of the SERVER (via Web services) and locally with a Transporter (FileCopy, FTP of SSH) and a scripting tool (Ant, NAnt or Maven), whereby these scripting tools need to be configured correctly in the IKAN ALM Web application and on the AGENT machine.

Like the SERVER, the AGENT is OSGi-aware. The AGENT also runs within Apache KARAF, giving the same benefits to the AGENT as it gives to the SERVER. The Agent code is packed into an OSGi “Agent Mega-bundle” where a number of key Blueprint Components are declared: the Phase Tracker and the Agent Core Phases. The Core phases are the basic building blocks for handling Level Requests: transport of source (build) or a build result (deploy) verification of a build or deploy script, execution of Builds and Deploys.

Each of those phases is registered as a Blueprint Service when the Agent mega-bundle is started and is kept by the Phase Tracker that traces the Blueprint Phase services.

Finally, the Phase Tracker will also register the Custom Phases: these are the Phases that execute a specific script (e.g., for an Update of a database, or a Deploy to a Web server) that eventually are defined in the Phase Catalog by the user himself and that are used in a Build



IKAN ALM Agent OSGI

or Deploy environment that is linked to an AGENT.

Each of those phases resides in a separate OSGi bundle that each time declares one Blueprint Service.

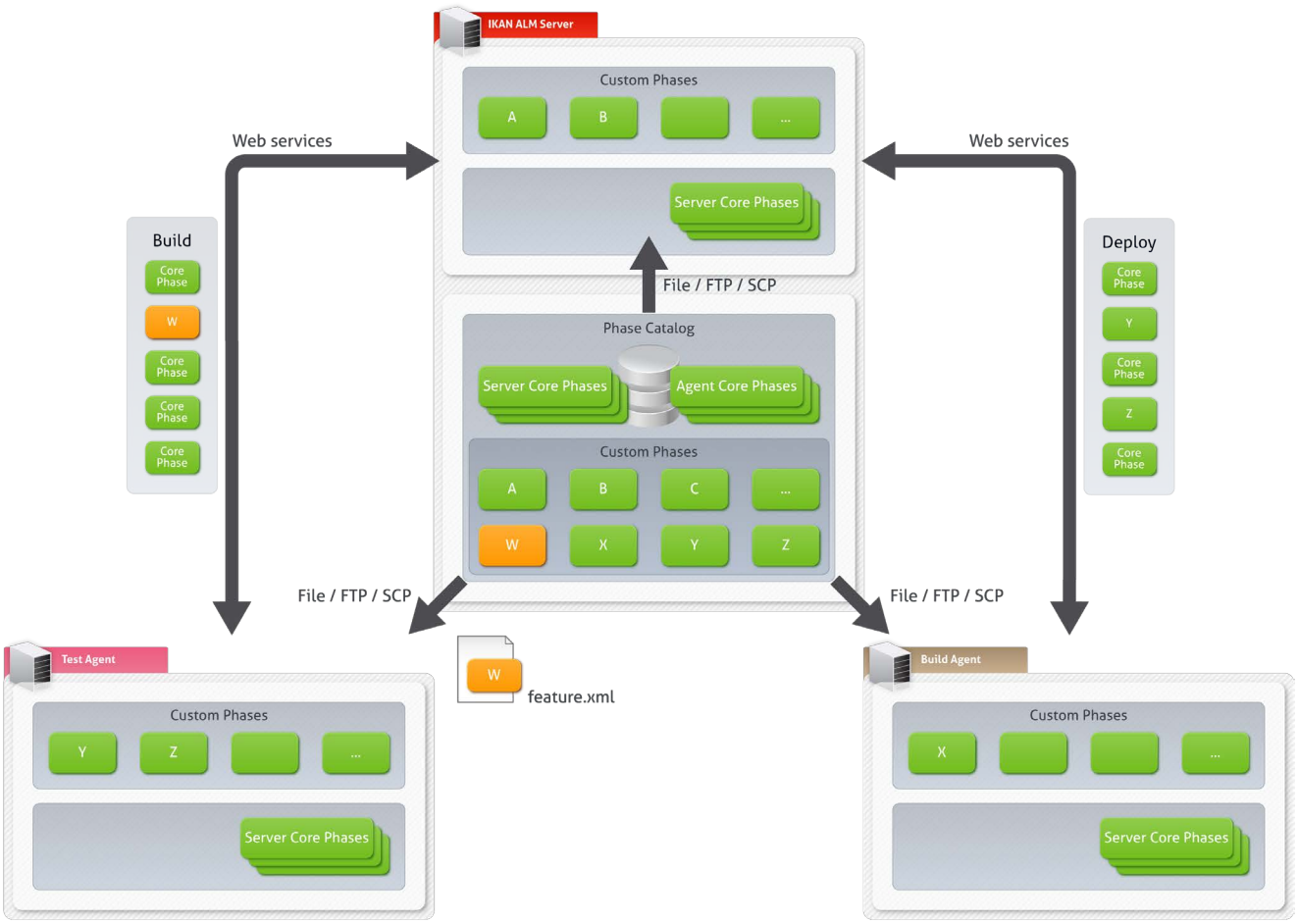
For the installation and update of the Custom Phases (and also the Core phases) from within the Phase Catalog, the Karaf feature mechanism is used (see *Interaction between the IKAN ALM SERVER/AGENT and the Phase Catalog*).

