IBM uDeploy® Architecture Guide

5.0

IBM uDeploy Architecture Guide: 5.0

Publication date September 2013

Documentation notices for IBM uDeploy

Programming interfaces: Intended programming interfaces allow the customer to write programs to obtain the services of IBM uDeploy.

Copyright license

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

If you are viewing this information in softcopy, the photographs and color illustrations may not appear.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use persistent cookies that collect each user’s user name and password for purposes of authentication and enhanced user usability. These cookies can be disabled, but disabling them will also likely eliminate the functionality they enable. This Software Offering uses session cookies for purposes of session management. These cookies cannot be disabled.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's privacy policy at [http://www.ibm.com/privacy], IBM's Online Privacy Statement at http://www.ibm.com/privacy/details/us/en, including the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Trademark acknowledgments

IBM, the IBM logo, and IBM are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at www.ibm.com/legal/copytrade.shtml [http://www.ibm.com/legal/copytrade.shtml].

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Architecture Overview

IBM uDeploy architecture consists of a service tier and a data tier. The service tier has a central server that provides a web server front-end and core services, such as workflow, agent management, deployment, inventory, security, as well as others. A service can be thought of as a self-contained mechanism for hosting a piece of business logic. Services can be consumed by clients\agents or other services. Deployments are orchestrated by the server and performed by agents distributed throughout the network. Most clients use browsers to communicate with the web server via HTTP(S). Most server-agent communication is done via JMS (discussed below) but HTTP(S) is also used as required.

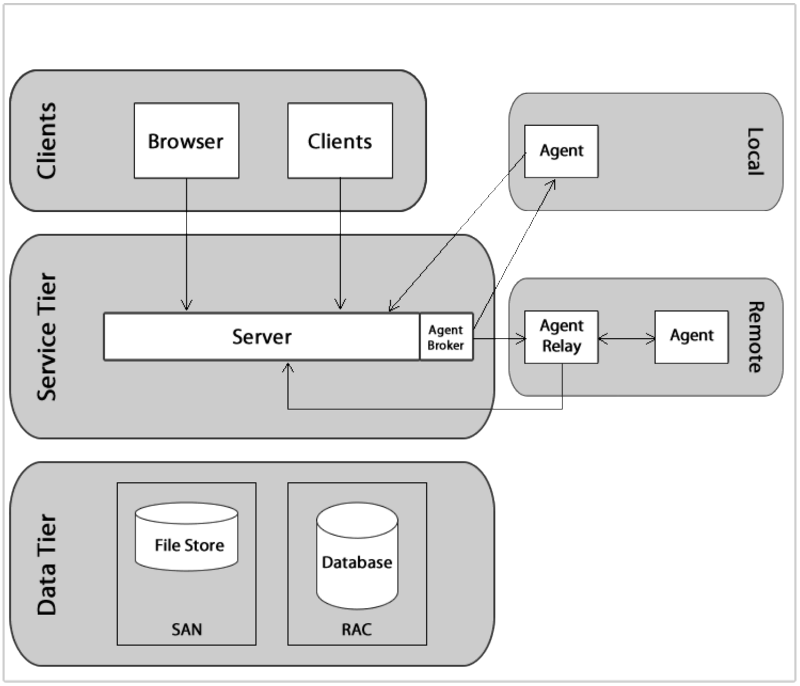
IBM uDeploy uses *stateless* communications for server-agent communications (JMS-based) as well as client-web service communications. Stateless, as used here, means the server retains little session information between requests and each request contains all the information required to handle it. The server sets-up listening sockets and listens for agents, relays, and users (clients). For added security, agents do not listen on ports. Agents send requests when they are ready to make the transition to a new state.

Server-agent communication is built around transferring—deploying—components. Components can contain any business-meaningful content, such as environment information, configuration data, source, static files, or anything else associated with a software project. Because JMS connections are persistent and not based on a request-response protocol, IBM uDeploy does not have to continually open and close ports, which enables the server to communicate with agents at any time while remaining secure and scalable.

Many IBM uDeploy services are REST-type (representational state transfer). REST-style services are web services that focus on transferring resources over HTTP. A resource can be any business-meaningful piece of data. Resources are transferred by a self-describing format such as XML or JSON (JavaScript Object Notation). The XML and JSON representations typically model resource states at the time of agent/client requests. REST-style services achieve statelessness by ensuring that requests include all the data needed by the server to make a coherent response.

The data tier's relational database stores configuration and run-time data. The data tier's file store—CodeStation—contains log files, artifacts, and other non-structured data objects. Reporting tools can connect directly to the relational database.

Figure 1. Overview



Service Tier

The IBM uDeploy server provides a variety of services, such as: the user interface, component and application configuration tools, workflow engine, and security services among others. The REST-based user interface provides the web-based front-end that is used to create components and fashion workflows; request processes, and manage security and resources, among other things.

When a workflow is requested, many services are used to fulfill the request, which are shown in the following illustration:

Figure 2. Services and Process Workflow

Workflow requests are initiated with the user interface, either the web-based application or the CLI (command line interface).

Table 1. Services

Service	Description
User Interface	Used to create components and fashion workflows, request processes and manage security and resources, among other things. REST-based.
Workflow Engine	Manages workflows—application and component processes. Calls the agent responsible for performing the workflow's current plug-in step. When the workflow is finished, alerts the notification and inventory services. Called by the deploy service. REST-based.
Agent	Tracks installed agents and routes plug-in commands to affected agents. Commands come from plug-in steps. Invoked by the workflow service. REST-based.

Service	Description
Work Item	Operates in tandem with the approval service; provides approver alerts and enables approvers to accept or reject workflows. If a scheduled workflow remains unapproved at run-time, the job fails automatically. REST-based.
Plug-in Manager	IBM uDeploy can interact with virtually any system through its extensible plug-in system; plug-ins provide functions by breaking-down tool features into automated steps. Plug-ins can be configured at design- and run-time. When a plug-in step executes, the controlling agent invokes its run-time process to execute the step. When a new component version is available, the agent compares the current component version and downloads and only new or changed artifacts.
Event	The event service is ubiquitous; it alerts other services as various trigger conditions occur.
Deployment Service	Manages deployments. When a deployment process is requested, invokes the workflow engine to perform the process. Works in tandem with the security service to ensure users have required permissions. REST-based.
Notification Manager	Notifies users about the status of deployments; notifications are sent to approvers if the system is configured with an email server and the user has an email address. Invoked by the workflow manager. REST-based.
Inventory Manager	When a workflow finishes, the inventory manager updates affected inventory records. IBM uDeploy maintains an inventory of every deployed artifact in every environment, which provides complete visibility across environments. REST-type service.
Approval Engine	Enables creation of approval-requiring jobs and approver roles. Works in tandem with the work item service to ensure required approvals are made before scheduled jobs. REST-based.
Security	Controls what users can do and see; maps to organizational structures by teams, roles, activities, etc. REST-based.
Calendar	Used to schedule processes to being at some future point; works in tandem with the approval and work item services. REST-based.
CodeStation	Handles versioning of artifacts; agents invoke it when downloading component versions. REST-based.

