

# **uDeploy<sup>™</sup> User Guide**

—

**uDeploy 4.5.0**

—

**Document Version 4.5.0.2**



# Contents

<b>List of Figures.....</b>	<b>7</b>
<b>List of Tables.....</b>	<b>11</b>
<b>Part I: Introduction.....</b>	<b>13</b>
Overview.....	14
Components.....	15
Applications.....	18
Agents.....	19
Resources.....	19
<b>Part II: Hands-On.....</b>	<b>21</b>
Getting Started.....	22
Creating Components.....	22
hello_world Component Version.....	23
Hello World Component Process.....	26
hello_world Component Process Design.....	27
Hello World Application.....	32
<b>Part III: Using uDeploy.....</b>	<b>43</b>
Components.....	44
Creating Components.....	44
Component Processes.....	46
Process Editor.....	48
Component Templates.....	55
Resources.....	57
Resource Groups.....	59
Applications.....	61
Creating Applications.....	62
Deployments.....	64
Reports.....	67
Deployment Reports.....	68
Security Reports.....	77
Saving and Printing Reports.....	79
Schedule Deployments.....	80
<b>Part IV: Administration.....</b>	<b>81</b>
Installation.....	82
System Requirements.....	82
Download UrbanDeploy.....	84
Database Installation.....	84
Server Installation.....	86
Agent Installation.....	88
Running uDeploy.....	89
Security.....	90
Authentication.....	91

Authorization.....	94
<b>Part V: Reference.....</b>	<b>97</b>
Plug-in Integration.....	98
Ant Plug-in.....	99
Groovy Plug-in.....	99
IIS_AppCmd Plug-in.....	99
JBoss Plug-in.....	100
SQL/JDBC Plug-in.....	100
SQLPLUS Plug-in.....	100
Tomcat Plug-in.....	101
WebSphere Plug-in.....	101
WLDeploy Plug-in.....	102
Standard Component Process Steps.....	103
Notifications.....	103
Configuration.....	105
Application Configuration.....	106
Component Configuration.....	108
Environment Configuration.....	109
Inventory.....	110
CLI Reference.....	113
addActionToRoleForApplications.....	113
addActionToRoleForComponents.....	113
addActionToRoleForEnvironments.....	113
addActionToRoleForResources.....	114
addActionToRoleForUI.....	114
addComponentToApplication.....	114
addGroupToRoleForApplication.....	115
addGroupToRoleForComponent.....	115
addGroupToRoleForEnvironment.....	116
addGroupToRoleForResource.....	116
addGroupToRoleForUI.....	117
addLicense.....	117
addNameConditionToGroup.....	117
addPropertyConditionToGroup.....	118
addResourceToGroup.....	118
addRoleToResource.....	119
addRoleToResourceWithProperties.....	119
addUserToGroup.....	119
addUserToRoleForApplication.....	120
addUserToRoleForComponent.....	120
addUserToRoleForEnvironment.....	121
addUserToRoleForResource.....	121
addUserToRoleForUI.....	121
addVersionFiles.....	122
addVersionStatus.....	122
createApplication.....	123
createApplicationProcess.....	123
createComponent.....	124
createComponentProcess.....	124
createDynamicResourceGroup.....	125
createEnvironment.....	125
createGroup.....	126
createMapping.....	126
createResourceGroup.....	126

createRoleForApplications.....	127
createRoleForComponents.....	127
createRoleForEnvironments.....	127
createRoleForResources.....	128
createRoleForUI.....	128
createSubresource.....	128
createUser.....	129
createVersion.....	129
deleteGroup.....	130
deleteResourceGroup.....	130
deleteResourceProperty.....	130
deleteUser.....	131
exportGroup.....	131
getApplication.....	131
getApplicationProcess.....	132
getApplicationProcessRequestStatus.....	132
getApplications.....	132
getComponent.....	133
getComponentProcess.....	133
getComponents.....	133
getComponentsInApplication.....	134
getEnvironment.....	134
getEnvironmentsInApplication.....	134
getMapping.....	135
getResource.....	135
getResourceGroup.....	135
getResourceGroups.....	135
getResourceProperty.....	136
getResources.....	136
getResourcesInGroup.....	136
getRoleForApplications.....	137
getRoleForComponents.....	137
getRoleForEnvironments.....	137
getRoleForResources.....	138
getRoleForUI.....	138
getUser.....	138
importGroup.....	139
importVersions.....	139
login.....	140
logout.....	140
removeActionFromRoleForApplications.....	140
removeActionFromRoleForComponents.....	141
removeActionFromRoleForEnvironments.....	141
removeActionFromRoleForResources.....	141
removeActionFromRoleForUI.....	142
removeGroupFromRoleForApplication.....	142
removeGroupFromRoleForComponent.....	143
removeGroupFromRoleForEnvironment.....	143
removeGroupFromRoleForResource.....	143
removeGroupFromRoleForUI.....	144
removeResourceFromGroup.....	144
removeRoleForApplications.....	145
removeRoleForComponents.....	145
removeRoleForEnvironments.....	145
removeRoleForResources.....	146
removeRoleForUI.....	146

removeRoleFromResource.....	146
removeUserFromGroup.....	147
removeUserFromRoleForApplication.....	147
removeUserFromRoleForComponent.....	147
removeUserFromRoleForEnvironment.....	148
removeUserFromRoleForResource.....	148
removeUserFromRoleForUI.....	149
repeatApplicationProcessRequest.....	149
requestApplicationProcess.....	150
setComponentEnvironmentProperty.....	150
setComponentProperty.....	151
setResourceProperty.....	151
updateUser.....	151

# List of Figures

Figure 1: uDeploy Deployment Process.....	14
Figure 2: Component Process with Switch Step.....	16
Figure 3: Version Pane.....	25
Figure 4: Component Artifacts.....	26
Figure 5: Process Design Pane.....	29
Figure 6: Adding a Step to an Anchor Point.....	30
Figure 7: Edit tool.....	32
Figure 8: Adding a component to an application.....	34
Figure 9: Adding a resource to an environment.....	35
Figure 10: Environments Tab.....	36
Figure 11: Create an Application Process dialog.....	36
Figure 12: Process Design Pane.....	37
Figure 13: Edit Properties Dialog.....	38
Figure 14: Run for Versions Without Inventory Status field.....	38
Figure 15: Nested parameters.....	39
Figure 16: Process Design Pane.....	49
Figure 17: Typical Process Step.....	50
Figure 18: Adding a Step.....	51
Figure 19: Typical Edit Properties Pop-up.....	51
Figure 20: Connection Tool.....	52
Figure 21: Dragging the Connection Over a Target Step.....	52
Figure 22: Completed Connection.....	53
Figure 23: Edit Properties Dialog.....	53
Figure 24: Process with Switch Step.....	54
Figure 25: Component Template template nameView.....	57

Figure 26: Resources Pane.....	58
Figure 27: Resource Groups Pane.....	58
Figure 28: Sub-Groups.....	59
Figure 29: Action Tool.....	60
Figure 30: Create a Resource Group Dialog.....	60
Figure 31: Add a Resource Dialog.....	60
Figure 32: Sub-resources.....	61
Figure 33: Deploy Application.....	65
Figure 34: Deployment Count Graph.....	72
Figure 35: Security Pane.....	90
Figure 36: Security Pane.....	91
Figure 37: Create Authorization Realm.....	91
Figure 38: Authorization Realm Dialog.....	92
Figure 39: Edit Users.....	93
Figure 40: Group Dialog.....	94
Figure 41: Role Pane.....	95
Figure 42: Notification Schemes.....	103
Figure 43: Notification Type.....	104
Figure 44: Notification Target.....	104
Figure 45: Notification Role.....	104
Figure 46: Template.....	105
Figure 47: Configuration Tab.....	106
Figure 48: Application Properties panel.....	107
Figure 49: Edit Property pop-up.....	108
Figure 50: Configuration Tab.....	109
Figure 51: Environment Configuration Tab.....	110
Figure 52: Resource inventory.....	111
Figure 53: Component inventory.....	112



Figure 54: Environment Inventory..... 112



# List of Tables

Table 1: Deployment Reports.....	67
Table 2: Security Reports.....	68



---

# Part

# I

---

## Introduction

---

Topics:

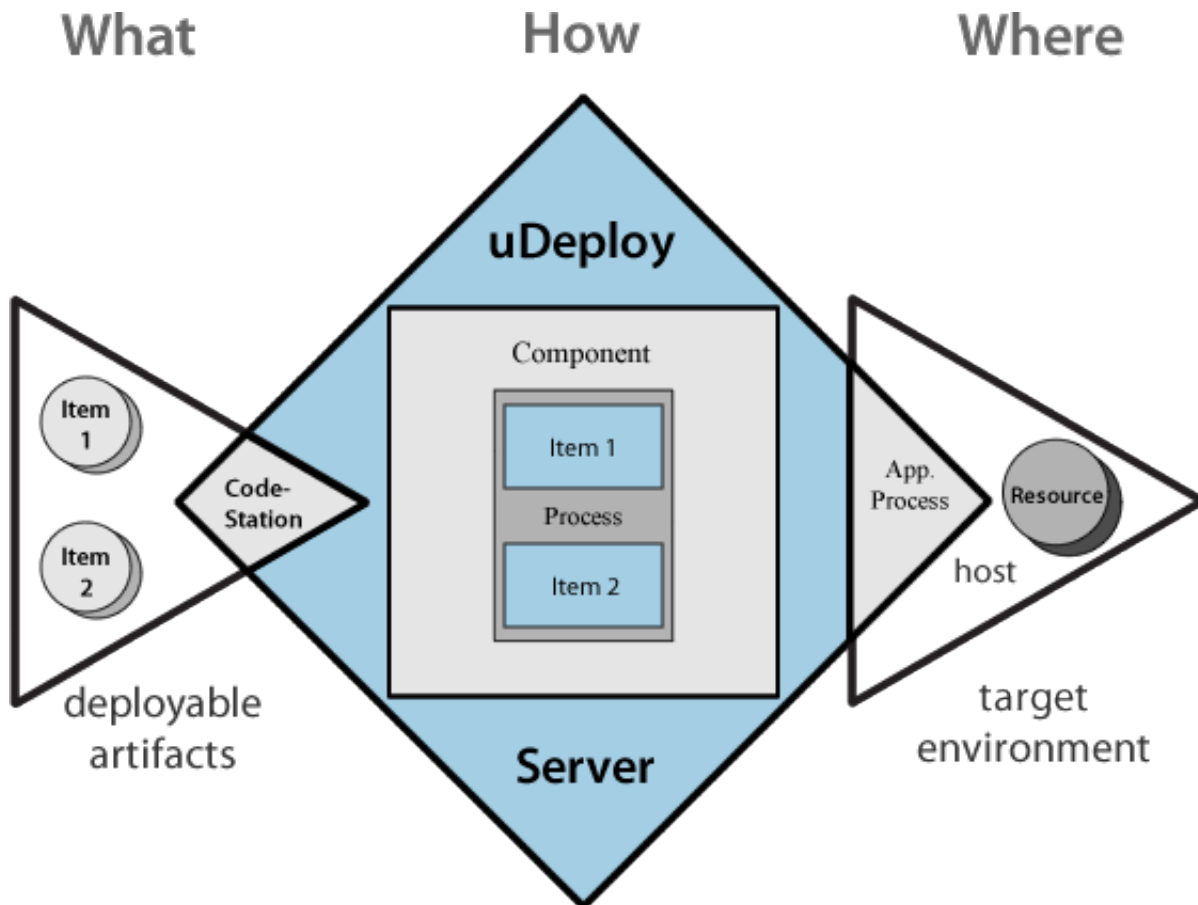
- [Overview](#)
-

## Overview

At its base, software deployment is a simple concept that sometimes gets obscured by jargon. Deployment is the process of moving software (broadly defined) through various preproduction stages to final production. Typically, each stage represents a step of higher criticality, such as quality assurance to production. Complexity arises from the sheer volume of things deployed, the number and variety of deployment targets, constantly-decreasing deployment cycles, and the ever-increasing rate of technological change. Virtualization, while providing some relief, also increases the challenge by its exponential growth of deployment targets.

uDeploy helps you meet the challenge by providing resources that improve deployment speeds while simultaneously improving their reliability. uDeploy's release automation tools provide complete visibility into  $n$ -tiered deployments, enabling you to model processes that orchestrate complex deployments across every environment and approval gate. uDeploy's drag-and-drop design tools decrease design-time by making it easy to visualize the end-to-end deployment process and develop the big picture--the *What*, *How*, and *Where* of the deployment workflow:

- **What:** the deployable items--binaries, static content, middleware updates, database changes and configurations, and anything else associated with the software--that uDeploy delivers to target destinations.
- **How:** refers to combining deployable items with processes to create components, and designing applications that coordinate and orchestrate multi-component deployments.
- **Where:** the target destination's hosts and environments--uDeploy can scale to any environment.



**Figure 1: uDeploy Deployment Process**

In uDeploy, deployable items are combined into logical groupings called *components*. Components are deployed by *component processes* which consist of user-configured steps, many taken from integrations with third-party tools called *plug-ins*. Multi-component deployments are handled by user-assembled *applications*.

uDeploy represents deployment targets by what it calls *resources*. Resources--databases, servers, and so on--reside on hosts. Complex deployments can contain numerous components. Components can also remain independent of one another, which enables incremental or targeted deployments. Of course, you can model your components as you see fit--uDeploy is flexible and works the way you work.

uDeploy helps you rapidly adapt to ever-changing market conditions by providing:

- continuous deployment using automated triggers
- scheduled deployments
- self-service deployments with per-environment access control
- automatic deployment roll-back
- integration with authentication systems such as LDAP
- artifact repository
- tight SCM integration
- build tool integration

### uDeploy Server

The uDeploy server is a standalone server that provides uDeploy's core services such as the user interface, component and application configuration tools, workflow engine, and security services, among others.

uDeploy supports cross-network deployments with *relay servers*. Relay servers enable network-to-network communications.

### uDeploy Agents

An agent is a lightweight process that runs on a host and communicates with the uDeploy server. Agents manage the resources that are the actual deployment targets. Each machine participating in a deployment usually has an agent installed on it. When not performing deployments, agents run in the background with minimal overhead. See [Resources](#) on page 19.

### Repository

Deployable items are stored in a repository, such as CodeStation or Maven. The uDeploy artifact repository, CodeStation, provides secure and tamper-proof storage. It tracks artifact versions as they change and maintains an archive for each artifact. Associations between repository files and components are built-in and automatic.

See [Architecture and Technology](#).

### Security

In uDeploy's role-based security, users are assigned roles, and role-permissions are assigned to things such as projects, build configurations, and other resources. For example, a developer may be permitted to build a project, but only view non-project related material. See [Security](#) on page 90.

## Components

Understanding how uDeploy uses the term *component* is critical to understanding uDeploy. Components represent deployable items along with user-defined processes that operate on them, usually by deploying them. Deployable items--also called *artifacts*--can be files, images, databases, configuration materials, or anything else associated with a software project. Components have *versions* which are used to ensure that proper component instances get deployed.

Artifacts can come from a number of sources: file systems, build servers such as AnthillPro, source version control systems, Maven repositories, as well as many others. When you create a component, you identify the source and define how the artifacts will be brought into uDeploy. If the source is Subversion, for example, you specify the Subversion repository containing the artifacts. Each component represents artifacts from a single source.