Interaction between the IKAN ALM Server/Agent and the Phase Catalog

The following figure describes how the AGENT and SERVER interact with the Phase Catalog when executing a Level Request with a Build and Deploy action. The focus is put on the distribution of a missing Custom Phase during the execution of a Build by an AGENT. This process consists of the following sequential steps.

The following steps happen before what is shown on the figure:

1. The SERVER Monitor process starts a Level Request.
2. The SERVER collects a list of phases that need to be executed from the database
3. This list may contain Server Core Phases and Custom Phases (for example: A, B).
4. The SERVER consults the Phase Tracker component to find the OSGi service objects from the Server phases. (We assume that all the phases from the Phase Tracker list are registered. If not, the same process as the one for the Agent applies).



IKAN ALM Server/Agent and Phase Catalog interaction

5. The SERVER starts the sequential execution of the phases. One of the Core Phases is a Build phase: The SERVER notifies the AGENT via a Web service that he needs to execute a Build.

Now the figure from above comes into play:

6. The AGENT Build process picks up the Build request

7. The AGENT asks the SERVER a list of phases it needs to execute.

8. The AGENT gets a list of phases it needs to execute, whereby the list can contain both Agent Core Phases and Custom Phases.

9. The AGENT consults its Phase Tracker component to retrieve the OSGi service objects from the phases. All Core Phases are found, the Custom Phase W has not been registered yet. As a consequence, the Build process is temporarily stopped.

10. The AGENT/Phase Tracker asks the SERVER to install the missing Custom Phase W.

11. The SERVER reacts by sending an XML file with the phase description and an URL that can be used by the Phase Tracker to install the missing phase in the AGENT Karaf Container. The XML file is a "Karaf feature file[4]", with a URL that refers to the location of the correct version of the missing phase W in the Phase Catalog. The URL has as file type FTP or SSH, depending on the Transporter that has been defined for that specific AGENT (Machine).

12. The AGENT passes the feature file to the Karaf Feature Service. The Feature Service gets the Custom Phase bundle from the Phase Catalog by using the given URL and installs it in the OSGi framework. The Blueprint component framework detects that the Custom phase bundle contains a Blueprint descriptor, and registers the phase as an OSGi service. As a result, the Phase is registered in the Phase Tracker.

13. When the Build process starts again, the AGENT consults the Phase Tracker again to find the OSGi service objects for this phase. All Core Phases are found, including the missing Custom Phase W.