Clients

Web browsers are IBM uDeploy's most common client (agents are discussed in another topic, see the section called “Agents”) but other clients can be developed to access the web services. Clients are deployed locally (on the same LAN as the IBM uDeploy server) or remotely, and communicate with the server via HTTP or HTTPS. The IBM uDeploy browser-based GUI is a Rich Internet Application (RIA) that maintains much of its functionality in the browser. Clients interact with RESTful (representational state transfer) services on the server as needed. A command line client is available that provides most of features found in the browser-based GUI. The command line client is also built on top of RESTful services.

Data Tier

Relational Database

Your relational database is a critical element for performance and disaster recovery. The provided Derby database, while sufficient for proof-of-concept work, is generally insufficient for the enterprise. Full-featured databases like Oracle, MS SQL Server, or MySQL are better options. Ideally, the database —whichever is used—should be configured for high-availability, high-performance, and be backed-up regularly.

10-20 GB of database storage should be sufficient for most environments. For Oracle, an architecture based on Oracle RAC is recommended; for Microsoft SQL Server, a clustered configuration is preferred; for MySQL, utilize MySQL Cluster.

File Storage—CodeStation

The data tier also provides log file and Codestation artifact storage. Artifacts represent deployable items such as files, images, databases, configuration materials, or anything else associated with a software project. By default, these are stored in the `var` subdirectory in the IBM uDeploy server installation directory. In an enterprise environment, the default installation might not be ideal, see the section called “Relocating Codestation” for a discussion about enterprise options.

IBM uDeploy's secure and tamper-proof artifact repository ensures that deployed components are identical to those tested in preproduction environments. Without the repository, artifacts would have to be pulled from network shares or some other system, increasing both security risks and the potential for error.

The artifact repository uses content addressable storage to maximize efficiency while minimizing disk use. The repository tracks file versions and maintains a complete history for all components. Maximizing efficiency is important, since the artifact repository stores files that are much larger than source files. Association of files with Components is built into the system. Without any configuration, each Component gets its own area of the repository for its files. There is no chance of confusion or mix-up of files to Components. And, each Component Package is mapped to a specific set of files and versions corresponding to the Component.

The artifact repository comes with a client application that provides remote access to the repository. Using the client, the user can add/modify files, create Packages, retrieve files, as well as view the history of changes. The client application provides a file transfer capability that can be used to deliver files to target servers during deployments. This built-in transfer mechanism verifies the integrity of all transferred files against their expected cryptographic signatures, thus guaranteeing that files have not been corrupted during transmission or tampered with during storage. In addition to the client application, the artifact repository exposes REST-based web services. These services are used to build integrations between build systems such as AnthillPro and IBM uDeploy. Such integrations automatically place the artifacts produced by the build process in the artifact repository, thus making the artifacts available for deployment.

Relocating Codestation

By default, the data tier's log files and Codestation artifacts are stored in the `var` subdirectory within the IBM uDeploy server directory. Ideally, this data should be stored on robust network storage that is regularly synchronized with an off-site disaster recovery facility. In addition, the IBM uDeploy server should have a fast network connection to storage (agents do not need access to the storage location). In Unix environments, you can use *symbolic links* from the `var` subdirectory to network storage. On Windows platforms there are several options for redirecting logs and artifacts, including `mklink` (supported in Windows 7 and later).

If you want to relocate Codestation, relocate both the `var` directory as well as the `\logs\store` directory. A good rule-of-thumb for determining Codestation storage requirements is: *`average artifact size * number of versions imported per day * average number of days before cleanup`*

Distributed teams should also take advantage of IBM uDeploy location-specific Codestation proxies to improve performance and lower WAN usage.

Data Center Configuration

This section provides several installation recommendations.

Cold Standby

IBM uDeploy employs the cold standby HA strategy for the application tier. When the primary system fails, the cold standby is brought online and promoted to primary server. Once online, the standby reestablishes connections with all agents, performs recovery, and proceeds with any queued processes. Because the most intense work is handed-off to agents, a high performance configuration should not have an agent installed on the same hardware as the main server.

The IBM uDeploy server aggressively utilizes threading and takes advantage of any additional CPU cores assigned to it. A small to midrange server with 2-4 multi-core CPUs is ideal, but, because it is relatively easy to move an existing IBM uDeploy server installation to a new machine, starting small and scaling as needed is a very legitimate strategy. The memory available to the application tier should also be increased from the default 256 MB to something on the order of 1 GB.

Platform Considerations

IBM uDeploy agents are platform agnostic, and can be installed on anything that provides a Java 1.5 JDK. The server process is also platform agnostic. Our customer base includes large IBM uDeploy installations on Windows, Solaris, AIX, HP-UX, other Unix flavors and various Linux platforms, all running successfully.

We have seen somewhat better performance from Unix and Linux operating systems, but recommend installing on the platform with which you are most familiar and comfortable.

Recommended Server Installation

- **Two server-class machines**

  IBM recommends two machines for the server: a primary machine and a standby for fail-over. In addition, the database should be hosted on a separate machine.

- **Separate machine for the database**

- **Processor** 2-4 CPUs, 2+ cores for each.

- **RAM** 8 GB

- **Storage** Individual requirements depend on usage, retention policies, and application types. In general, the larger number of artifacts kept in IBM uDeploy's artifact repository (CodeStation), the more storage needed.

- **Network** Gigabit (1000) Ethernet with low-latency to the database.

Agent Minimum Requirements

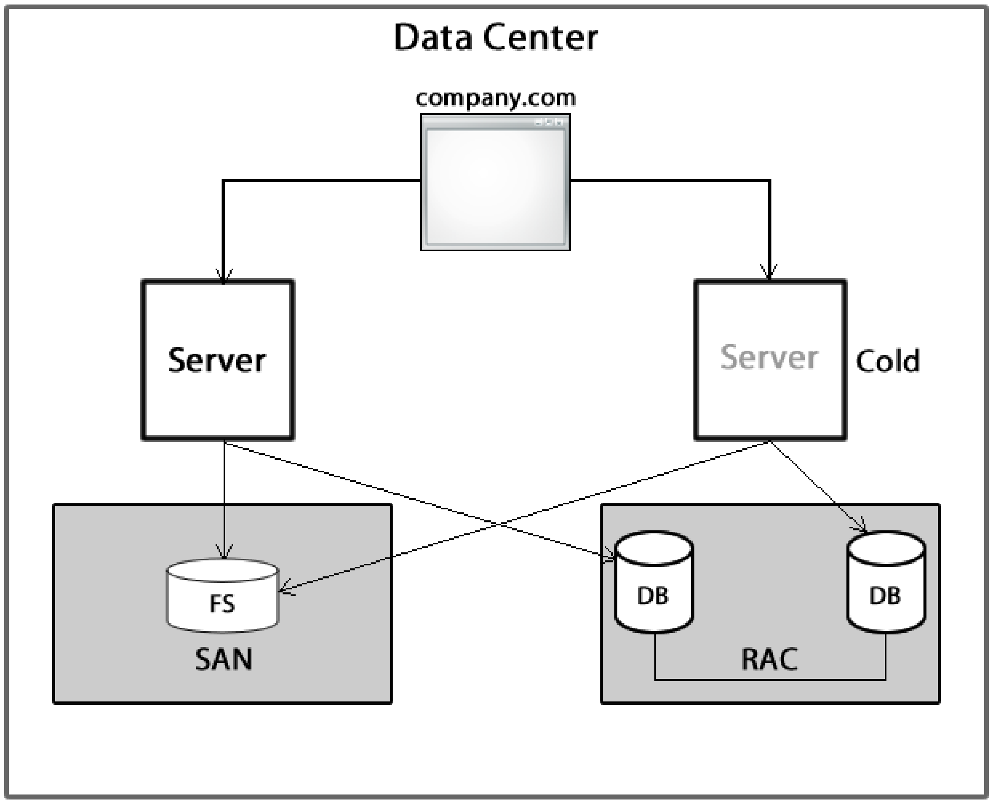
Designed to be minimally intrusive, agents require 64-256 MB of memory and 100 MB of disk space. Additional requirements are determined by the processes the agent will run. Agents should be installed on separate machines. For evaluation purposes, a good option is to install an agent on a virtual machine.