## Component Processes

A component process is a series of user-defined steps that operate on a component's artifacts. Each component has at least one process defined for it and can have several. A component process can be as simple as a single step or contain numerous steps and relationships. The *switch* step, for instance, enables you to create conditional processes. You might, say, take artifacts from a source like an AntHillPro project and map the ones that get deployed to an HTTP server into one component; those that get deployed to a J2EE container to another; and those that get deployed to a database to yet another. Or, to take another example, a single-component deployment might consist of two processes: the first moves component files to a server on Friday night (a lengthy operation), while the second deploys the files Saturday morning.

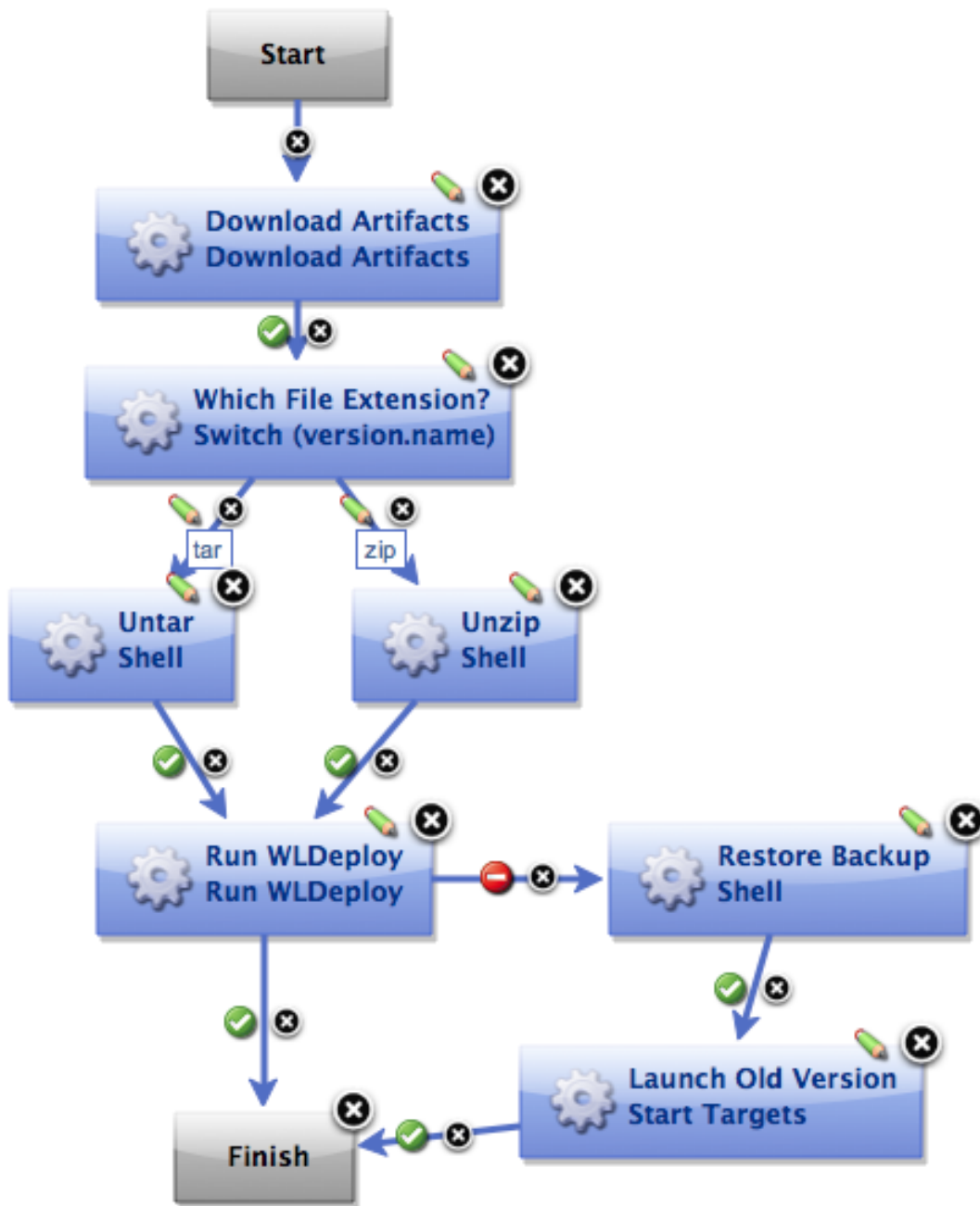


Figure 2: Component Process with Switch Step



Component processes are created with uDeploy's *process editor*. The process editor is visual drag-and-drop editor that enables you to drag process steps onto the workspace and configure them as you go. As additional steps are placed, you visually define their relationships with one another. Process steps are selected from a menu of standardized steps that replace typical deployment scripts and manual processes. uDeploy provides steps for several utility processes, such as inventory management, and workflow control. Additional process steps are provided by plug-ins. Plug-ins provide integration with common deployment tools and application servers, such as WebSphere, Microsoft IIS, and many others. Out-of-the-box, uDeploy provides plug-ins for many common processes, such as downloading and uploading artifacts, and retrieving environment information. A component process can have steps from more than one plug-in.

A component process is defined for a specific component. A component can have more than one process defined for it, but each component requires at least one process.

For example, deploying a J2EE EAR file to WebSphere server typically consists of the following operations:

1. transfer the EAR file to the target machine
2. stop the WebSphere server instance
3. invoke wsAdmin with deployment properties
4. start the WebSphere instance
5. verify that the deployment succeeded by accessing a specified URL

The WebSphere plug-in provides a configurable process step for each operation.

A frequently used component process can be saved as a template and applied later to new components.

Component processes are executed by uDeploy agents running on hosts. One instance of a component process is invoked for each resource mapped to a component in the target environment, see [Resources](#) on page 19.

## Plug-ins

Plug-ins provide integration with third-party tools. uDeploy ships with plug-ins for several common deployment processes, and others are readily available for a wide variety of tools, such as middleware tools, databases, servers, and other deployment targets.

Third-party tools exhibit wide and varied functions, of course. Plug-in integration is achieved by breaking down a tool's functions into simple, discrete steps that invoke a specific behavior. A plug-in step might invoke a tool, or invoke different functions in a tool, such as extracting or inserting some type of data.

When you use plug-ins to create a component process, you can use steps from several plug-ins and configure the steps as you go. For example, you might create a process using a plug-in for a source control tool that deploys a component to a middleware server, and another plug-in to configure a step that removes the component from the server.

A component process that contains a plug-in step requires an agent. Unless the agent needs to interact with the host's file system or system processes, the agent does not have to be on the same host as the target resource.

uDeploy enables you to download and install numerous component plug-ins. UrbanCode does not charge any additional fees for plug-ins. The plug-in system is open and extensible--plug-ins can be written in any language.

## Component Versions and the CodeStation Repository

After defining a component's source and processes, you import its artifacts into uDeploy's artifact repository CodeStation. Artifacts can be imported automatically or manually. By default, a complete copy of an artifact's content is imported into CodeStation (the original artifacts are untouched). This provides several benefits, such as tamper-proof storage, and the ability to review and validate artifacts with uDeploy's user interface. But if you have storage concerns or use a tool like Maven, you can limit CodeStation to using references to the artifacts instead of actually copying them.

Each time a component is imported, including the first time, it is versioned. Versions can be assigned automatically by uDeploy, applied manually, or come from a build server. Every time a component's artifacts are modified and reimported, a new version of the component is created. So a component might have several versions in CodeStation and each version will be unique.

A version can be *full* or *incremental*. A full version contains all component artifacts; an incremental version only contains artifacts modified since the previous version was created.

## Applications

Applications are the mechanism that initiate component deployments; they bring together components with their deployment targets, and orchestrate multi-component deployments.

### Application Process

When you create an application, you identify the included components and define an *application process*. Application processes, like component processes, are created with the process editor. uDeploy provides several common process steps, otherwise application processes are assembled from processes defined for their associated components.

Application processes can run manually, automatically on some trigger condition, or on a user-defined schedule. When a component has several processes defined for it, the application determines which ones are executed and in which order. For instance, an n-tiered application might have a web tier, a middleware tier, and a database tier. And, continuing the example, the database tier must be updated before the other two, which are then deployed concurrently. An application can orchestrate the entire process, including putting servers on- and off-line for load-balancing as required.

An application process is always associated with a target environment. When an application process executes, it interacts with a specific environment. An environment is a collection of one or more resources. At least one environment must be associated with the application before the process can be executed. Application processes are environment agnostic; processes can be designed independently of any particular environment. This enables a single application to interact with separate environments, such as QA, or production. To use the same application process with multiple environments (a typical scenario), you associate each environment with the application and execute the process separately for each one.

In addition to deployments, several other common processes are available, such as rolling-back deployments. uDeploy tracks the history of each component version, which enables application processes to restore environments to any desired point.

### Environments

An *environment* is a user-defined collection of resources that host an application. Environments are typically modeled on some stage of the software project life cycle, such as development, QA, or production. A resource is a deployment target, such as a database or J2EE container. Resources are usually found on the same host where the agent that manages them is located. A host can be a physical machine, virtual machine, or be cloud-based. See [Architecture and Technology](#).

Environments can have different topologies--for example: an environment can consist of on a single machine; be spread over several machines; or spread over clusters of machines. Environments are application scoped. Although multi-tenant machines can be the target of multiple applications, experience has shown that most IT organizations use application-specific environments. Additionally, approvals are generally scoped to environments.

uDeploy maintains an inventory of every artifact deployed to each environment and tracks the differences between them.

### Snapshots

You can also create application *snapshots*. A snapshot is a collection of specific component versions, usually versions that are known to work together. Typically, a snapshot is generated in an uncontrolled environment--meaning one without approvals. When a snapshot is created, a picture of the application's current state is captured. As an application moves through different environments, snapshots can validate that proper component versions are used.

Snapshots help manage complex deployments--deployments with multiple tiers and development teams. For example, after testing and confirming that team A's component works with teams B's, a snapshot could be taken. Then, as development progressed, additional snapshots could be taken and used to model the effort and drive the entire deployment, coordinating versions, configurations, and processes.

## Agents

Agents are integral to uDeploy's client/server architecture. An agent is a process that runs on target host and communicates with the uDeploy server. Agents perform the actual work of deploying components and so relieves the server from the task, making large deployments involving thousands of targets possible.

Typically, an agent run on the same host where the resources it handles are located. A single agent can handle all resources on its host. If a host has several resources, an agent process is invoked separately for each resource. For example, a test environment might contain a single web server, a single middleware server, and a single database server all running on the same host (machine). A deployment to this environment might have one agent and three separate resources.

Depending on the number of hosts in an environment, a deployment might require a large number of agents. Agents are unobtrusive and secure. Agent communications use SSL encryption and mutual key-based authentication. For added security, agents do not listen to ports, but open direct connections to the server instead.

## Resources

Resources are logical constructs based on uDeploy's architectural model. Resources aid bookkeeping; inventory is tracked for resources. Resources are created and managed through the user interface.

A resource represents a deployment target--a physical machine, virtual machine, database, J2EE container, and so on. Components are deployed to resources by agents (which are physical processes). Resources generally reside on the same host where its managing agent runs. A host can have more than one resource. If an agent is configured to handle multiple resources, a separate agent process is invoked for each one.

A resource can represent a physical machine, which is the simplest configuration, or a specific target on a machine, such as a database or server. So a host (machine) can have several resources represented on it. In addition, a resource can represent a process distributed over several physical or virtual machines. Environments consist of resources.

To perform a deployment, at least one resource must be defined and (usually) at least one agent. ("Usually" because trivial deployments can be done without an agent.) Typically, each host in a participating environment has an agent running on it to handle the resources located there.

A *proxy resource* is a resource effected by an agent on a host other than the one where the resource is located. If an agent does not require direct interaction with the file system or with process management on the host, a proxy resource can be used. When a deployment needs to interact with a service exposed on the network (a database or J2EE server, for instance), the interaction can happen from any machine that has access to the networked service.

## Resource Groups

A *resource group* is a logical collection of resources. Resource groups enable collections of resources to be easily reused. Resource groups can manage multi-tenant scenarios, for example, in which several machines share the same resources.



---

# Part II

---

## Hands-On

---

Topics:

- [Getting Started](#)
-

## Getting Started

---

Welcome to uDeploy! This section gets you started by providing immediate hands-on experience using uDeploy. The `hello_world` walk-through shows you how to create a simple deployment using out-of-the-box features. The second walk-through, `hello_worldWS`, shows you how to install a freely-available plug-in (for the WebSphere server in this instance) and create a basic deployment using it.



**Note:** This section assumes you have installed the uDeploy server and at least one agent. For the walk-through, the agent can be installed on the same machine where the server is installed. If the agent or server have not been installed, see [Installation](#) on page 82 for information about installation.

### Quick Overview

Generally, the following steps are performed when creating a deployment:

#### 1. Configure Resources

Resources are agents and agent groupings. Typically, at least one agent is installed in every environment used by the deployment. As mentioned, *Quick Start* assumes that at least one agent has already been installed and so we will not cover agent creation here. See [Resources](#) on page 57 for more information about agents.

#### 2. Define Components

Components represent the source items that will be installed and managed by the deployment. When you create a component, you tell uDeploy where the items are found--a file system or source code repository, for example--and what processes should be performed on them. The source items and processes together define the component. See [Components](#) on page 44 for more information about creating components.

#### 3. Define Application

An application brings together all the components used by the deployment. When you create an application, you identify the components and define the processes required to move the components through all required environments. See [Applications](#) on page 61 for more information about creating applications.

## Creating Components

Components are the artifacts--files, images, databases, etc.--that UrbanDeploy manages and deploys. When creating a component, one good approach is:

#### 1. Create a version.

After you identify where the component's artifacts are stored on your system, you assign a version identifier to it. UrbanDeploy can use existing version schemes, such as the numbers assigned by your build server or artifact management server.

#### 2. Define processes.

The process is where you tell UrbanDeploy what to do with the component. A process is designed by assembling basic units of automation, called steps. Steps replace most deployment scripts and/or manual processes. Processes are designed using a drag-and-drop tool.

### `hello_world` Deployment

The `hello_world` deployment moves some files on the local file system to another location on the file system, presumably a location used by an application server. `hello_world` is a very simple deployment but it has several advantages: it uses many of UrbanDeploy's key features--features you will use every day, and it does not require the installation of additional plug-ins.

UrbanDeploy plug-ins provide integration with many common deployment tools and application servers. Each integration has at least one step, which can be thought of as a distinct piece of automation. By stringing individual steps together, you create a fully automated process that can replace many of your existing scripts and manual processes. Plug-ins are available for Subversion, Maven, Tomcat, WebSphere (which we demonstrate later), and many others.

## A Note Before You Begin

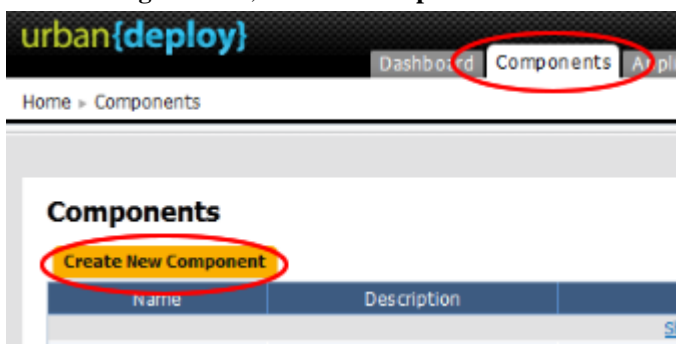
You can read the walk-through without actually performing the steps, or you can perform them as you read along. If you want to actually perform the steps as we go, do the following before starting:

1. Create a directory somewhere on your system named `helloWorld`.
2. Within `helloWorld` create a sub-directory named `1.0`.
3. Within `1.0` place several--say 5--files. For speed, text-type files should be used.
4. Create another directory somewhere on your file system.
5. Within the directory just created, create a sub-directory. This sub-directory will be the target for our deployment. I created `C:\UAT\appUAT` on my system.