14. The AGENT starts the sequential execution of the phases for this specific Build.

[4] https://cwiki.apache.org/confluence/display/KARAF/4.6.+Provisioning

Glossary

Build	A Build is an action on a Build Environment which involves several sub processes, called Build Phases. It is always part of a Level Request, which may also contain other Builds or Deploys. [A Build is handled by the IKAN ALM Builder]. It starts from sources that were retrieved from the VCR to the Build Environment. A Build script is executed on those sources by a Scripting Tool and gives a Build Result that is transferred to the Build Archive.
Build Archive	The physical location (path) on the IKAN ALM Server where the Build Results are stored in a compressed and archived format (*.zip or *.tar.gz). The Build Results are organized by Project and by Project Stream.
Build Phase	A Build Phase is a sub process that must be performed to complete a Build action. Different Build Phases form the workflow of a Build and are inserted into a Build Environment. They are executed by the IKAN ALM Builder Thread of the IKAN ALM Agent. A Build Phase may be a Core Phase (e.g. Verify Build Script), or a Custom Build Phase created or imported by the User in the Phase Catalog.
Build Tool	Scripting Tool installed on a Build Environment

Core Phase	Core Phases form the IKAN ALM "Core" functionality. They can only be viewed, and cannot be altered nor deleted. Consider them an integral part of IKAN ALM. When a new Level, Build or Deploy Environment is created, its default workflow will be created and will completely consist of a sequence of Core Phases. This default workflow may be changed by removing Core Phases, by changing the sequence order, or by adding Custom Phases.
Custom Phase	A Phase added by the User is also called a "Custom" Phase. It may be created from scratch in Global Administration based on one or more working scripts and resources, or it may be imported, using the "Import Phase" functionality. Once defined in Global Administration, a Custom Phase may be inserted into (and consequently change) the default work flow of a Level, Build or Deploy Environment. All Custom Phases are stored in the Phase Catalog on the IKAN ALM Server, and are transported automatically to the IKAN ALM Server (Level Phase) or IKAN ALM Agent (Build or Deploy Phase) when they are to be executed.
Deploy	A Deploy is an action on a Deploy Environment which involves several sub processes, called Deploy Phases. It is always part of a Level Request, which may also contain (an)other Build(s) or Deploy(s). [A Deploy is handled by the IKAN ALM Deployer]. It starts from a Build Result which is retrieved from the Build Archive. A Deploy script is executed on this Build Result by a Scripting Tool.
Deploy Phase	A Deploy Phase is a sub process that must be performed to complete a Deploy action. Different Deploy Phases form the workflow of a Deploy and are inserted into a Deploy Environment. They are executed by the IKAN ALM Deployer Thread of the IKAN ALM Agent. A Deploy Phase may be a Core Phase (e.g. Transport Build Result) or a Custom Deploy Phase created or imported by the User in the Phase Catalog.
Deploy Tool	A scripting Tool installed on a Deploy Environment
IKAN ALM Agent	A process (daemon) running on a Machine with sub processes to handle Build or Deploys. An Agent running on the same Machine as the IKAN ALM Server is also referred to as "local", whereas running on a different Machine it is indicated as "remote". During a Build or Deploy the IKAN ALM Agent interacts remotely with the IKAN ALM Monitor, and locally with a Transporter and with a Scripting Tool that must be correctly configured on the Machine.
IKAN ALM Server	The Machine hosting the IKAN ALM web application and the IKAN ALM Monitor and Scheduler processes.
Issue Tracking	A system external to IKAN ALM, where Issues (defects, enhancements, tasks, …) may be defined for a Project. Samples are Atlassian JIRA, HP Quality Center, Collabnet TeamForge, Bugzilla or Trac. IKAN ALM can plug in to such a System and keep up with the Issues that were handled for a Level Request. The integration with JIRA, HP Quality Center and TeamForge is more advanced: Issues are automatically synchronized through the lifecycle, and it is possible to keep a link with the Level Requests in the JIRA Issue, HP Quality Center Defect or TeamForge Artifact.