Typical Data Center Configurations

Most organizations configure the data tier with network storage and a clustered database. The service tier performs best when it's on a dedicated, stable, multi-core machine with a fast connection to the data tier. A standby machine should be maintained and kept ready in case the primary server goes down.

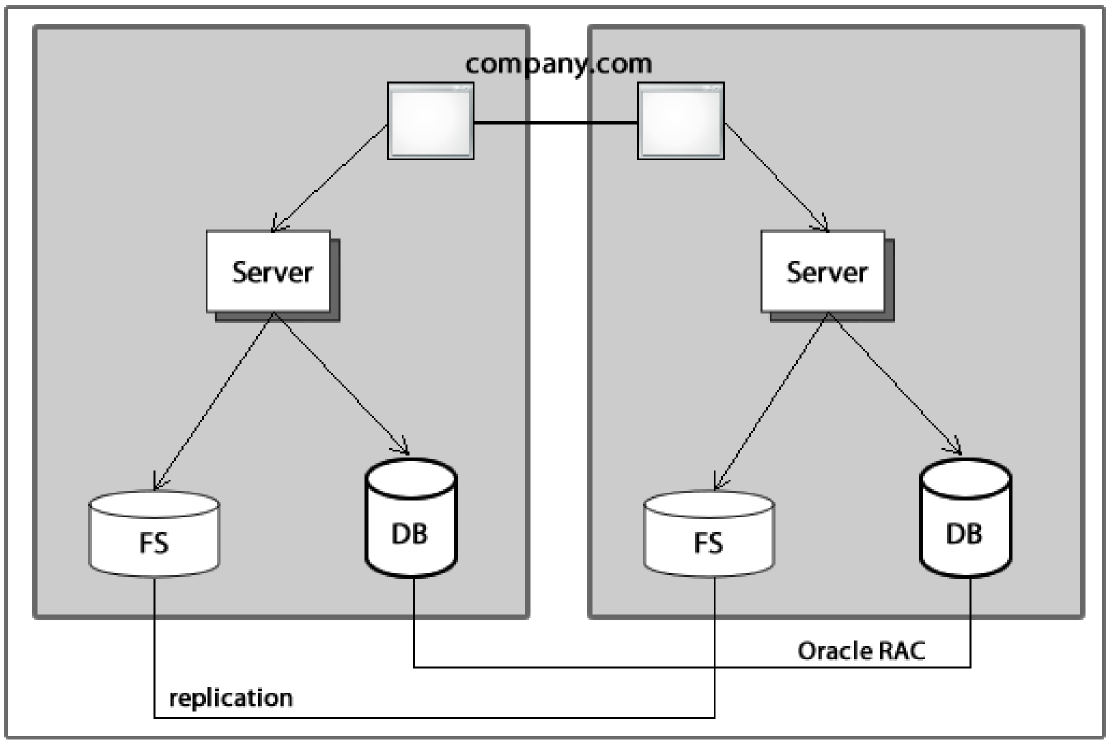
The following figures illustrate typical IBM uDeploy configurations.

Figure 3. Single Data Center Configuration



There are no remote agents or agent relays in this configuration.

Figure 4. Multiple Data Centers



Recovery Using a Database Back-up

To prepare a back-up of your IBM uDeploy installation, copy the database and server files. Back-up the server by copying the server directory along with all subdirectories. This ensures that you can revert to a server version that matches your configuration, while also preserving your artifact repository.

If you are not using the Derby database, copy the database too. If you are using Derby, it will be copied along with the server files.

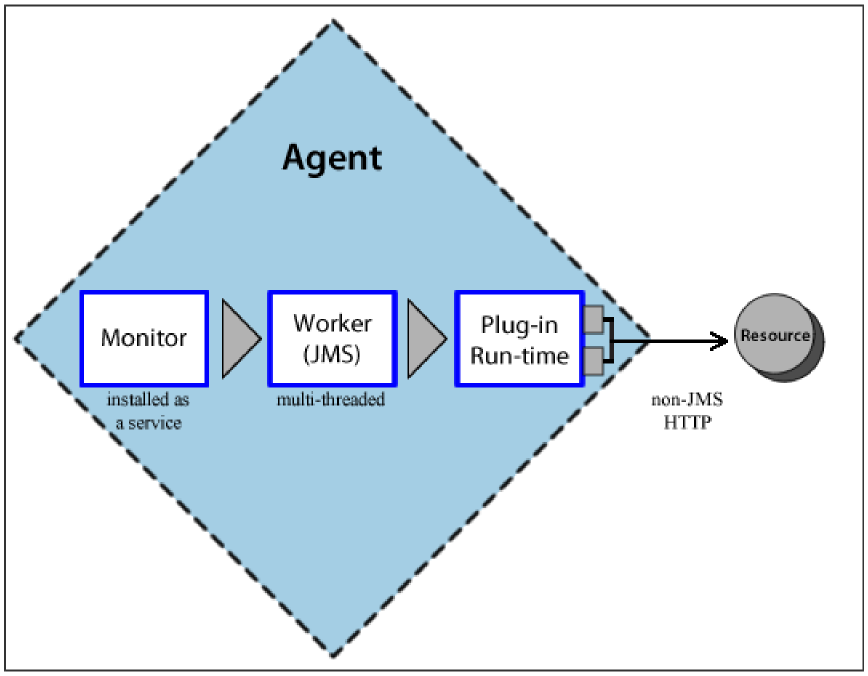
Install the back-up using the original path, or some configuration files will need to be changed.

Agents

Agents play a central role in the IBM uDeploy architecture. An agent is a lightweight process that runs on a deployment-target host and communicates with the IBM uDeploy server. Agents perform the actual work of deployment which relieves the server from the task. All processes—packaging, configuration, deployments, and so on—requested by the IBM uDeploy server are executed on hardware assigned to agents. Once an installed agent has been started, the agent opens a socket connection to the IBM uDeploy server. Communication between server and agents uses a JMS-based (Java Message Service) protocol and can be secured using SSL, with optional mutual key-based authentication for each end-point. This communication protocol is stateless and resilient to network outages (the benefits of statelessness are discussed below).

While we characterize an agent as a single process, technically an agent consists of a *worker* process and a *monitor* process. The worker is a multi-threaded process that performs the actual deployment work after receiving commands from the server. Work commands come from plug-in steps which provide seamless integration with many third-party tools. The monitor is a service that manages the worker process--starting and stopping, handling restarts, upgrades, and security, for example. Agents are rarely upgraded because their functionality is derived from plug-ins, which can be upgraded at will. Once an agent is installed, it can be managed from the IBM uDeploy web application.