## hello\_world Component Version

To configure the Hello World Component Version:

1. On the **Navigation** bar, click the **Components** tab.



2. On the **Components** pane, click **Create New Component**.

Components are defined with the **Create New Component** dialog box. The first four and last two fields displayed are the same for every source type; the remaining fields depend on the value selected in the **Source Config Type** field.

 A screenshot of the 'Create New Component' dialog box. The dialog has a title bar with 'Create New Component' and a close button. The fields are:
 

- Name \***: A text input field.
- Description**: A text input field.
- Template**: A dropdown menu with 'None' selected and a help icon.
- Status Plugin \***: A dropdown menu with 'Default' selected and a help icon.
- Inherit Cleanup Settings**: A checkbox with a help icon.
- Days to Keep Versions \***: A text input field with a help icon.
- Number of Versions to Keep \***: A text input field with a help icon.
- Source Config Type**: A dropdown menu with 'None' selected and a help icon.
- Import Versions Automatically**: A checkbox with a help icon.
- Copy to CodeStation**: A checked checkbox with a help icon.

 At the bottom are 'Save' and 'Cancel' buttons.

3. Enter `hello_world` in the **Name** field.

The name is used when assembling the application. If the component will be used by more than one application, the name should be generic rather than project-specific. For components that are project-specific, a name that conveys something meaningful about the project should be used.

4. Enter a description in the **Description** field.

The optional description can be used to convey additional information about the component. If the component is used by more than one application, for example, entering "Used in applications A and B" can help identify how the component is used. If you are unsure about what to enter, leave the field blank. You can always return to the component and edit the description at any time. In an attempt to appear hip, I entered `Euro store` for my component.

5. Accept the default value in the **Status Plug-in** field--Default.

Experienced users can use this field to customize plug-ins designed to monitor the component's status. See [TBD].

6. Select `File System` from the **Source Config Type** field.

Selecting a value changes several fields to those required by the selected value. The type-dependent fields are used to identify where the artifacts comprising the component are stored. See [TBD] for a description of the supported types.

`File System` is used when the artifacts are on a file share or the local file system. This is the simplest configuration option and can be used to quickly set up a component for evaluation purposes, as we do here.

7. Complete this option by entering the path to the artifacts.

In our example, the artifacts are stored in `C:\helloWorld`. Inside the base-directory, artifacts are stored in numbered directories; the numbers represent distinct versions. `C:\helloWorld` has only one version and so only one sub-directory--`1.0`. When automatically polling the base directory, or manually requesting a new version, `uDeploy` will compare the current version in the base-directory with the one stored in `CodeStation` (`uDeploy`'s artifact repository). If changes are found, a new version, using the name/number found, will be created.

8. Check the **Import Versions Automatically** check box.

`uDeploy` will automatically poll the source location for new versions when this option is selected. If new material is found, a new version will be created, based on the new version number. You can manually create versions by using the **Versions** tab. If this option is not selected, you will have to manually create a new version every time one becomes available.

9. Ensure the **Copy to CodeStation** check box is selected.

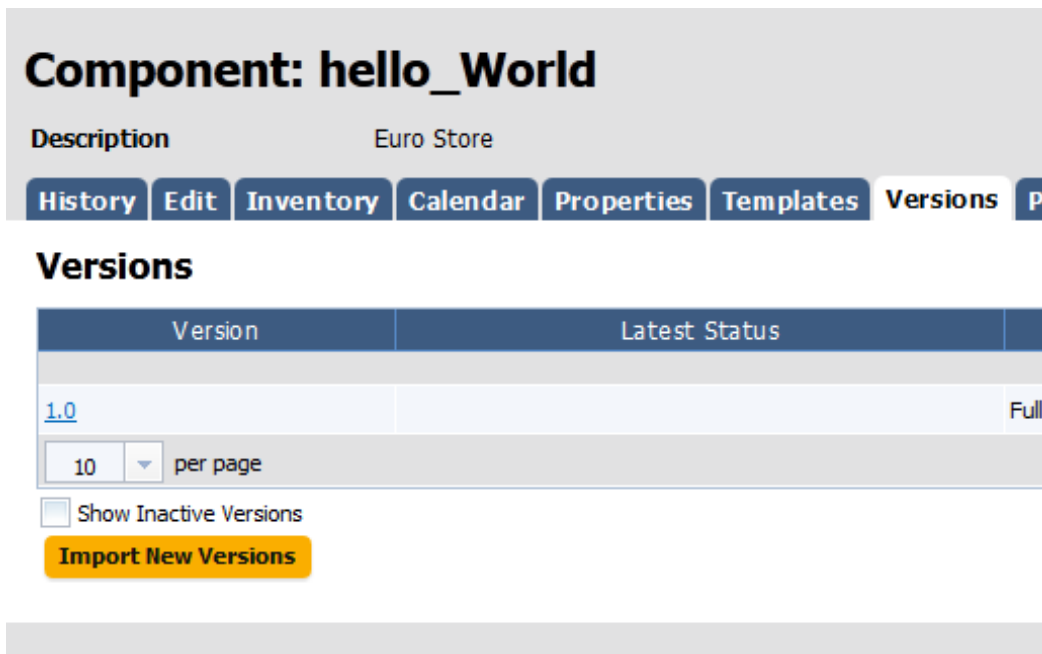
This option, which is recommended by `UrbanCode` and selected by default, creates a tamper-proof copy of the specified component and stores it in the embedded artifact management system--`CodeStation`. If this option is not selected, only meta data about the component version will be imported. The only advantage to bypassing `CodeStation` is the avoidance of storing the files in two places. In most situations this advantage is far outweighed by the reduced visibility into the artifacts.



10. Click the **Save** button to save the component.

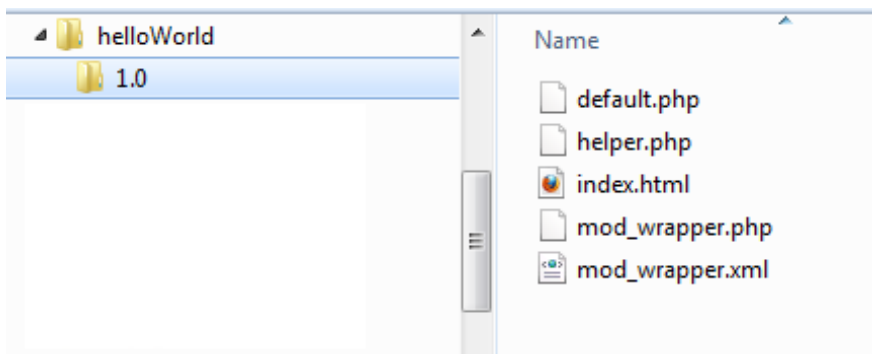
11. To verify that the correct files are imported into UrbanCode, click the **Versions** tab.

The Versions pane displays all versions for the selected component. If all went well, the material in the specified base-path was imported automatically.



**Figure 3: Version Pane**

The base-path, as you will recall, is C:\helloWorld. Within helloWorld is the single sub-directory, 1.0, as shown in the following illustration.



The 1.0 directory contains the artifacts that comprise the version. To see the artifacts, click on the version name in the **Version** pane.

**Statuses**

Status	Description	Created	By	Actions
No statuses have been assigned to this version. - <a href="#">Refresh</a>				

[Add a Status](#)

**Artifacts**

Name	Size	Last Modified	Version	Actions
default.php	0.8 KB	6/20/10 7:58 AM	1	<a href="#">Download</a>
helper.php	1.4 KB	6/20/10 7:58 AM	1	<a href="#">Download</a>
index.html	44 bytes	6/20/10 7:58 AM	1	<a href="#">Download</a>
mod_wrapper.php	1.0 KB	6/20/10 7:58 AM	1	<a href="#">Download</a>
mod_wrapper.xml	2.5 KB	6/20/10 7:58 AM	1	<a href="#">Download</a>

Figure 4: Component Artifacts

## Hello World Component Process

Once a component has been created and a version imported, a process to deploy the artifacts--called a component process--must be defined.

### To Configure the `hello_world` Component Process:

1. On the **Navigation** bar, click the **Components** tab.
2. On the **Components** pane, click the on the name of the component--`hello_world` on my machine.
3. On the **Component: *Name\_of\_selected\_component*** pane, click the **Processes** tab.
4. Click the **Create New Process** button.

**Component: hello\_world**

Description Euro Store

[History](#) [Edit](#) [Inventory](#) [Calendar](#) [Properties](#) [Templates](#) [Versions](#) [Processes](#)

**Processes**

Process	Description
No processes have been added to this component. - <a href="#">Refresh</a>	

Show Inactive Processes

[Create New Process](#)

5. In the **Create New Process** dialog, enter a name in the **Name** field.

The screenshot shows a 'Create New Process' dialog box with the following fields and values:

- Name \***: hello\_worldInstall
- Description**: hello\_world remote install
- Default Working Directory \***: \${p:resource/work.dir}/\${p:comp}
- Requires a Version**:
- Required Component Role**: None
- Inventory Action Type**: Add Inventory
- Inventory Status \***: Active

Buttons: Save, Cancel

The name and description typically reflect the component's content and process type.

6. Enter a meaningful description in the **Description** field.

If the process will be used by several applications, you can specify that here.

7. Accept the default value in the **Default Working Directory** field.

This is the location where the process steps will be executed. The default value enables the process to work in different environments, and for our exercise (and for most processes), the default value is fine. If you change the default value, the process might not work in every environment visited by the component.

8. Check the **Requires a Version** check box.

When checked, the version will be passed to the process at run-time.

9. Accept the default value (None) in the **Required Component Role** field.

This option enables you to restrict who can run this process. The available options are derived from the uDeploy Security System. For information about security roles, see [Security](#) on page 90.

10. Select Add Inventory in the **Inventory Action Type** field.

This field is displayed if the **Requires a Version** check box is selected. For information about inventory, see [Inventory](#) on page 110.

11. Accept the default value of Active in the **Inventory Status** field.

This field is displayed if the Add Inventory or Remove Inventory values are selected in the **Inventory Action Type** field. The Staged status is used when performing a rolling deployment.

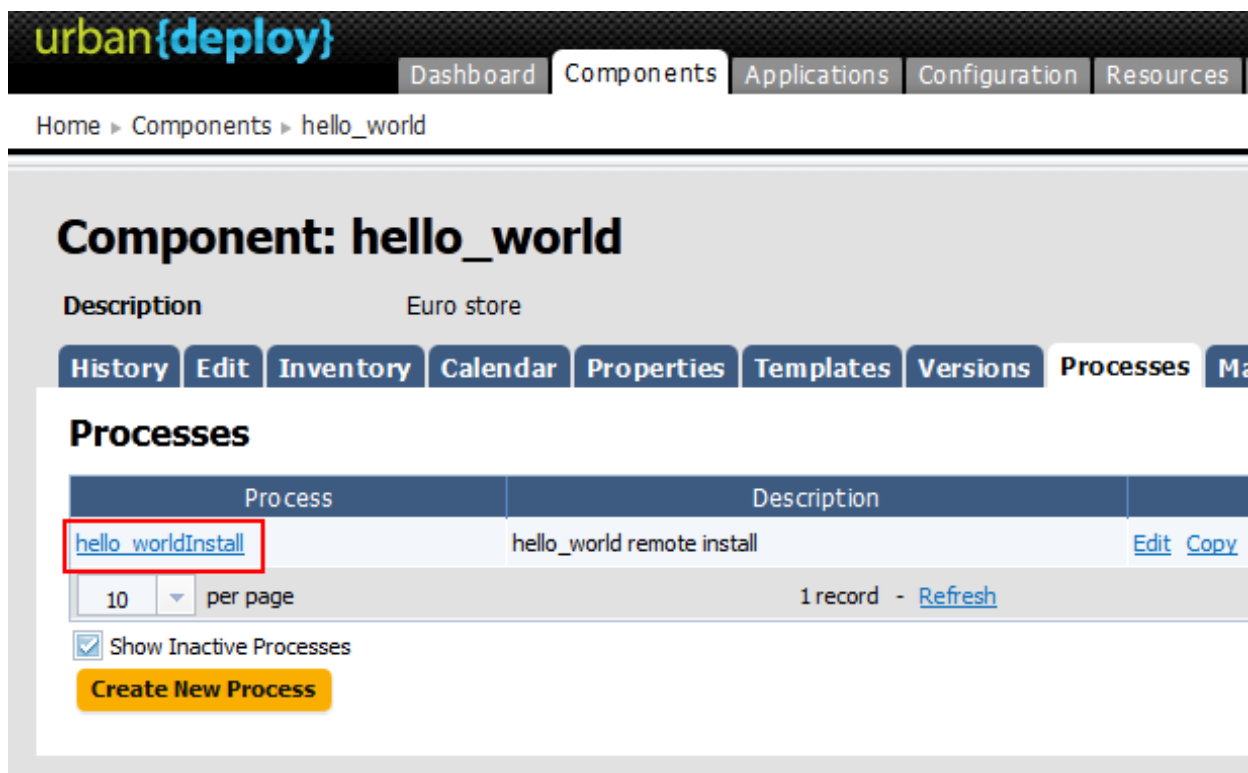
12. Use the **Save** button to save your work.

## hello\_world Component Process Design

To complete the process, you must define its individual steps. A component process must have at least one step. The steps are defined with the **Process Design** pane, see [Figure 5: Process Design Pane](#) on page 29. You define the steps by dragging-and-dropping them onto the design area and arranging them in the order they are to be executed.