Level Request	A Level Request is an action on a Level which involves several sub processes, called Level Phases. In most cases, a Level Request will contain at least one Build or Deploy action, which will be executed on local or remote Machines. A Level Request may be created manually by the user via the Web interface or the Command Line interface, or automatically by the Scheduler Thread of the IKAN ALM Server. A Level Request is handled by the Monitor Thread of the IKAN ALM Server.
Life-Cycle	A Life-Cycle is a sequence of Levels that is linked to a Project Stream. It enables to set up the step-by-step process to promote sources and build results from development, to test, QA, ... to end up into production. One Project may have different Life-Cycles, e.g. for development on the next release, for maintenance or urgency fixes on the release currently in production, for parallel development,… A Life-Cycle may be reused in more than one Project Stream.
Phase Catalog	The physical location (path) on the IKAN ALM Server where the Custom Phases (created from scratch or imported) are stored in an archived format (Phase.name-Phase.version.jar, e.g. com.ikanalm.echoproperties-1.0.0.jar). When an IKAN ALM Server or Agent needs to install a missing Custom Phase, it will be retrieved from that location. That will be done using the Transporter linked to the Server or Agent Machine.
Project	An IKAN ALM Project maps to a project or subproject in a versioning system (VCR) which bundles related sources. An IKAN ALM Project is a shell for one or more Project Streams in which the real actions (Level Requests, Builds, Deploys) are done. It is possible to set up dependencies between different Projects, also through the Project Streams. There are 2 types of Projects: Release-based Projects: IKAN ALM will work with the existing structure in the VCR system, so that the objects to be extracted will be retrieved automatically when starting the build process. Package-based Projects: this concept enables to work with isolated files from the VCR system. Objects must be selected manually in a Package structure created in IKAN ALM before starting the Build process.
Scripting Tool	A system external to IKAN ALM which can execute user-created scripts and which is installed on a Machine. IKAN ALM integrates with ANT, NAnt and Maven2. When the Scripting Tool is linked to a Build respectively Deploy Environment it is also referred to as a Build respectively Deploy Tool. The script for executing a Build or Deploy must be stored in the VCR (together with the sources) or in the Script Location on the IKAN ALM Server.
Transporter	A Transporter is used for transporting files and directories between the IKAN ALM server and a local or remote Agent handling the Build or Deploy processes. Therefore, a Transporter must be defined for a specific Machine that is linked to the Build or Deploy Environment. IKAN ALM supports the local FileCopy, remote FileCopy, SecureCopy and FTP Transporters. A Transporter may transport checked-out sources from the Versioning System, a Build result from the Build Archive, but also Custom Phases from the Phase Catalog.

Version Control Repository (VCR)	An external versioning system holding different versions of sources. Related sources are bundled in a Project or subproject (sometimes also called a Module). A VCR Project may contain different development streams, called head (=main or trunk) or branch streams. IKAN ALM integrates with the following VCR's: CVS, Subversion, Microsoft Visual SourceSafe, IBM ClearCase and Serena PVCS. In order to connect to the VCR, a VCR Client must be installed on the IKAN ALM Server and correctly configured. The IKAN ALM Monitor interacts with the VCR by retrieving or tagging sources. The web interface interacts with the VCR to show revision numbers, modified sources, … related to a Level Request.

For More Information

To know more, visit http://www.ikanalm.com
Contact IKAN Development: info@ikanalm.com

IKAN Development (Belgium)
Kardinaal Mercierplein 2
2800 Mechelen, Belgium
Tel. +32 15 797306

IKAN Development (France)
3, Rue du Général De Gaulle
28700 Aunay-Sous-Auneau, France
Tél: +33 2 37 25 31 22

info@ikan.be
www.ikan.be