Figure 5. Agent Processes



Agents are an important part of IBM uDeploy's scalability. By adding more agents, the throughput and capacity of the system increases almost exponentially and so can scale to fit even the largest enterprise.

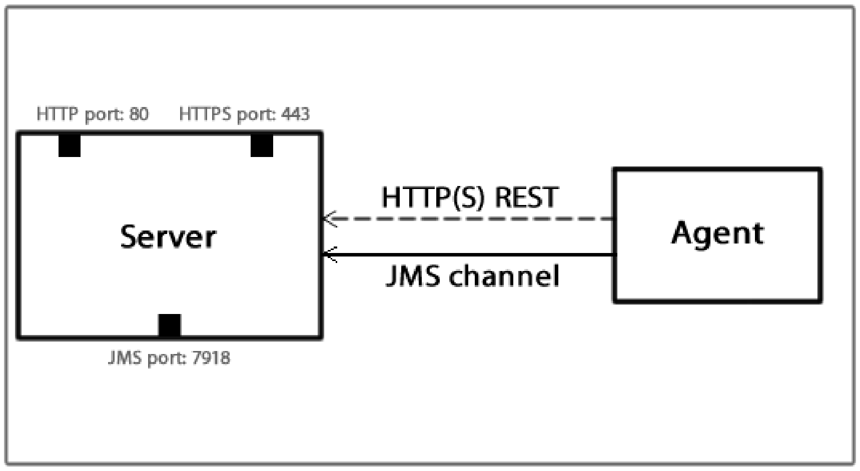
Server-Agent Communication

Most agent communication is done with JMS, but some agent activities—posting logs, transmitting test results, or posting files to CodeStation, for example—use the web tier via HTTP(s) as needed. The JMS channel is IBM uDeploy's primary control channel; it's the channel the server uses to send agent commands. By default the server listens on only three ports: port 7918 for JMS, 8080 for HTTP, 8443 for HTTPS.

The agent monitor service uses JMS for all server communications and for sending commands, such as "run step," to the worker process. The worker process uses JMS for system communications, and HTTP REST services when performing plug-in steps or retrieving information from the server.

Stateless server-agent communication provides significant benefits to performance, security, availability, and disaster recovery. Because each agent request is self-contained, a transaction consists of independent message which can be synchronized to secondary storage as it occurs. Either endpoint--server or agent—can be taken down and brought back up without repercussion (other than lost time). If communications fail mid-transaction, no messages are lost. Once reconnected, the server and agent automatically determine which messages got through and what work was successfully completed. After an outage, the system synchronizes the endpoints and recovers affected processes. The results of any work performed by an agent during the outage are communicated to the server.

Figure 6. Stateless Communication



In Figure 6, “Stateless Communication”, the arrow represent the direction in which communications was established, but the flow can be in both directions with JMS.

Remote Agents--Crossing Network Boundaries and Firewalls

IBM uDeploy supports *remote agents*—cross-network deployments. As long as there is at least a low bandwidth WAN connection between the server and remote agents, the IBM uDeploy server can send work to agents located in other geographic locations. To aid performance and ease maintenance, IBM uDeploy uses *agent relays* to communicate with remote agents. An agent relay requires that only a single machine in the remote network contact the server. Other remote agents communicate with the server by using the agent relay. All agent-server communication from the remote network goes through the relay. Agent relays can be configured as CodeStation proxies in order to optimize the transfer of large objects.

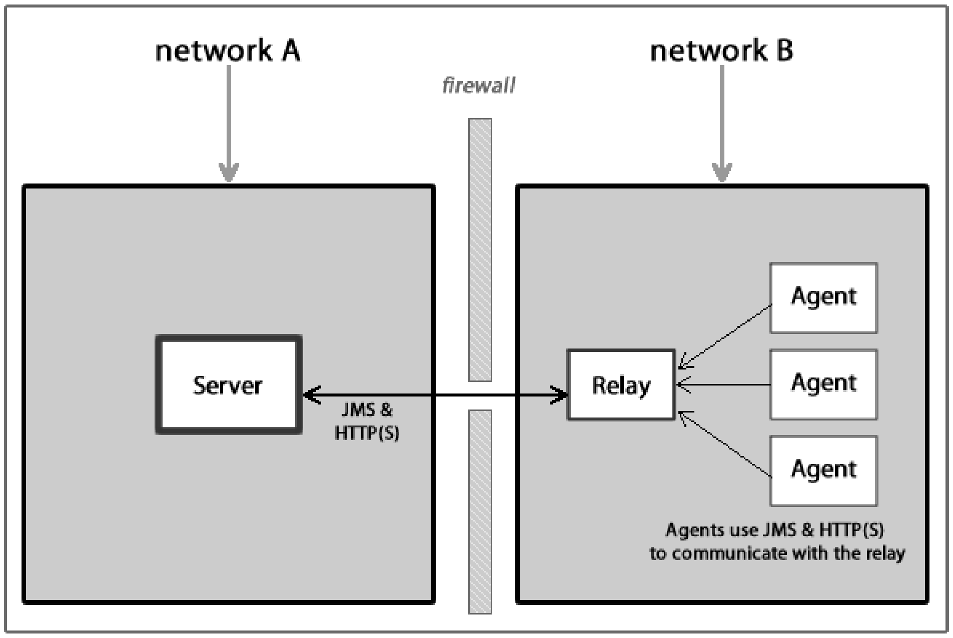
The following, a simple artifact move, illustrates the mechanics of remote communications:

1. A remote agent starts and establishes a connection to the agent relay via JMS, which, in turn, alerts the IBM uDeploy server via JMS that the remote agent is online.

2. The server sends, say, an artifact download command to the relay via JMS, and the relay delivers the message to the remote agent (also via JMS).

3. The server locates the artifacts, and then:
   - Uploads the artifacts to the relay over HTTP(s), which begins streaming them directly to the agent over the server-relay HTTP(s) connection.
   - Once the remote agent completes the work, it informs the server via JMS.

Figure 7. Crossing Network Boundaries



By default, agent relays open the connection to the IBM uDeploy server, but the direction can be reversed if your firewall requires it. Remote agents open connections to the agent relay.

In configurations with relay agents, agents local to the IBM uDeploy server continue to use direct communications.

Agent Security

IBM uDeploy agents employ user impersonation when required to perform tasks for which they would not otherwise have permission. To run a database update script, for example, an agent might need to be the "oracle" user; but to update the application, the agent might need to be the "websphere" user. By using impersonation, the same agent can run the script and update the application, which enables you to combine these steps into a single process. For information about user impersonation, see the section called “User Impersonation”

User Impersonation

IBM uDeploy can use *user impersonation* when an agent must execute a command for which it might not otherwise have permission, or when a specific user must be employed for a given process. On Unix/Linux systems the *su/sudo* commands are used to impersonate users; on Windows IBM uDeploy provides a utility program to handle impersonation. You implement impersonation when you configure a component's plug-in process step.