### To Define the hello\_world Process Steps

1. On the **Component: hello\_world** pane, click the **Processes** tab.
2. Click the name of the process you created in the previous section--hello\_worldInstall in my case.



urban{deploy} Dashboard Components Applications Configuration Resources

Home > Components > hello\_world

## Component: hello\_world

Description Euro store

History Edit Inventory Calendar Properties Templates Versions Processes Ma

### Processes

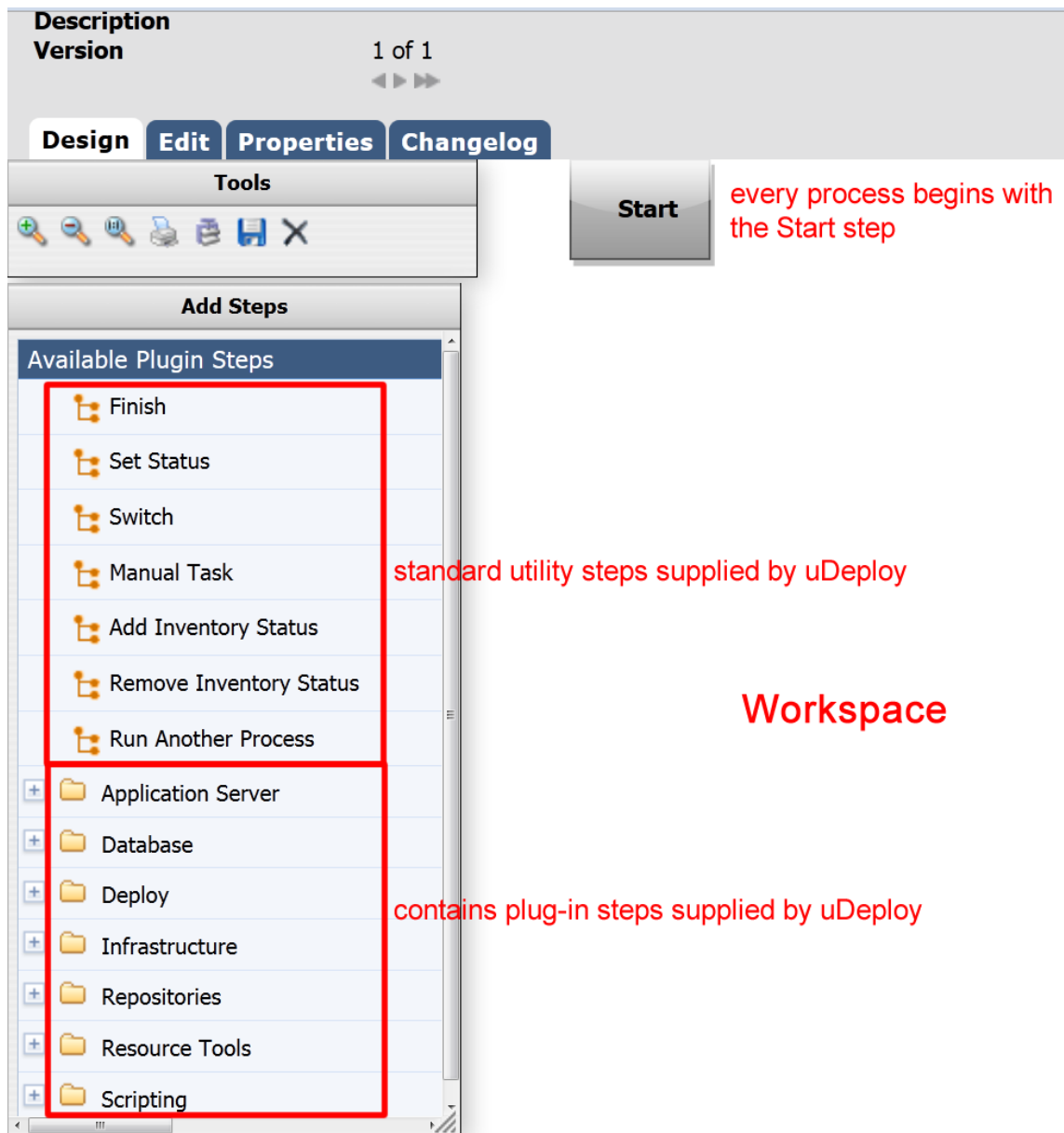
Process	Description	
<a href="#">hello_worldInstall</a>	hello_world remote install	<a href="#">Edit</a> <a href="#">Copy</a>

10 per page 1 record - [Refresh](#)

Show Inactive Processes

[Create New Process](#)

The **Process Design** pane is where the individual steps are defined.




**Figure 5: Process Design Pane**

The steps are listed in the **Available Plug-in Steps** list-box. Take a moment to expand the listings and review the available steps. Out-of-the-box, uDeploy comes the listed plug-in steps. In the next walk-through ([hello\\_worldWS](#)) you will learn how to add additional plug-ins.

3. In the **Available Plug-in Steps** box, expand the Artifacts item.
4. Drag the Download Artifacts by Label item into the design area and release it on the anchor point as shown in the following illustration.

The screenshot shows the UrbanDeploy web interface for the 'Process: hello\_worldInstall'. The breadcrumb trail is 'Home > Components > hello\_world > Processes > Process: hello\_worldInstall'. The page title is 'Process: hello\_worldInstall'. Below the title, there is a description 'hello\_world remote install' and a version '8 of 8'. There are tabs for 'Design', 'Edit', 'Properties', and 'Changelog'. On the left, there is a 'Tools' section with various icons, an 'Add Steps' section, and a list of 'Available Plugin Steps'. The 'Available Plugin Steps' list includes: Manual Task, Add Inventory Status, Remove Inventory Status, Artifacts (with a sub-item 'Download Artifacts By Label' highlighted by a red box), Upload Artifacts, and Verify Local Artifacts. On the right, there is a 'Start' button with a small green circle on its top edge, and a 'Finish' button with a gear icon. A red line connects the 'Download Artifacts By Label' step to the green circle on the 'Start' button.

**Figure 6: Adding a Step to an Anchor Point**

 **Note:** Most deployments should start with this step.

Releasing the mouse-pointer on the anchor point displays the **Edit Properties** dialog. The fields on this dialog are always tailored for the selected step.

The screenshot shows the 'Edit Properties' dialog box for a deployment step. The fields are as follows:

- Name \***: Download Artifacts By Label
- Repository URL \***: `$(p:server.url)/vfs`
- Repository ID \***: `$(p:component/code_station/rep)`
- Label \***: `$(p:version.name)`
- Directory Offset \***: .
- Includes \***: `**/*`
- Excludes**: (empty)
- Sync Mode**:
- Allow Failure**:
- Working Directory**: C:\UAT\appUAT
- Precondition**: None (Always Runs)
- Use Impersonation**:

Buttons: Save, Cancel

These fields, along with the fields for the other steps, are described in [Plug-in Integration](#) on page 98. For this exercise, we can achieve our goal by changing one field--**Working Directory**.

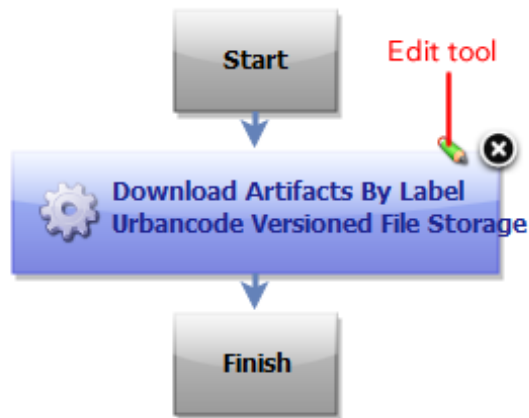
Recall that the goal for this deployment is to move the source files in the base-directory to another location. As you might guess, uDeploy provides several methods for accomplishing this goal; changing the **Working Directory** field here is one of the simplest.

5. Enter the path to the target directory you created at the beginning of the exercise, as we discussed in [Creating Components](#) on page 22.

If the field is left blank, the process will use the working directory defined earlier. Entering the path overrides the previous value and causes UrbanCode to place the source files in the specified location.

6. Use the **Save** button to save the step and close the dialog.

We can accept the default values for the other fields. If you need to edit the step properties, click the Edit tool on the step graphic.



**Figure 7: Edit tool**

7. Save the component by using the **Save** tool on the **Tools** menu.

Typically, we would define additional steps by dragging them onto the design area and defining them as we did here, but for this simple deployment the single step--`Download Artifacts by Label`--accomplishes the goal.

Once the process steps are defined, the final task is to define an application that uses the component.

## Hello World Application

Deployments are performed by applications. Applications bring together the component versions, environments, and application processes required to perform the deployment.

An environment is a collection of resources that host the application. Environments typically include host machines and uDeploy agents.

Application processes play a coordinating role in a deployment. Application processes are authored in a manner similar to component processes (see [Hello World Component Process](#) on page 26).

After creating an application, you perform the deployment by running the application.

### Creating an Application

1. On the **Navigation** bar, click the **Applications** tab.
2. On the **Applications** pane, click **Create New Application**.

Components are defined with the **Create New Application** dialog.



The screenshot shows the UrbanDeploy web interface. At the top, there is a navigation bar with 'Dashboard', 'Components', 'Applications', and 'Configuration'. Below this, a breadcrumb trail reads 'Home > Applications'. The main content area is titled 'Applications' and features a 'Create New Application' button. A table lists existing applications with columns for 'Application', 'Description', and 'Created'. A modal dialog box titled 'Create New Application' is open, containing the following fields:

- Name \***: hello\_world
- Description**: Hello world app
- Notification Scheme**: None (with a dropdown arrow and a help icon)

Below the 'Notification Scheme' dropdown, a list shows 'None' as the selected option and 'Default Notification Scheme' as an alternative. At the bottom of the page, the version 'UrbanDeploy 4.3.0-b1.214367' is visible.

3. Enter a name in the **Name** field.
4. Enter a description in the **Description** field.
5. Select the default value of **None** from the **Notification Scheme** drop-down list box.

uDeploy integrates with LDAP and e-mail servers which enables it to send event-based notifications. For example, the default notification scheme will send an e-mail when a deployment finishes. Notifications can also play a role in deployment approvals. See [Security](#) on page 90 for information about security roles.

6. Use the **Save** button when you are finished.

The **Application: name** pane is displayed. If you need to change your work, use the **Edit** tab.

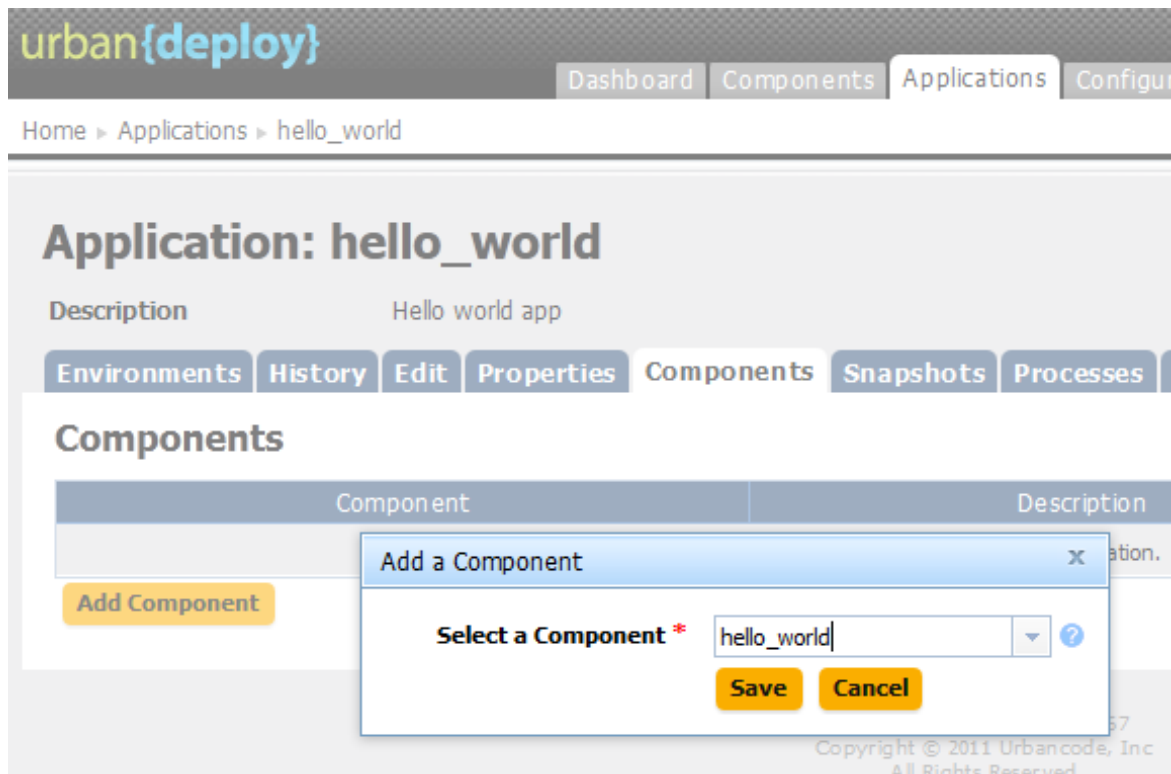
### Adding a Component to the Application

After the application is saved, the components it requires must be identified. We will add the *hello\_world* component we created earlier.

1. On the **Application: name** pane, click the **Components** tab.
2. Click the **Add Component** button.

An application must have at least one component.

3. If you created the *hello\_world* component described earlier (see [hello\\_world Component Version](#) on page 23), select *hello\_world* from the **Select a Component** list box.



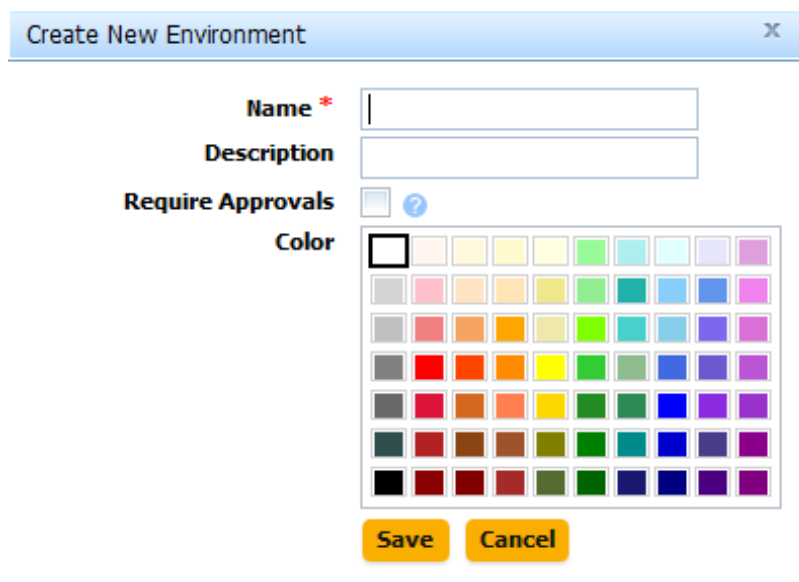
**Figure 8: Adding a component to an application**

4. Click the **Save** button.

The **Application: *name*** pane is redisplayed.

### Adding an Environment to the Application

1. On the **Environments** tab, click the **Create New Environment** button.



Before an application can run, it must have at least one environment created for it. An environment defines the resources (agents and machines) used by the application.

2. Use the **Create New Environment** dialog to define the environment.

The value in the **Name** field will be used in the deployment.

If you check the **Require Approvals** check box, uDeploy will enforce an approvals process. This is our first deployment so an uncontrolled environment will do fine--leave the box unchecked.

Selecting a color provides a visual identifier for the environment. Typically, every environment will be assigned its own color.

After saving your work, the **Environment: name** pane is displayed.

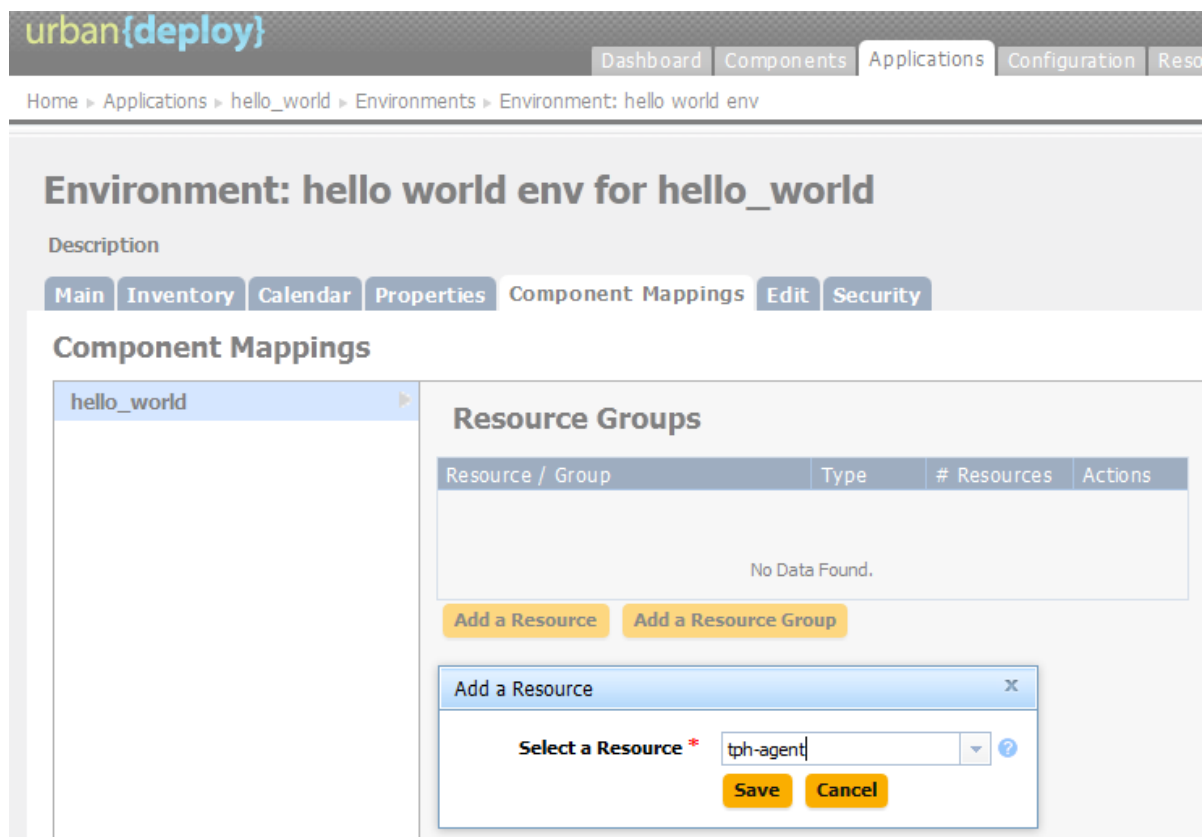
3. Click the **Component Mappings** tab.