Using *su/sudo*

The `su` command (as used by IBM uDeploy) enables a user to start a shell as another user (process steps can be considered individual shells). When you configure a process step (see ???), you can tell IBM uDeploy to use impersonation for the step. By default, `su` is used but you can use `sudo` instead. To configure impersonation, you supply the user name required by the target host. When the impersonation-configured process step runs, the *su* or *sudo* command runs the step as the impersonated user. Each step that needs user impersonation must be configured independently.

Before `sudo` can be used, impersonation privileges must be defined in the `/etc/sudoers` file. When you configure `sudoers`, ensure that the impersonating user does not have to supply a password. Typically, you would configure the `/etc/sudoers` file like this:

```
Defaults:X !requiretty

X ALL=(Y) NOPASSWD: ALL
```

where X and Y are user names. Configured this way, user X can run any command as user Y without supplying a password.

`su` and `sudo` maintain a record in the system logs of all of their activity. `su` can be used without configuring the `sudoers` file. For information about `su/sudo` see the Unix/Linux documentation.

Note

For Unix- or Linux-based agents the password option is ignored.

Impersonation on Windows Systems

For agents running on Windows platforms, IBM uDeploy provides a program that handles impersonation. You implement impersonation for Windows-based agents the same way you do for Unix- or Linux-based agents: when you configure a process step, you specify the *local user* credentials—user name and password —that will be used when the step is processed. For impersonation purposes, a local user is one whose user name and password are stored on the target computer and who is part of the administration group and has, at a minimum, the following privileges:

- `SE_INCREASE_QUOTA_NAME` (adjust memory quotas for a process)
- `SE_ASSIGNPRIMARYTOKEN_NAME` (replace a process-level token)
- `SE_INTERACTIVE_LOGON_NAME` (local logon)

You can also impersonate the Windows LocalSystem account. The LocalSystem account is installed on every Windows machine and is the equivalent of the root user on Unix/Linux. It is guaranteed to have the privileges listed above.

Note

For Windows-based agents the `sudo` option is ignored if selected.

SSL Mutual Key-based Authentication

SSL (Secure Socket Layer) technology enables clients and servers to communicate securely by encrypting all communications. Data are encrypted before being sent and decrypted by the recipient—communications cannot be deciphered or modified by third-parties.

SSL technology can be used in several modes. In *unauthenticated mode*, communication is encrypted/decrypted but users do not have to authenticate or verify their credentials. By default IBM uDeploy uses this mode for its JMS-based server/agent communication. By default, JMS-based communication uses port 7918.

SSL unauthenticated mode can also be used for HTTP communication. You can implement this mode for HTTP during server/agent installation, or activate it afterward. See ???.

In *mutual authentication mode*, communications are encrypted as usual, but users are also required to authenticate themselves by providing digital certificates. A *digital certificate* is a cryptographically signed document intended to assure others as to the identity of the certificate's owner. IBM uDeploy certificates are self-signed.

When mutual authentication mode is active, IBM uDeploy uses it for JMS-based server/agent communication. In this mode, the IBM uDeploy server provides a digital certificate to each agent, and each agent provides one to the server. This mode can be implemented during server/agent installation, or activated afterward. See ??? for information about activating this mode and exchanging certificates between the server and agents.

Unauthenticated mode for HTTP and mutual authentication mode for JMS are optional; you can implement one without implementing the other, or implement both.

High Availability

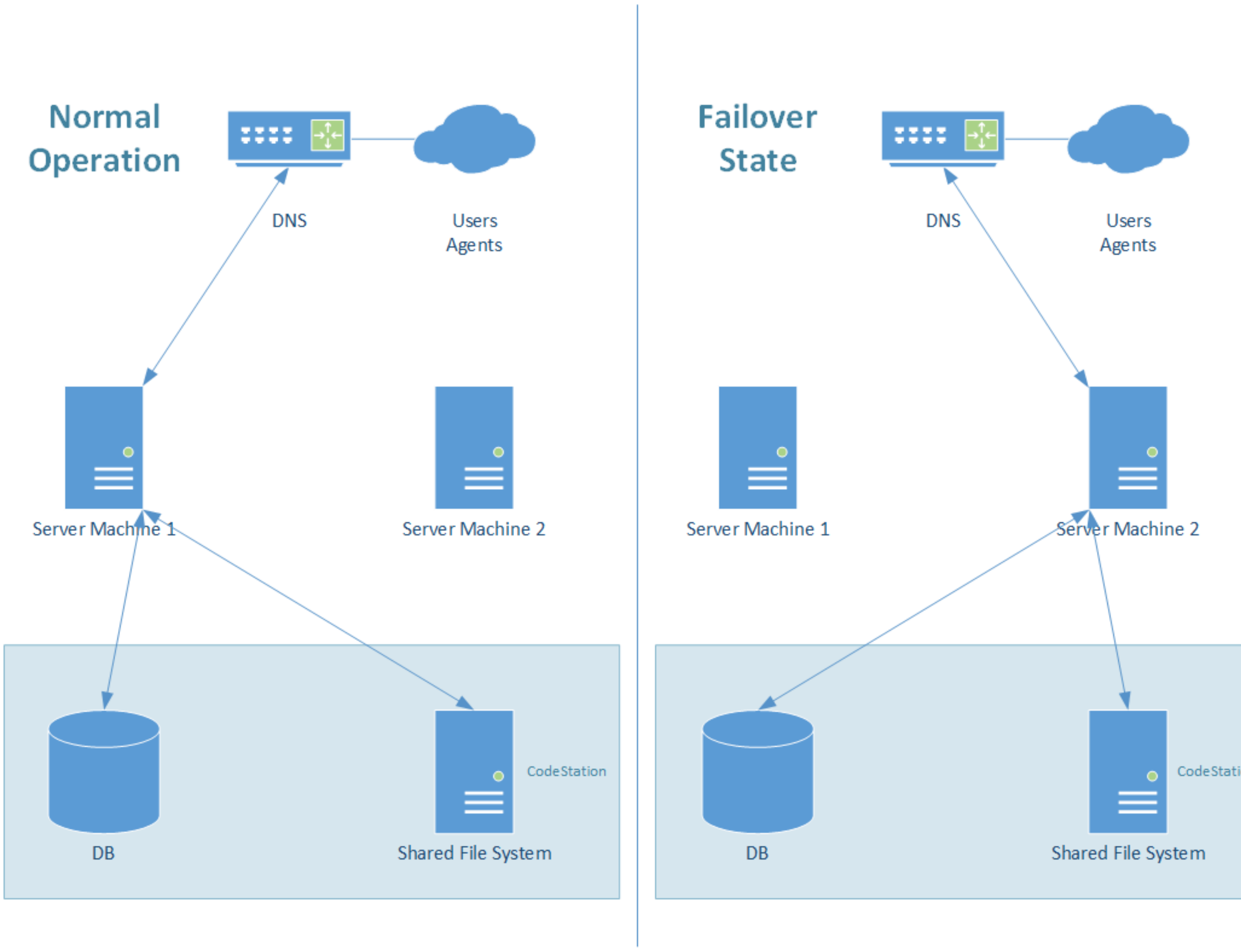
Options

There are two options to ensure IBM uDeploy services are uninterrupted: cold standby, and clustering.