The `hello_world` component we added earlier to the application is listed in the **Component Mappings** list box.

4. Click the **Add a Resource** button. The Add a Resource dialog is displayed.

5. In the **Add a Resource** list box, select the agent that was created when uDeploy was installed on your system.

While our example uses but a single resource, deployments can use many resources and *resource groups*. Resource groups provide a way to combine resources, which can be useful when multiple deployments use overlapping resources. See [Resources](#) on page 57 for information about resource groups.




**Figure 9: Adding a resource to an environment**

### Adding a Process to the Application

Now that our application has an environment, we are ready to create an application-level process that we can use to perform the deployment.

1. Click the breadcrumb trail to redisplay the **Application: name** pane.

**Figure 10: Environments Tab**

 **Note:** You might be wondering why you need to create an application-level process when the process you created for the component should be able to perform the deployment by itself. For a single-component deployment like *hello\_world*, an application-level process might not be required. You might also want to skip an application-level process when you are testing or patching a component. But for non-trivial deployments, especially deployments that have more than one component, you will want to create one or more application-level processes. Application-level processes enable you to combine components into a single deployment.

2. Click the **Processes** tab.
3. Click the **Create New Process** button. The **Create an Application Process** dialog is displayed.

**Figure 11: Create an Application Process dialog**

4. Enter a name in the **Name** field.
5. In the **Required Application Role** drop-down list box, accept the **None** default value.

This option enables you to restrict who can run this process. The available options are derived from the uDeploy Security System. For information about security roles, see [Security](#) on page 90.

6. In the **Inventory Management** drop-down list box, accept the default value of **Automatic**.

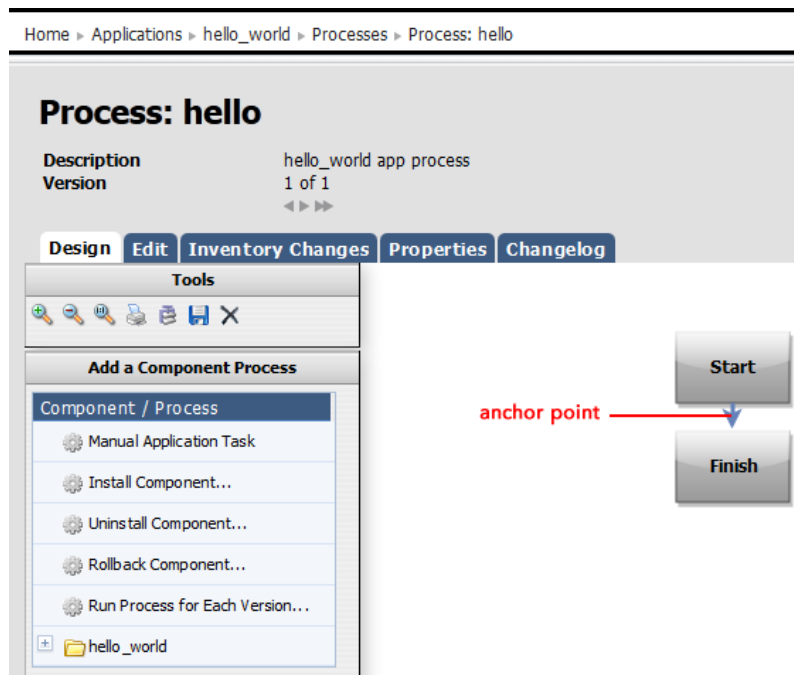
Automatic inventory management is sufficient for most applications. If you need to manually control inventory, select the **Advanced** option. See [Inventory](#) on page 110 for information about inventory management.

7. Use the **Save** button when you are finished.

## Designing the Process Steps

To create an application-level process, you define the individual steps as you did earlier ([Hello World Component Process](#) on page 26) when you used the **Process Design** pane to create the *hello\_world* component process.

1. On the **Application: hello\_word** pane, click the **Processes** tab.
2. Click the name of the application you defined earlier to display the **Process Design** pane.

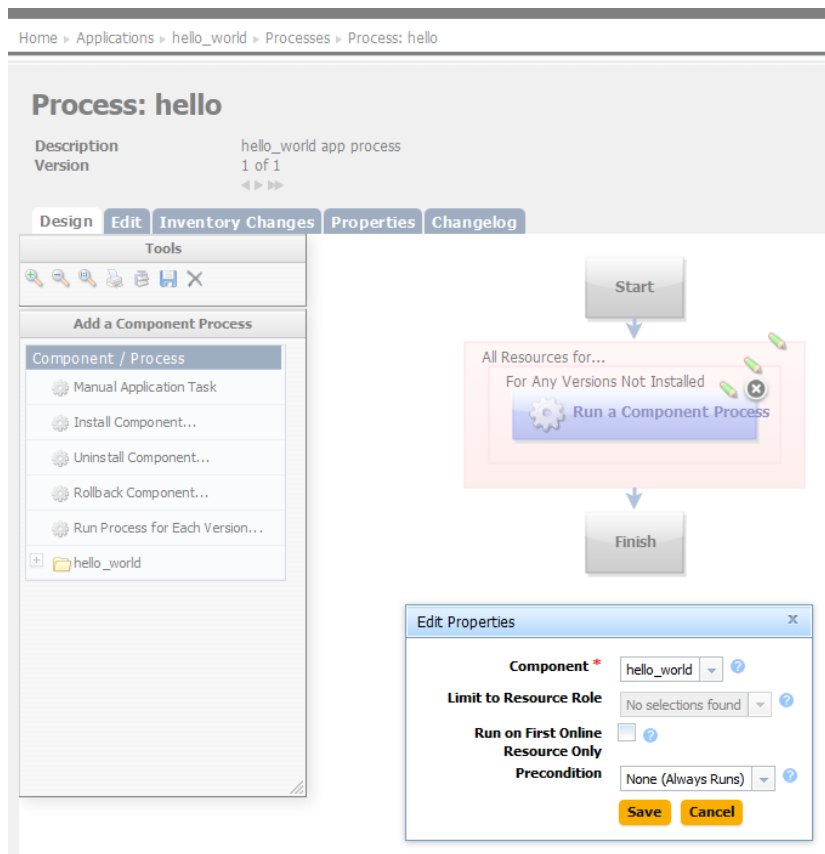


**Figure 12: Process Design Pane**

The out-of-box process steps are listed in the **Add a Component Process** list box.

3. Drag the **Install Component** step onto the design area and release the mouse pointer on the anchor point.

The step graphic is inserted into the design area and the **Edit Properties** dialog is displayed, as shown in the following illustration.



**Figure 13: Edit Properties Dialog**

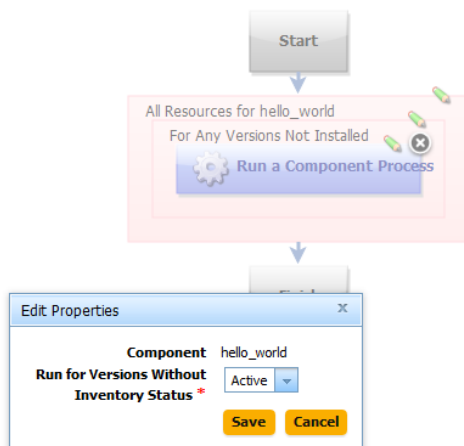
uDeploy will walk you through the three steps required to configure `Install Component`: first, select the component; second, select the version; finally, name the process. At each point, the **Edit Properties** dialog is updated with the required fields.

4. Select a component from the **Component** drop-down list box.

If you followed the *Quick Start Guide*, the `hello_world` component will be listed.

5. Accept the default values for the other fields (see [Applications](#) on page 61 for information about the other fields), and click **Save**.

The **Edit Properties** dialog is refreshed--the **Run for Versions Without Inventory Status** drop-down list box is displayed.

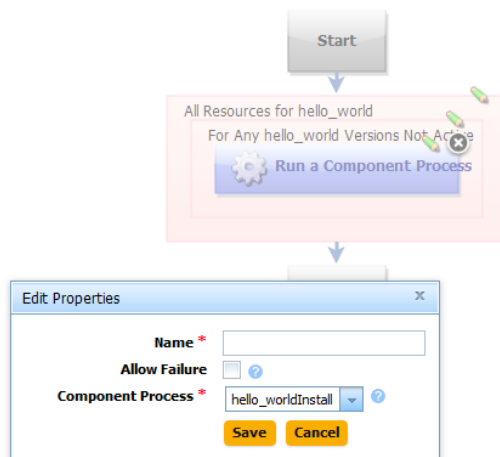


**Figure 14: Run for Versions Without Inventory Status field**

- Accept the default value `Active` (see [Applications](#) on page 61 for information about the other fields), and click **Save**.

`Active` means uDeploy will deploy any version not previously deployed and part of the inventory system. The `Staged` value is used when performing a rolling deployment. See [Applications](#) on page 61 for information about rolling deployments.

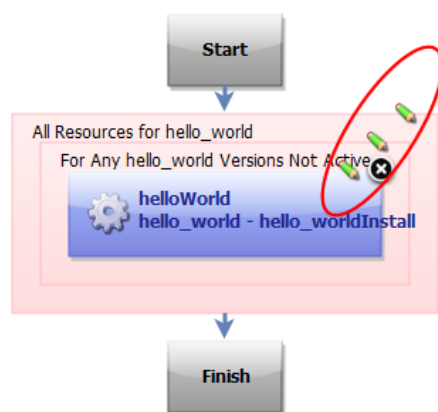
The dialog box is refreshed, as shown in the following illustration.



- Enter a process name in the **Name** field.
- Leave the **Allow Failure** check box unchecked. If checked, processes that perform several actions will continue processing even if one component fails. See [Applications](#) on page 61 for information about this option.
- Select a component process from the **Component Process** list box, then use the **Save** button to save the process step.

Components can have several processes defined for them.

The three steps are nested in the step graphic, as you can see from the following illustration. The first step is the outermost one. If you need to edit a step, click on the corresponding edit tool.



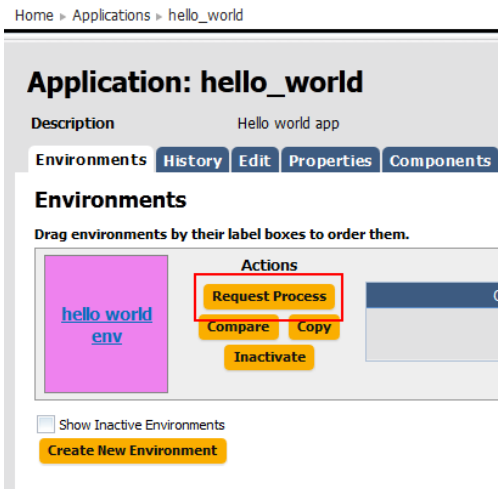
**Figure 15: Nested parameters**

- Finally, save the process by clicking the **Save** tool on the **Tools** bar.

## Running the Application

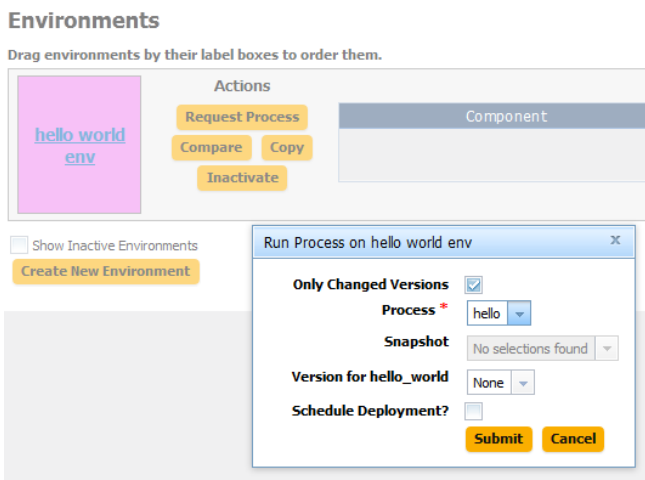
Now that the component, environment, and application are complete, you are ready to perform the deployment by running the application.

- On the **Application** pane, click the **Request Process** button for the environment you created earlier.



The **Run Process** dialog is displayed.

2. Leave the **Only Changed Versions** check box checked. For this deployment, we only want to run the application on changed (new) versions.



3. Select the process you created from the **Process** drop-down list box. Applications can have more than one process defined for them.

Because we did not create a snapshot of the application, the **Snapshot** field is inactive. See [Applications](#) on page 61 for information about snapshots.

4. Select **Latest Version** from the **Version** drop-down list box. This option ensures that the latest (or first and only) version is affected by the application.

Leave the **Schedule Deployment?** check box unselected. Selecting this option displays fields you can use to schedule the deployment.

5. Click the **Submit** button to run the application.

The **Application Process** pane is displayed.



### Application Process Request: hello\_world

Process [hello \(Version 2\)](#)  
Environment [hello\\_world\\_env](#)  
Date Requested 1/9/12 5:31 PM  
Requested By admin  
Scheduled For 1/9/12 5:31 PM  
[View Deployment Request](#)

Log Properties Manifest

	Component Process / Resource	Start	Duration	Status	Actions
P	All Available Resources for <a href="#">hello_world</a>	1/9/12 5:31:45 PM	0:00:04	Success	
S	<a href="#">top-agent</a>	1/9/12 5:31:45 PM	0:00:03	Success	
	<a href="#">helloWorld (1..4)</a>	1/9/12 5:31:45 PM	0:00:03	Success	<a href="#">Details</a>

Take a few moments to examine the information on this pane. Hopefully, you will see several **Success** messages in the **Status** field. To see additional information about the process, click the **Details** link in the **Actions** field.



---

# Part III

---

## Using uDeploy

---

### Topics:

- [Components](#)
  - [Resources](#)
  - [Applications](#)
  - [Deployments](#)
  - [Reports](#)
  - [Schedule Deployments](#)
-

## Components

---

Components represent deployable items along with user-defined processes that operate on them, usually by deploying them. Deployable items, or artifacts, can be files, images, databases, configuration materials, or anything else associated with a software project. Artifacts can come from a number of sources: file systems, build servers such as AnthillPro, as well as many others. When you create a component, you identify the source and define how the artifacts will be brought into uDeploy.

### Component Processes

A component process is a series of user-defined steps that operate on a component's artifacts. Each component has at least one process defined for it and can have several. A component process can be as simple as a single step or contain numerous relationships, branches, and process switches. Component processes are created with uDeploy's process editor. The process editor is a visual drag-and-drop editor that enables you to drag process steps onto the workspace and configure them as you go. As additional steps are placed, you visually define their relationships with one another. Process steps are selected from a menu of standardized steps. uDeploy provides steps for several utility processes, such as inventory management, and workflow control. Additional process steps are provided by plug-ins. A component process can have steps from more than one plug-in. See [Plug-in Integration](#) on page 98.

Additionally, you can create processes and configure properties and save them as templates to create new components. See [Component Templates](#) on page 55

### Component Versions and the CodeStation Repository

After defining a component's source and processes, you import its artifacts into uDeploy's artifact repository CodeStation. Artifacts can be imported automatically or manually. By default, a complete copy of an artifact's content is imported into CodeStation (the original artifacts are untouched). Each time a component is imported, including the first time, it is versioned. Versions can be assigned automatically by uDeploy, applied manually, or come from a build server. Every time a component's artifacts are modified and reimported, a new version of the component is created.

## Creating Components

In general, component creation is the same for all components. When creating a component, you:

1. Define and configure the component.

You name the component and identify the artifacts' source, such as AnthillPro, a file system, or Subversion. A component can contain any number of artifacts but they must all share the same source.

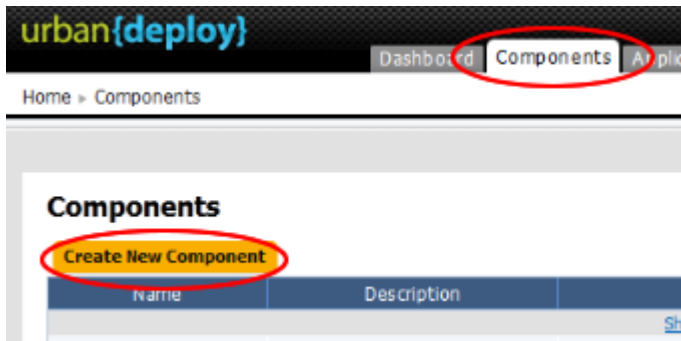
2. Assemble process(es).

A process defines what uDeploy does with the component's artifacts. A process might consist of any number of steps, such as starting and starting servers, and moving files. In addition to deploying, other processes can import artifacts and perform various utility tasks.

### Configuring Components

**To create a component:**

1. On the **Navigation** bar, click the **Components** tab.



2. On the **Components** pane, click **Create New Component**.

Components are defined with the **Create New Component** dialog box. Several fields displayed are the same for every source, while others depend on the source type selected with the **Source Config Type** field.

3. Enter the component's name in the **Name** field.

The name is used when assembling an application. If the component will be used by more than one application, the name should be generic rather than project-specific. For components that are project specific, a name that conveys something meaningful about the project should be used.

4. Enter a description in the **Description** field.

The optional description can be used to convey additional information about the component.

5. Optionally, select a template from the **Template** drop-down list.

"Template" refers to component templates. Any previously created templates are listed. A component can have a single template associated with it. The default value is `None`. See [Component Templates](#) on page 55.

If you select a template, the **Template Version** field is displayed. Use this field to instruct the component to use a specific template version, including configuration parameters and process. By controlling the version used, you can roll-out template changes as required. The default value is `Latest Version` which means the component will automatically use the newest version.



**Note:** If you select a template that has a source configured for it, the dialog box will change to reflect values defined for the template. Several fields, including the **Source Config Type** field, will become populated and locked. If this is not what you want, select **None**.

6. Select a plug-in from the **Status Plug-in** field.

A status plug-in provides information relating to the component's process and inventory conditions, such as process success or failure. If you previously created any status-related plug-ins, they will be listed here. The default value is **Default**, meaning that the component will use uDeploy-supplied utility-type steps. See [Plug-in Integration](#) on page 98.

7. Specify whether the component will inherit clean-up settings by checking the **Inherit Cleanup Settings** check box.

If checked, the component will use the values specified on the **System Settings** pane for the **Days to Keep Versions** and **Number of Versions to Keep** fields. The default value is unchecked. See [System Settings](#).

8. Enter the number of days to keep each component version in the **Days to Keep Versions** field.

To keep versions indefinitely, enter **-1**. If the **Inherit Cleanup Settings** check box is checked, the field is no longer displayed. The default value is **-1**. See [System Settings](#).

9. Enter the number of component versions to keep in the **Number of Versions to Keep** field.

To keep all versions, enter **-1**. If the **Inherit Cleanup Settings** check box is checked, the field is no longer displayed. The default value is **-1**. See [System Settings](#).

10. Select the source for the component artifacts from the **Source Config Type** drop-down list.

Selecting a value other than the default **None**, displays additional fields associated with your selection. Source-dependent fields are used to identify and configure the component's artifacts. A component's artifacts can have a single source. If you selected a template in the **Template** field that specified a source, this field is locked. See [Source Configuration Reference](#) for a description of the supported types and associated fields.

11. If you want uDeploy to automatically import component versions, check the **Import Versions Automatically** check box.

If checked, uDeploy will periodically poll the source location for new versions and import any it finds. The default polling period is 15 seconds. You can change the value with the **System Settings** pane. If left unchecked, you can manually create versions by using the **Versions** pane. By default, the box is unchecked.

12. If you want uDeploy to copy artifacts into CodeStation, leave the **Copy to CodeStation** box checked.

When checked, UrbanCode creates a tamper-proof copy of the component's artifacts and stores them in CodeStation. If unchecked, only meta data about the artifacts will be imported. UrbanCode recommends that the box be left checked.

13. Click the **Save** button to save the component.

Saved components are listed in the **Component** pane.

## Component Processes

A component process is a series of user-defined steps that operate on a component's artifacts. Each component has at least one process defined for it and can have several. Component processes are created with uDeploy's process editor. The process editor is a visual drag-and-drop editor that enables you to drag process steps onto the workspace and configure them as you go. Process steps are selected from a menu of standard steps. See [Process Editor](#) on page 48.

uDeploy provides steps for several utility processes such as inventory management and workflow control. Additional process steps are provided by plug-ins. Out-of-the-box, uDeploy provides plug-ins for many common processes, such as downloading and uploading artifacts, and retrieving environment information. See [Plug-in Integration](#) on page 98.

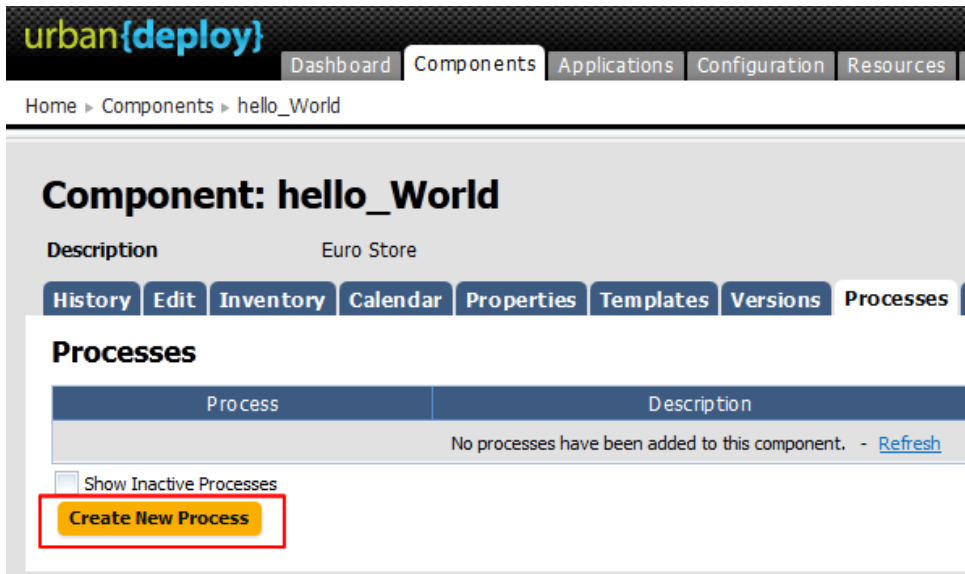
A frequently used process can be saved as a template and applied to other components. See [Component Templates](#) on page 55.

## Configuring Component Processes

A component process is created in two steps: first, you configure basic information, such as name; second, you use the process editor to assemble the process.

**To configure a component process:**

1. On the **Navigation** bar, click the **Components** tab.
2. On the **Components** pane, click the on the name of the component you want to use from among the listed components.
3. On the **Component: *Name\_of\_selected\_component*** pane, click the **Processes** tab.
4. Click the **Create New Process** button.



5. In the **Create New Process** dialog, enter a name in the **Name** field.

The screenshot shows a 'Create New Process' dialog box. It has a title bar with a close button (X). The form contains the following fields and controls:
 

- Name \***: A text input field, highlighted with a red box.
- Description**: A text input field.
- Default Working Directory \***: A text input field containing the value `${p:resource/work.dir}/${p:comt`.
- Requires a Version**: A checkbox that is currently unchecked, with a help icon (?) to its right.
- Required Component Role**: A dropdown menu currently set to 'None', with a help icon (?) to its right.
- At the bottom, there are 'Save' and 'Cancel' buttons.

The name and description typically reflect the component's content and process type.

6. Optionally, enter a description in the **Description** field.

If the process will be used by several applications, you can specify that here.

7. Enter a location in the **Default Working Directory** field.

This is the location where the process steps will be executed. The default value enables the process to work in different environments. For most processes the default value is fine, but if you want to deploy files to where they are running, for example, you might want to change the value. If you change the default value, the process might not work in every environment visited by the component.

8. If you want users to supply a component version, check the **Requires a Version** check box.

When checked, users will be prompted for a component version at run-time. If the process requires a specific component version, check this box. By default, the box is unchecked.

9. If you want to restrict who can run the process, select a value from the **Required Component Role** drop-down list.

The available options are derived from the uDeploy security system. The default value is None, meaning anyone can run the process. For information about security roles, see [Security](#) on page 90.

10. Click the **Save** button to save your work.

## Process Editor

After configuring a process with the **Create New Process** dialog, use the process editor to assemble the process.

### To Display the Process Editor

1. On the **Component: *name*** pane, click the **Processes** tab.
2. Click on the name of the process you want to edit.

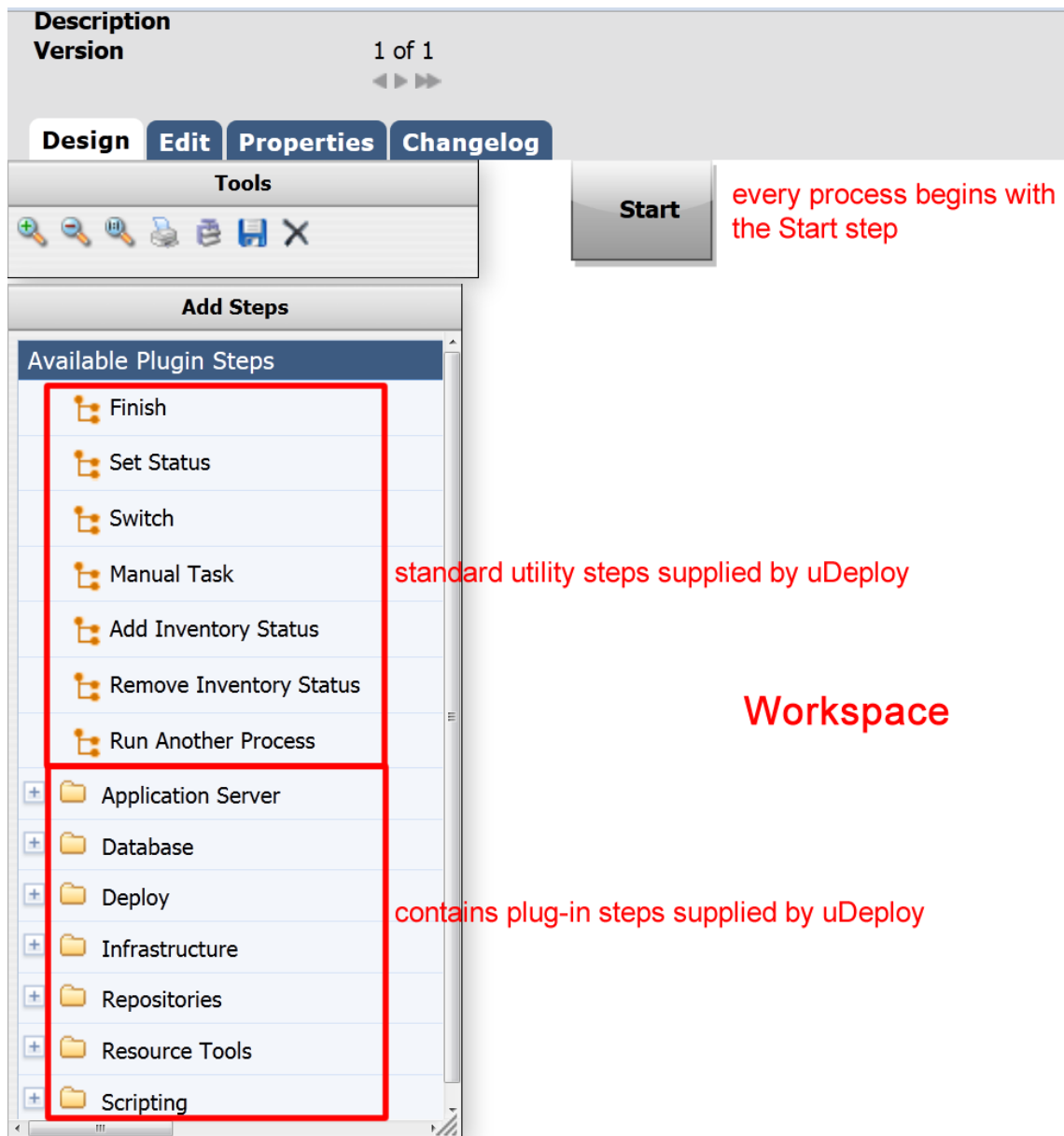
The screenshot shows the uDeploy web interface. At the top, there's a navigation bar with 'urban{deploy}' logo and tabs for 'Dashboard', 'Components', 'Applications', 'Configuration', and 'Resources'. Below the navigation bar, the breadcrumb path is 'Home > Components > hello\_world'. The main content area is titled 'Component: hello\_world' and has a description 'Euro store'. A row of tabs includes 'History', 'Edit', 'Inventory', 'Calendar', 'Properties', 'Templates', 'Versions', 'Processes', and 'Ma'. The 'Processes' tab is active, showing a table with the following data:

Process	Description	
<a href="#">hello_worldInstall</a>	hello_world remote install	<a href="#">Edit</a> <a href="#">Copy</a>

Below the table, there are pagination controls showing '10 per page' and '1 record - Refresh'. There is also a checkbox labeled 'Show Inactive Processes' which is checked, and a yellow button labeled 'Create New Process'.

The **Process Design** pane is displayed.





**Figure 16: Process Design Pane**

Available steps are listed in the **Available Plug-in Steps** list. uDeploy provides several utility steps and plug-ins which are highlighted in the accompanying illustration. The illustration also shows several user-installed plug-ins.