Option 1: Cold Standby

The cold standby option is simple to implement, reduces downtime near to zero, but does not improve IBM uDeploy performance. In this setup, there is a single active IBM uDeploy server (primary server) connected to a database and a remote file system. There is also a secondary IBM uDeploy server configured to connect to the same database and file system, but IBM uDeploy is not running. If the active node fails, the secondary server is started and network traffic is routed to it. This is called *failover*. IBM uDeploy has no automatic process to failover, but it can be automated.

Figure 8. Cold Standby

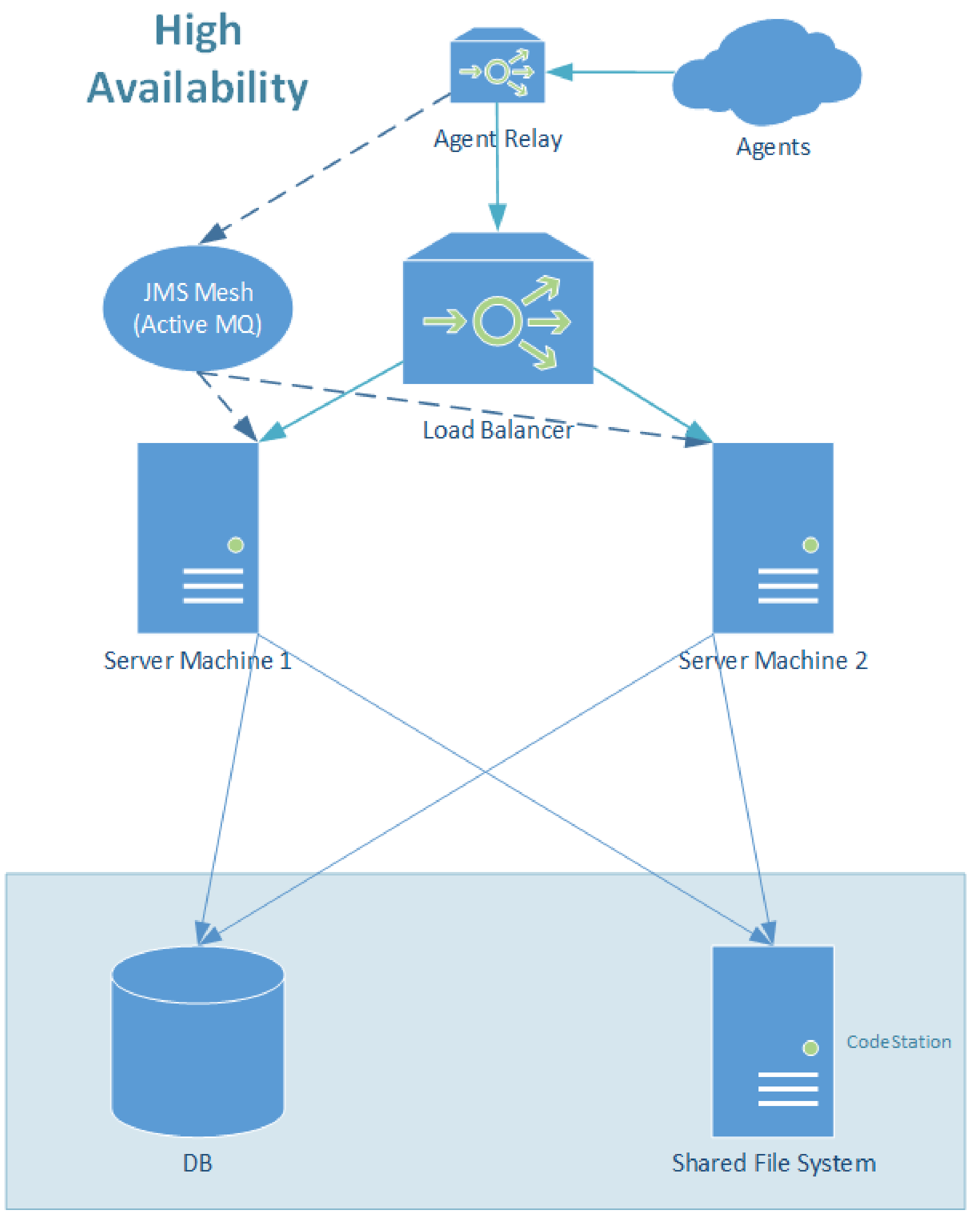


Option 2: Clustered

The IBM uDeploy high availability (HA) feature increases scalability and availability by distributing processing across *n* number of servers. Each server is an independent node cooperating in common processing. The goal is to be as fault-tolerant as possible while requiring little or no manual intervention.

The IBM uDeploy servers create a JMS mesh (via ActiveMQ); all servers know about each other. All services are alive on each server.

Figure 9. HA Design



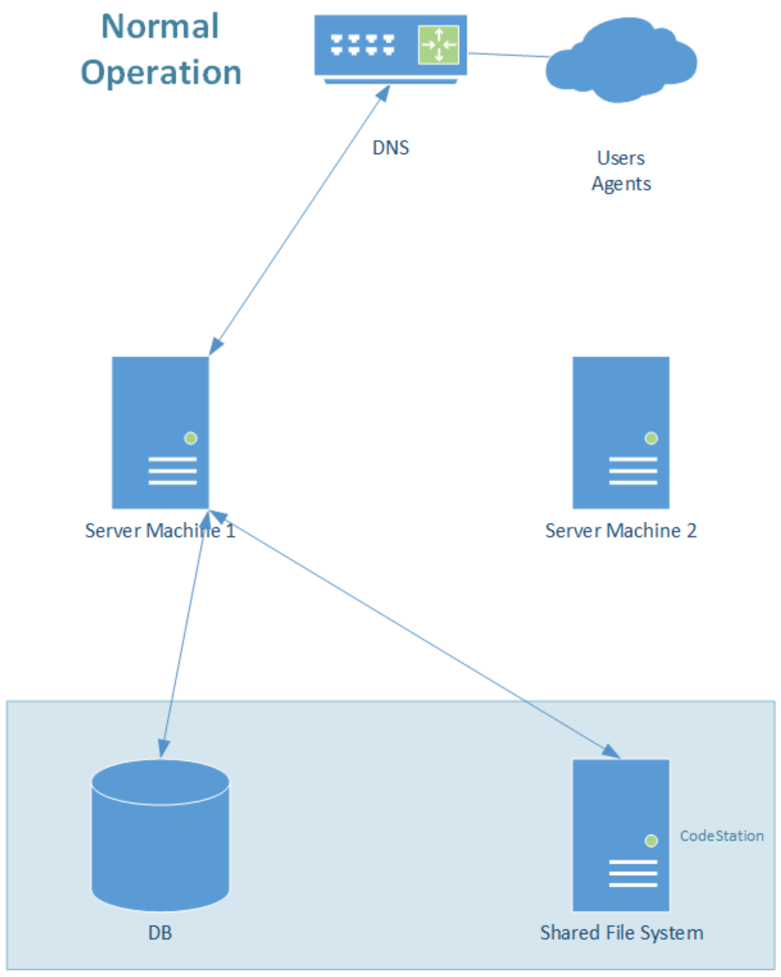
Note

The clustering feature is part of a suite of features scheduled for the IBM uDeploy 5.0 release, and is provided in the 4.8 release for those designers and architects interested in the feature.