**Using the Process Editor**

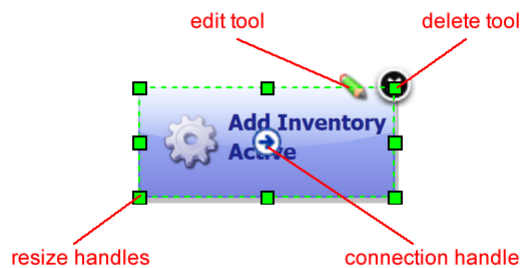
When the **Process Design** pane opens, the **Design** view is displayed. Processes are assembled with the **Design** view. Several other views can be displayed by clicking the associated tab:

<b>Edit</b>	Displays the <b>Edit</b> view where you can change process parameters. See <a href="#">Component Processes</a> on page 46.
<b>Properties</b>	Displays the <b>Properties</b> view where you can create and change process properties. See <a href="#">Process Properties</a> on page 53.

**Changelog**

Displays the **Process Changelog** view. This view provides a record for every process change--property add or delete, and process save or delete.


In outline, processes are assembled by dragging individual steps onto the workspace and configuring and connecting them as they are placed. When a step is dragged onto the workspace, a pop-up is displayed that is used to configure the step. Once configured and the pop-up closed, relationships between steps are formed by dragging *connection handles* between associated steps.



**Figure 17: Typical Process Step**

Graphically, each step (except for the Start step which cannot be deleted or edited) is the same and provides:

<b>edit tool</b>	displays the step configuration pop-up where you can modify configuration parameters
<b>delete tool</b>	removes the step from the workspace
<b>resize handle</b>	enables you to resize the step graphic
<b>connection tool</b>	used to create connections between steps

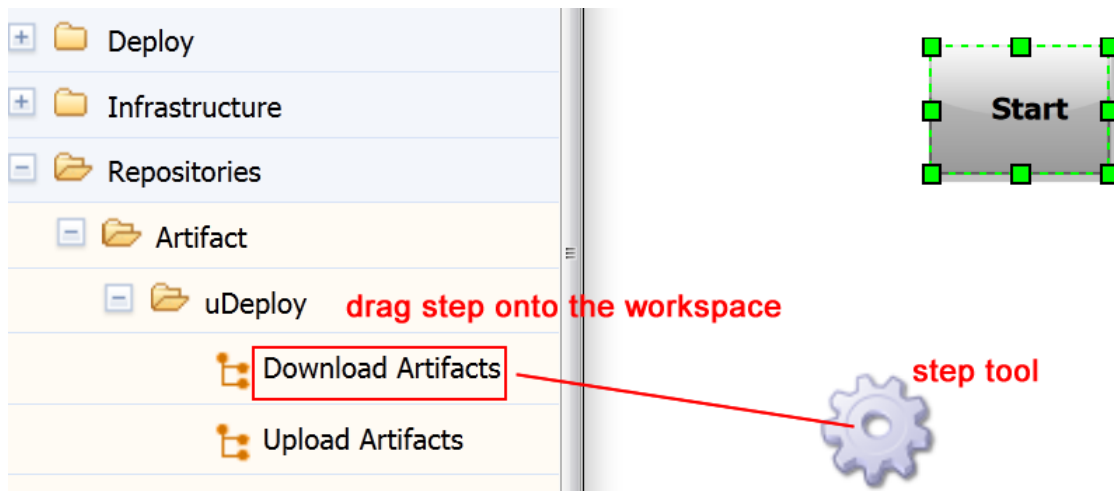
 **Note:** If you delete a step, its connections (if any) are also deleted.

### Adding Process Steps

#### To add a step:

1. In the **Available Plug-in Steps** list, click and hold down the mouse on the step you want to use, and drag it onto the workspace.

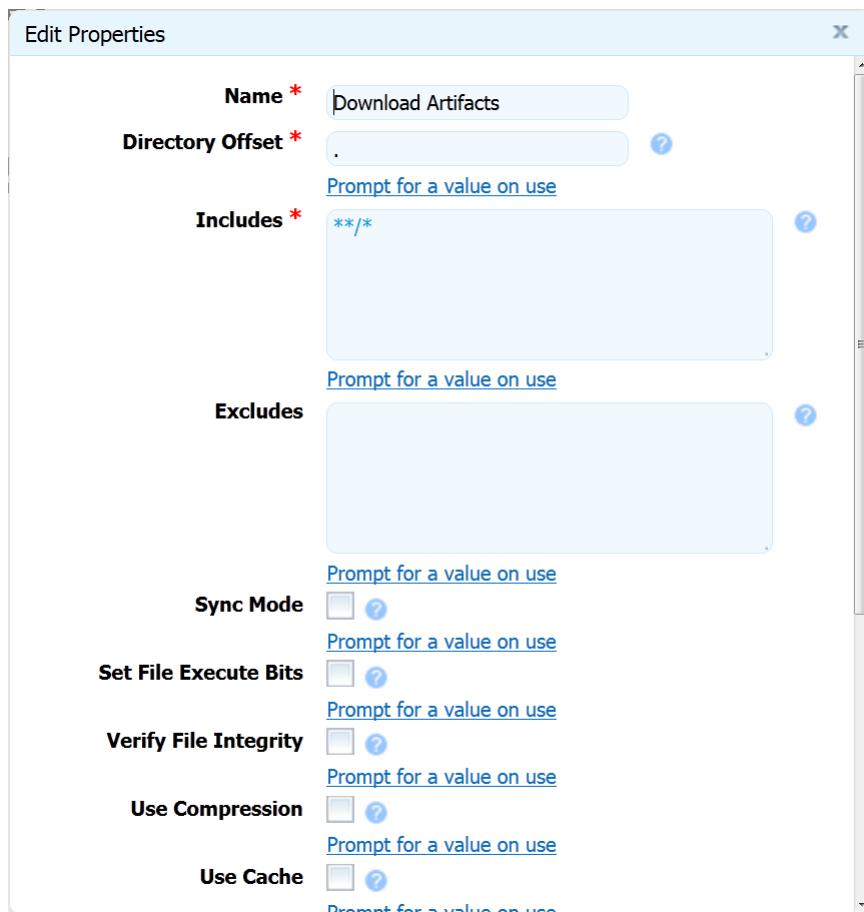
The cursor changes to the *step tool*.



**Figure 18: Adding a Step**

2. Release the step tool over the workspace.

The **Edit Properties** pop-up is displayed. Because connections are created after configuring the step's properties, you can place the step anywhere on the workspace. Steps can be dragged and positioned at any time. See [Plug-in Integration](#) on page 98 for information about configuring specific steps.



**Figure 19: Typical Edit Properties Pop-up**

Configuration dialogs are tailored to the selected step--only parameters associated with the step type are displayed.

3. After configuring the step's properties, save the step by clicking the **Save** button.

The step is in the workspace and ready to be connected to other steps. If you change your mind, click the **Cancel** button to remove the step from the workspace. You can add connections immediately after placing a step or place several steps before defining connections.

### Connecting Process Steps

Connections control process flow. The originating step will process before the target step. Creating a connection between steps is a simple process: you drag a connection from the originating step to the target step. Connections are formed one at a time between two steps, the originating step and the target step.

#### To create a connection:

1. Hover the cursor over the step that you want to use as the connection's origin.

The connection tool is displayed.

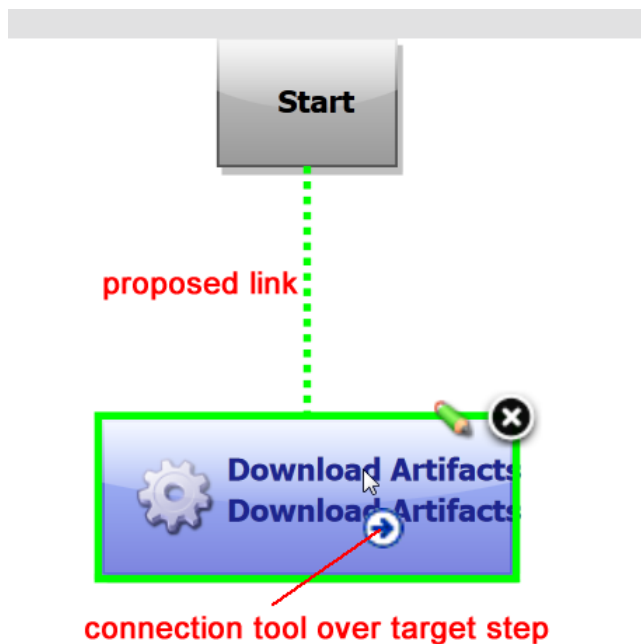
connection tool



**Figure 20: Connection Tool**

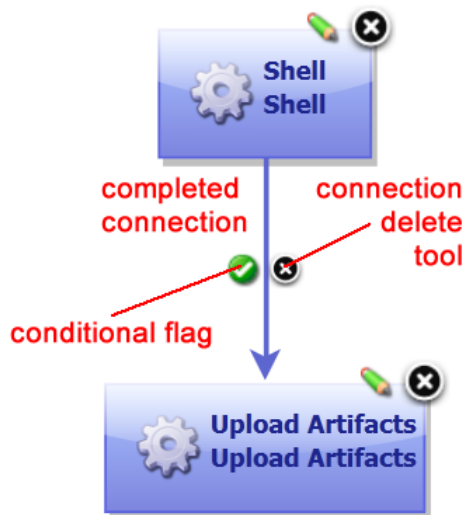
2. Drag the connection tool over the target step.

The step beneath the connection tool is highlighted.



**Figure 21: Dragging the Connection Over a Target Step**

3. Release the connection tool over the target step to complete the connection.



**Figure 22: Completed Connection**

Each connection has a connection delete tool, *conditional flag*, and might have others depending on the originating step. See [Switch Steps and Conditional Processes](#) on page 54. Remove a connection by clicking on the delete tool.

See [Standard Component Process Steps](#) on page 103 for information about configuring standard steps.

## Process Properties

A processing property is a way to add user-supplied information to a process. A running process can prompt users for information and then incorporate it into the process. Properties are defined with the **Edit Property** dialog.

### To define a property:

1. On the **Properties** tab, click the **Add Property** button.

**Figure 23: Edit Properties Dialog**

2. In the **Edit Properties** dialog, enter a name in the **Name** field.
3. Optionally, enter a description in the **Description** field.
4. Enter a label in the **Label** field.

The label will be associated with the property in the user interface.

5. If the property is required, check the **Required** check box.

Default value is unchecked--not required.

- Specify the type of expected value with the **Type** drop-down list box.

Supported types are: text, text area, check box, select, multi select, and secure.  
Default type is text.

- In the **Default Value** field, enter a default value (if any).

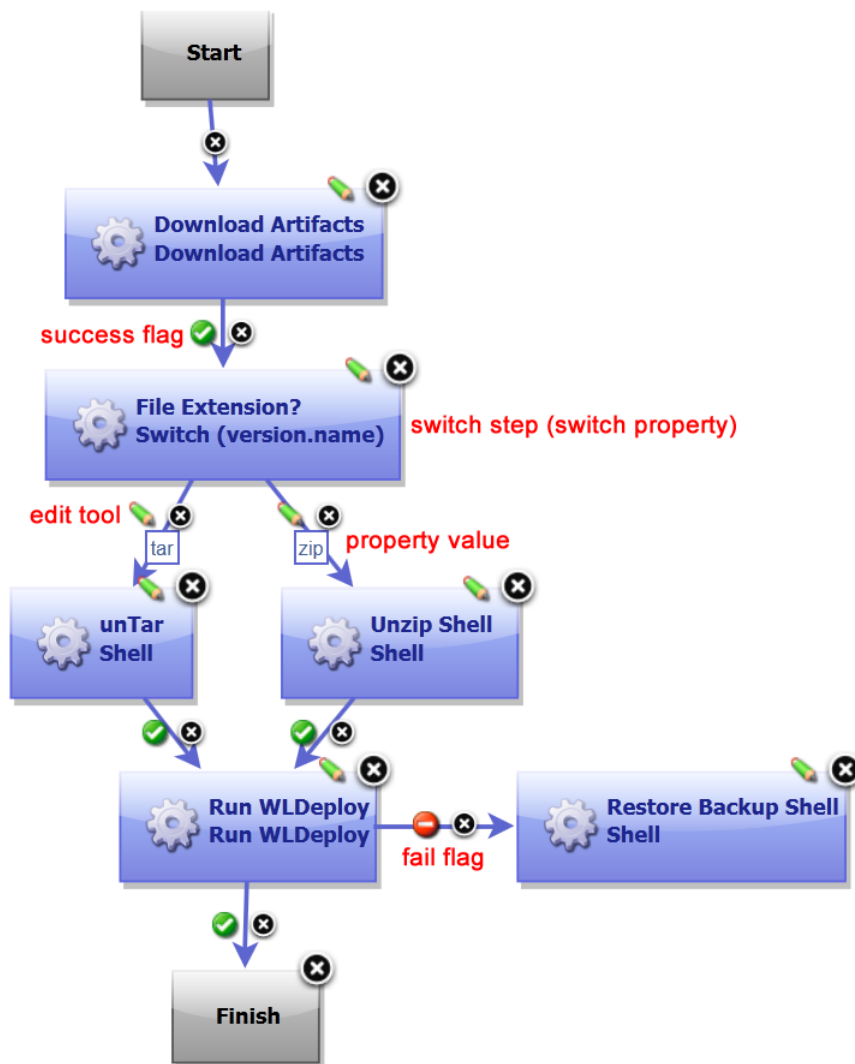
- To save your work, click the **Save** button. To discard changes, use the **Cancel** button.

To use a property in a process, reference it when you configure (see [Component Processes](#) on page 46) a step that uses it.

### Switch Steps and Conditional Processes

Every connection (except connections from the Start step) has a delete tool and conditional flag. The conditional flag enables you to set a condition on a connection. The condition refers to the processing status of the originating step--success or failure. Possible flag conditions are: *success* (the process completed successfully), *fail* (the process did not finish successfully), or *both* (accept either status). By default, all connections have the flag set to checked (true), meaning the originating step must successfully end processing before the target step starts processing.

To change a flag's value, cycle through possible values by clicking the flag.



**Figure 24: Process with Switch Step**

A *switch step* is a uDeploy-supplied utility step that enables process branching based on the value of a property set on the step. [Figure 24: Process with Switch Step](#) on page 54 illustrates a switch step. In this case, the switch

property is `version.name`. The connections from the switch step represent process branches dependent on the value of `version.name`. In this example, regardless of which branch is taken, the process will proceed to the `Run WLDeploy` step. Note that `Run WLDeploy` has success and fail conditions.

See [Plug-in Integration](#) on page 98 for information about configuring specific steps.



**Note:** If a step has multiple connections that eventually reach the same target step, determining whether the target will execute depends on the value of the intervening flags. If all of the intervening connections have success flags, the target will only process if all the steps are successful. If the intervening connections consist of an assortment of success and fail flags, the target will process the first time one of these connections is used.

For a process to succeed, execution must reach a Finish step. If it does not end with Finish, the process will fail every time.

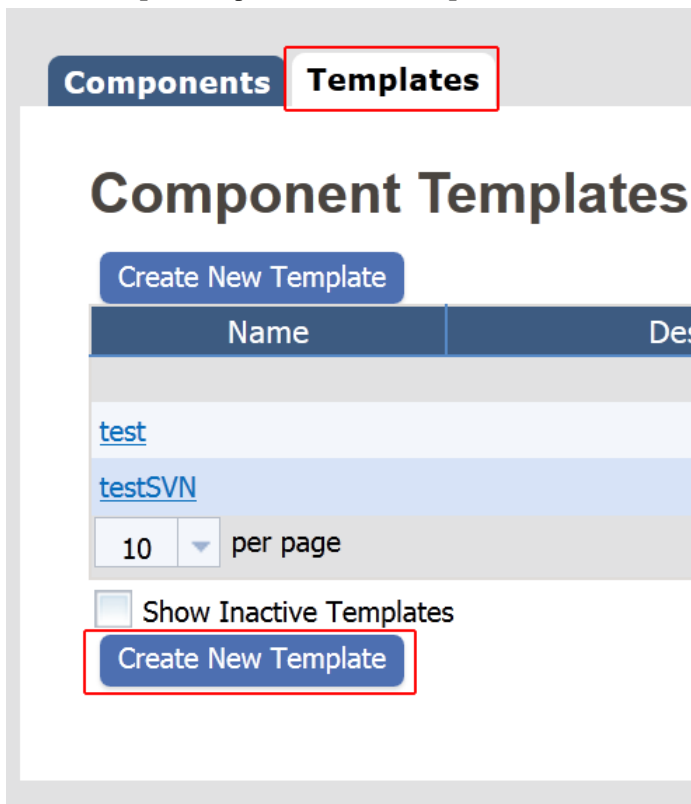
## Component Templates

Component templates enable you save and reuse component processes and properties. Components based on templates inherit the template's properties and process.