Converting a Cold Standby Installation to Clustered

Typically, in a cold standby configuration, the IBM uDeploy server shares the database and file system with a standby server. During a disruption, the DNS routes the traffic to the standby. A typical configuration is shown in the following illustration.

Figure 10. Cold Standby Topology

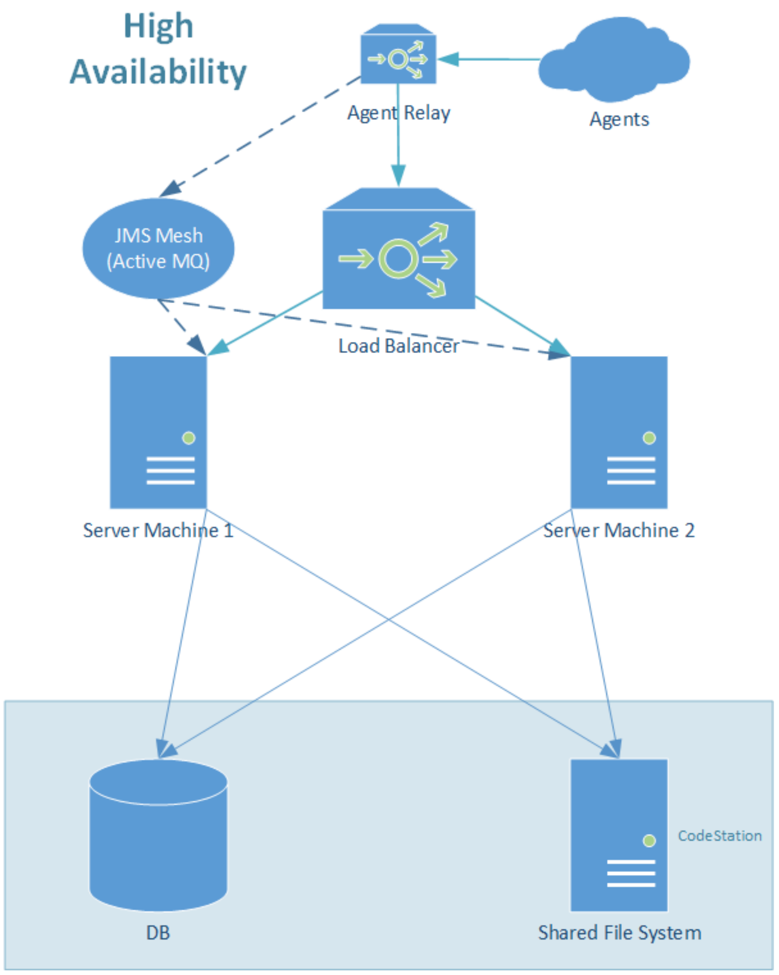


To convert the cold standby configuration to clustered:.

1. Upgrade all IBM uDeploy servers to the same version, if necessary.
2. Start the standby server and ensure all servers can communicate with one another.
3. Add a load balancer and point it to both servers.
4. Modify the DNS to point to the load balancer. At this point, traffic will start to be routed to the previously cold server.

The converted configuration is shown in the following illustration.

Figure 11. HA Topology



Agents

JMS Communications

JMS communications is subscription based. All agents subscribe to all topics and filter the list using their agent ID (which is one reason agents should have unique IDs).

For JMS-based communications, agents can be configured in several ways:

- **Approach 1** Agents configured to connect to a series of endpoints which will be tried in some order if failures occur (not optimal)
- **Approach 2** Agents connect to a single endpoint that uses round-robin DNS or an agent relay (recommended)

With Approach 2, workloads are distributed randomly and transparently. If a server cannot keep up with the workload, work is queued—no additional configuration is required.

HTTP/HTTPS Communications

HTTP/HTTPS server-agent communications are handled by a standard load balancer. Authentication is performed on every request; any server can perform validation for any request.

In case of server failure, user-server authentication must be redone unless ‘remember me’ cookie used.

Servers

All servers use the same database and a common file share. The file share is used for logs and several other directories—specifically: var/config, var/email, var/plugins, and var/repository. Each server also independently maintains some configuration information, such as ports and hosts. The database is used for configuration information, run-time data, etc.

Because the servers share the database, all servers run on the same interval.

Some configuration properties remain on the server, such as database and JMS connection information. Database configuration is handled during product installation; no additional configuration is required post-installation.

Importing Files (CodeStation)

All servers poll for component version changes. Polling intervals are specified by a user-configured parameter (15 minutes by default). The database handles server synchronization: before writing to the repository, a server acquires a lock in the database. Polling times are reset after a job finishes.

Events

Events are handled by the server firing the event.

Workflow Engine

Workflows consist of activities. Activities can be performed sequentially, run in parallel with one another, or some combination of the two. A typical workflow might consist of several sequential activities, such as:

For JMS-based communications, agents can be configured in several ways:

- Activity A: run process A on Agent 1
- Activity B: run process B on Agent 2
- Activity C: run process C on Agent 3

All servers constantly poll for pending workflows, so any server might initiate the workflow. The server that acquires this workflow will:

1. Create a run-time instance of Activity A and acquire a database lock.
2. Record the command it intends to send in the database.
3. Send the command to the agent over JMS.
4. Release the database lock.

After performing the work, the agent will send a response message over JMS. The message will be written to the database (by one of the servers) and the next activity started (by one of the servers). The server starting Activity B will perform the same steps as described above.

In the simple workflow sketched here, the activities might all be handled by the same server or different servers (or some combination). Of course, the same would be true if this workflow consisted of three parallel activities.

An application workflow is maintained by a single record in the database (only one thread handles a workflow at the same time).

Failure Handling

During application processing, command failures are marked in the workflow. Error handling is the responsibility of the application author. Component rollback can be handled with rollback command/steps. Rolling-back, as used here, means reinstalling an earlier component version.

If the server crashes while an agent is performing a command, the JMS mesh will assign the workflow to another server.

If an agent crashes or otherwise disappears while running a command (remembering that failed steps do not cause agents themselves to fail), the server assumes the command is still running—there is no ‘automatic time out. Normally, it is neither feasible nor practicable to assign a time-out interval.

Load Balancing

Standard load balancing is assumed. Each server has the same configuration and points to the same database. Load balancing is done around user requests and associated workflows.

In addition, load-balancing should improve the performance of the IBM uDeploy user interface.

Upgrading HA

All servers configured for the HA feature must be on the same version, which means all servers must be upgraded at the same time.

Table 2. Server Upgrade Task Flow

Step	Task
Step 1	Download and uncompress the installation package. Upgrades are done with the same package used to perform new installations.
Step 2	Shutdown the servers. As usual, make sure the servers have actually stopped before performing the upgrade.
Step 3	Run the server installation script.
Step 4	When prompted for the location of the installation directory, enter the path to one of the servers. By specifying an existing installation, you are asked if you want to upgrade the current installation. If you answer yes, the script will lead you through the required steps.

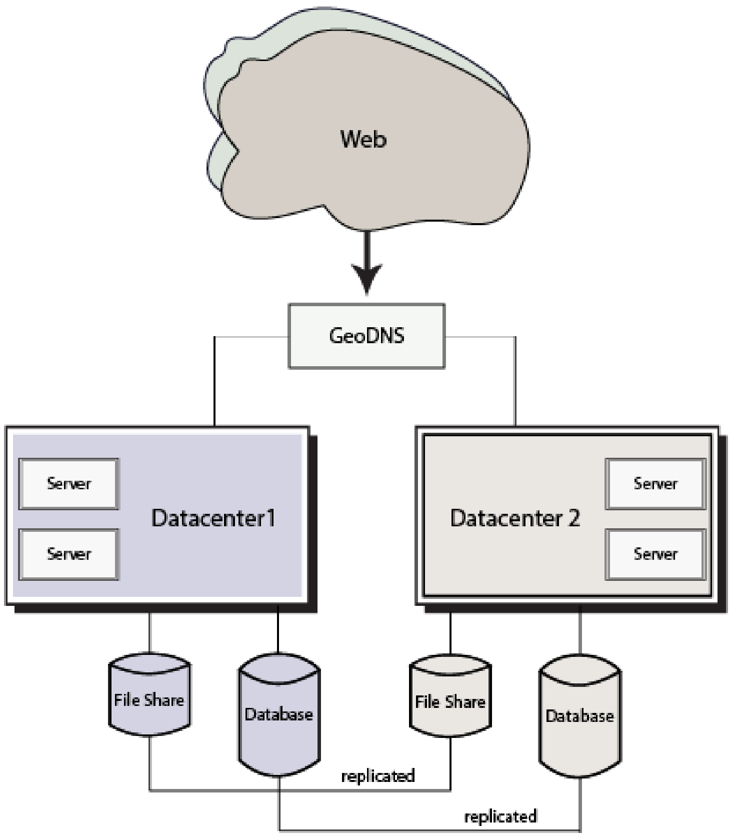
Step	**Task**
Step 5	When the script prompts you to upgrade the database, answer *yes* for the first server (only). When performing a database-only upgrade, the script will ensure that the update is performed only once. Note: upgrading the first server is sufficient even when using multiple data centers.
Step 6	Restart the upgraded server. Always use service scripts to start/stop the server.
Step 7	Once the first server is back on line, upgrade the other servers, taking care to skip the database upgrade step when prompted.

Multi-Datacenter Clustering

MDHA Topology

In a multi-datacenter high availability topology, each datacenter runs the same application and has access two copies of the database (which maintains application state). Users can communicate with any datacenter. There is no single point of failure, and any element can be removed—or fail—transparently.

Figure 12. General Topology

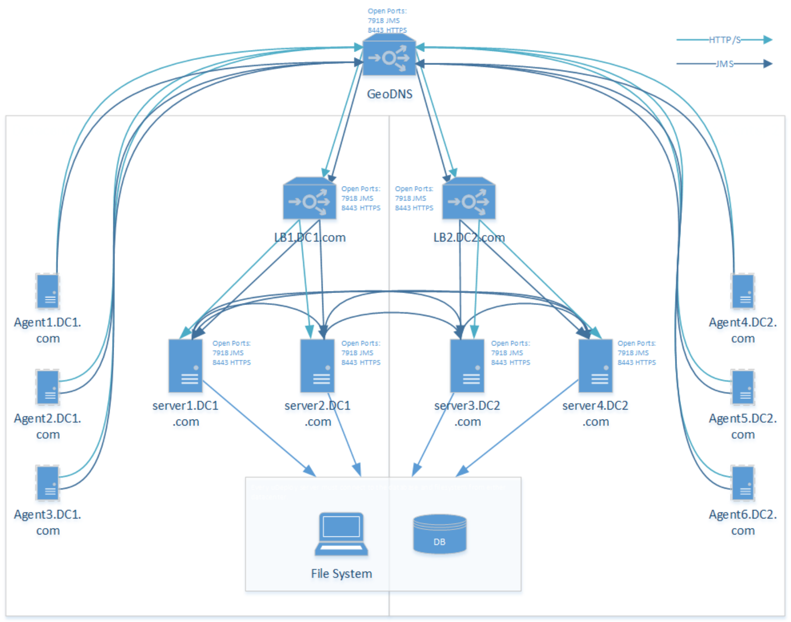


Changes to one database are replicated in the others. There are many techniques for replicating data, such as asynchronous, synchronous replication, or transactional.

Master-master or multi-homing datacenters as described here can work well as long as there is a low latency connection between data centers.

The following illustrates one viable approach.

Figure 13. Sample Topology



MDHA Setup

This section discusses MDHA-specific setup. For general information about installing and configuring IBM uDeploy, see ???.

MDHA Configuration

The following activities should be performed in the order presented.

1. **Install database** The shared database should be installed on a separate machine and before the servers are installed. See ???. Complete database installation by configuring the appropriate JDBC driver (typically supplied by the database vendor).

2. **Create an empty database for IBM uDeploy** The database should have at least one dedicated user account.

3. **Install the first IBM uDeploy server** See see ??? for server installation information.

   For HDMA, link the following server directories to a shared network drive:

   ```
   server_install_directory/logs
                            /var/config
                            /var/email
                            /var/plugins
                            /var/repository
   ```

   In the uDeploy web-based user interface, configure the server as a network relay by defining:

   ```
   Name
   ```

```
Host
Port
```

Make sure you check the Active check box.

4. **Additional servers** For each additional server participating in the HDMA topology:

   - Link it to the listed directories on the shared drive.
   - Configure each one to use the same database configuration as the first server.
   - Configure each one as a network relay, as described above.

5. **Install load balancers** The load balancers should be in front of the IBM uDeploy web-based user interface.

Appendix A. High-availability Use Cases

These cases cover situations that could negatively impact communication or activity on servers and agents.

Case 1: Server crashes during Agent HTTP/S request

Description

Agent is retrieving information from the server over HTTP/S when the server stops responding.

Scenario

Log requests on the agents will retry after a failure and that new request will be routed to an available server that will fulfill the request.

Plug-ins control their own communication with the server, and therefore may not have a retry on failure mechanism. For these, the step will fail.

Case 2: Server crashes during UI session

Description

Server crashes during a user session in the UI.

Scenario

The next request will be fulfilled by a different server. If the “remember me” option was checked, the user does not need to log back in. Otherwise, the user is prompted to login again.

Case 3: Server crashes during importing of data from an external source

Description

Two or more servers poll for a component in the artifact repository at the same time.

Scenario

Only one server acquires the lock for a component and polls for that artifact while the other servers wait for the lock. Once the lock is released, another server will receive the lock and poll and not get the same work as the last one.

Case 4: Server crashes after sending command to agent

Description

Communication between the agent and server is severed after a command begins executing on the agent.

Scenario

The reply from command execution will be queued and another server will process that reply.

Case 5: Server issues command to agent and crashes before writing the status to the database

Description

The server crashes after it has sent a command to an agent and before it writes the status to the database.

Scenario

The next server will restore the workflow and send the command again since the server does not know it was sent the first time. So the agent will get the command twice.

Case 6: Server crashes when about to issue a command

Description

When running a workflow the server crashes as it's about to issue a command to an agent.

Scenario

The next server will load the workflow instance, issue the command to the agent, and continue on with the execution of the workflow.

Case 7: Server crashes before a workflow runs any steps on the agent

Description

When running a workflow the server crashes before any plug-in steps are run.

Scenario

The next server will load up the workflow instance. The workflow will execute on the next server.