### Creating Templates

To create a template:

1. On the **Components** pane, click the **Templates** tab.



2. Click the **Create New Template** button.

The **Create New Component Template** pop-up is displayed.

3. Enter the template's name in the **Name** field.
4. Enter a description in the **Description** field.


The optional description can be used to convey additional information about the template.

5. Select a plug-in from the **Status Plug-in** field.

If you previously created any status-related plug-ins, they will be listed here. The default value is `Default`, meaning that the template will have uDeploy-supplied steps available for use. See [Plug-in Integration](#) on page 98.

6. Select the source for the artifacts from the **Source Config Type** drop-down list.

Selecting a value other than the default `None`, displays additional fields associated with your selection. Source-dependent fields are used to identify and configure the artifacts. If you select a source, components based on the template will use the same source. See [Source Configuration Reference](#) for a description of the supported types and associated fields.

 **Note:** If you select a source, any properties you configure will be set for any components created with the template.

7. Click the **Save** button to save the template.

Saved templates are listed in the **Component Templates** pane.

You create a process for the template in the same way processes are created for components. For information about creating component processes, see [Process Editor](#) on page 48.

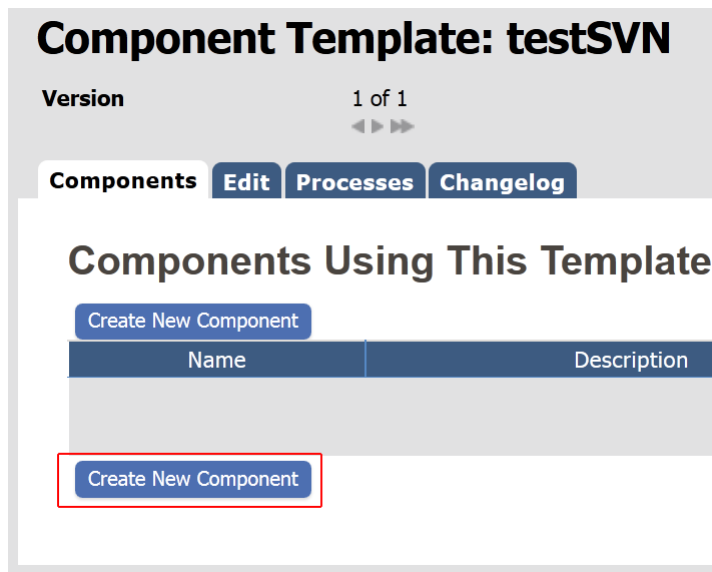
## Using Templates

When you create a component based on a template, the component inherits the template's process (if any, see [Component Processes](#) on page 46), and properties (if any, [Creating Components](#) on page 44).

### To create a template-based component:

1. In the **Component Templates** pane, click the name of the template you want to use.
2. In the **Component Template *template name*** view, click the **Create New Component** button.





**Figure 25: Component Template *template name*View**

The **Create New Component** dialog box is displayed. This dialog is used to configure component properties. Any properties defined in the template will be defined here. If a source was selected for the template, the source is set here and the **Source Config Type** field is locked. For information about using this dialog, see [Creating Components](#) on page 44

3. After configuring editable properties, save the component by clicking the **Save** button.

Templates used to create components are listed in the **Templates** view.

## Components

Create New Component	
Name	Template
<a href="#">DBSchema</a>	Dev
<a href="#">eStore-UAT</a>	
<a href="#">eStoreDev</a>	
<a href="#">JPetStore-SIT</a>	
<a href="#">suportStaging</a>	Dev
<a href="#">SVNtph</a>	
<a href="#">tomSource</a>	

10 per page

Show Inactive Components

Create New Component

Components created from templates are listed in the **Components** view.

## Resources

To run a deployment uDeploy requires an agent, or resource, on the target machine. Typically, at least one agent is installed in every environment the application must pass through on its way to production. A typical production pipeline may be SIT, UAT, PROD (the application must pass through two testing Environments and then can be pushed to Production). In this scenario, at least three agents need to be installed: one per Environment.



**Note:** When configuring resources for a production instance of uDeploy, you will need to take the Environmental differences into consideration, which may require gathering some information in order to fully roll out uDeploy. The Getting Started section includes some general guidelines for setting up and using uDeploy. See [Getting Started](#) on page 22.

urban{deploy} Hello admin | [Logout](#)

Components Applications Resources Deployment Calendar Work Items Settings

Home » Resources

Resources Groups Roles

### Resources

Name	Type	Description	Status	Actions
<a href="#">Show Filters</a>				
<a href="#">Master</a>			Online	<a href="#">Edit</a> <a href="#">Restart</a>
<a href="#">Deploy SIT</a>			Online	<a href="#">Edit</a> <a href="#">Restart</a>
<a href="#">Deploy UAT</a>			Online	<a href="#">Edit</a> <a href="#">Restart</a>
<a href="#">Deploy PROD</a>			Online	<a href="#">Edit</a> <a href="#">Restart</a>

10 per page 4 records - [Refresh](#) << 1 / 1 >>

[Create New Resource](#)

**Figure 26: Resources Pane**

To successfully deploy the application to the different environments, at least one agent needs to be installed in every environment; however, many users will install multiple agents per environment: this is usually the case where the different components run on different machines within a given Environment.

Resources Groups Roles

### Resource Groups

**To view and select groups:**  
Click a group name to select it or view its contents in the detail pane.  
Hold CTRL (or Command on a Mac) and click on additional groups to select or deselect them.

**To move or copy groups:**  
Drag a group (or multiple selected groups) into another group to move them into that group.  
Hold CTRL (or Command on a Mac) before dropping to copy groups instead of moving them.

Name	Size	Actions
<input checked="" type="checkbox"/> All Resource Groups	1 (0)	
PROD	0 (0)	
<input checked="" type="checkbox"/> SIT	1 (1)	
UAT	1 (1)	

**Figure 27: Resource Groups Pane**

Whether you need one or multiple resources per environment is determined by your current infrastructure, deployment procedures, and other requirements: Many UrbanDeploy users have differences among the different Environments; e.g., in SIT they need only to deploy a Component to one machine; however, for UAT, they must

deploy the Component to multiple machines. Under this scenario, you would configure Sub-groups for the single agent in the SIT Environment and then set up individual Resources for each agent in the UAT Environment.

**Resources** **Groups** **Roles**

## Resource Groups

**To view and select groups:**  
Click a group name to select it or view its contents in the detail pane.  
Hold CTRL (or Command on a Mac) and click on additional groups to select or deselect them.

**To move or copy groups:**  
Drag a group (or multiple selected groups) into another group to move them into that group  
Hold CTRL (or Command on a Mac) before dropping to copy groups instead of moving them.

Name	Size ?	Actions
All Resource Groups	1 (0)	
SIT	1 (1)	
APP	1 (1)	
DB	1 (1)	
WEB	1 (1)	
UAT	1 (1)	

Figure 28: Sub-Groups

## Resource Groups

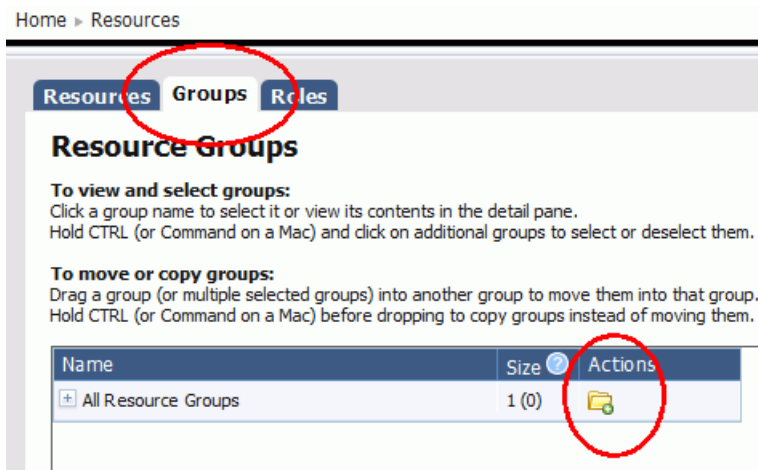
uDeploy uses the concept of resource groups to help you organize and manage the agents installed in different environment throughout the network. You need to create at least one resource group per installed agent, as when configuring your Processes you will need to select the appropriate Group. What groups you create and how you organize the groups, e.g., using subgroups, depends on your existing organizational processes and infrastructure.



**Note:** Before continuing, ensure that at least one agent has been installed in a target environment (for evaluation purposes, the agent can be on the same machine as the server).

### Creating a Resource Group

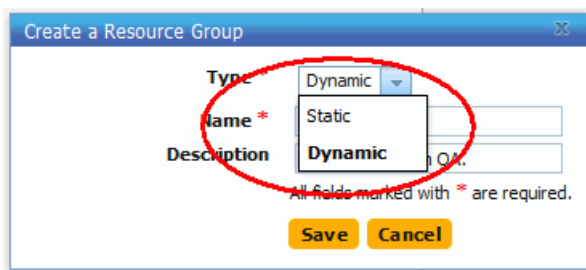
1. Go to **Resources** > **Groups**. and click on the folder icon.



**Figure 29: Action Tool**

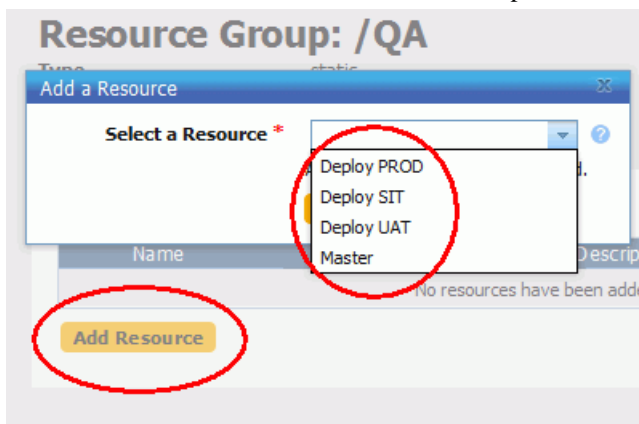
- For the Type, most often Static is used.

Name and description. Typically, the name will correspond to either the Environment the Resource participates in, the Component that uses the Resource Group, or a combination of both (e.g., SIT, DB, or SIT-DB). What description you give depends on how you intend to use the Resource that this Group is assigned to, etc.



**Figure 30: Create a Resource Group Dialog**

- Once the Resource has been created, select the pencil icon to edit the Group.



**Figure 31: Add a Resource Dialog**

- Once you assign a Group to a Resource, you add Subresources. Subresources enable you to apply logical identifiers, or categories, within any given Group. During deployment configuration, you can Select a given Subresource that the Process will run on. To create a Subresource, select the New Resource icon for the Group. Configuration is similar to Resource Group creation.

Name	Size	Actions
All Resource Groups	1 (0)	
PROD	0 (0)	
QA	1 (1)	

**Figure 32: Sub-resources**

## Setting Roles

Roles enable you to further refine how a Resource is utilized, and are similar to Subresources. For most Deployments, you will not need to define a Role. During Process configuration, you select a specific role when determining the resource. A role can be used to set up UrbanDeploy for rolling deployments, balancing, etc. For example, you can set up your Process to only deploy to a percentage of targets first; add a manual task in the middle of the Process that requires a user to execute (e.g., after they have tested the partial deployment); and then once the manual task has completed the rest of the Process is assigned a second role responsible for deploying to the rest of the target machines.

## Next Steps

With the Resources configured, it is now possible to configure a deployment. To get started, you will need to first set up a Component Version, which corresponds to the artifacts you want to deploy. See [Components](#) on page 44..


## Applications

---

Applications are responsible for bringing together all the components that need to be deployed together. This is done by defining the different versions of each component as well as defining the different environments the components must go through on the way to production. In addition, Applications also map the constituent hosts and machines (called resources) a component needs within every environment.

Applications also implement automated deployments, rollbacks, etc. These are called Processes; however, at the Application level Processes are only concerned with the Components and Resources necessary for deployment, etc. -- differentiating Application-processes from those of Components (which are concerned with running commands, etc.).

Applications also introduce Snapshots to manage the different versions of each Component. A Snapshot represents the current state of an Application in the Environment. Typically, the Snapshot is generated in an Environment that has no Approval gates -- called an uncontrolled Environment. For most users, the Snapshot is pushed through the pipeline.

 **Note:** Before configuring an Application, you will need to ensure that at least one agent has been installed in a target environment (for evaluation purposes, the agent can be on the same machine as the server). In addition, you will also need to add at least one Resource Group to the agent. See [Resources](#) on page 57.

## Environments

An Environment is a collection of Resources that host the Application. Environments typically include host machines and UrbanDeploy agents. When a deployment is run, it is always done so in an Environment. While Environments are collections of Resources, Resources can vary per Environment.

For example, Environment 1 may have a single web server, a single middleware server, and a single database server, that must be deployed to; UrbanDeploy represents these as three, separate Resources running in Environment 1. Environment 2, however, may have a cluster of Resources that the same Application must be deployed to. UrbanDeploy compensates for these differences with Resource Groups (more at Resources by keeping an Inventory of everything that is deployed to each Environment: UrbanDeploy knows exactly the Environment and Server(s) where the Application was deployed to: and tracks the differences between the Environments.

## Processes

Processes play a coordination role. They are authored using a visual drag-n-drop editor, and composed of Steps that call the Component Processes. For example, to deploy the Application you may invoke a Process called Deploy. This Deploy Process would in turn call out to the requisite Components and execute the deployment.

## Snapshots

Snapshots specify what combination of Component versions you deploy together. They are models you create before deploying the Application. A Snapshot specifies the exact version for each Component in the Application. When a Snapshot is created, UrbanDeploy gathers together information about the Application, including the Component versions, for a given Environment. Typically, the Snapshot is generated in an Environment that has no Approval gates -- called an uncontrolled Environment. For most users, the Snapshot is pushed through the pipeline. Typically, one of the Environment will always remain uncontrolled to allow for Snapshots. When a successful deployment has been run in the uncontrolled Environment, a Snapshot is created based on the Application's state within the Environment: thus capturing the different versions of the Components at that time. As the Application moves through various testing Environments, for example, UrbanDeploy ensures that the exact versions (bit for bit) are used in every Environment. Once all the appropriate stages and Approvals for a Snapshot are complete, the Snapshot is pushed to Production.

## Creating Applications

Before configuring an Application, you will need to ensure that at least one agent has been installed in a target environment (for evaluation purposes, the agent can be on the same machine as the server). In addition, you will also need to add at least one Resource Group to the agent. See [Resources](#) on page 57.

1. To get started go to Applications > Create New Application.

Name and Description. Typically the name and description correspond to the application you plan on deploying.

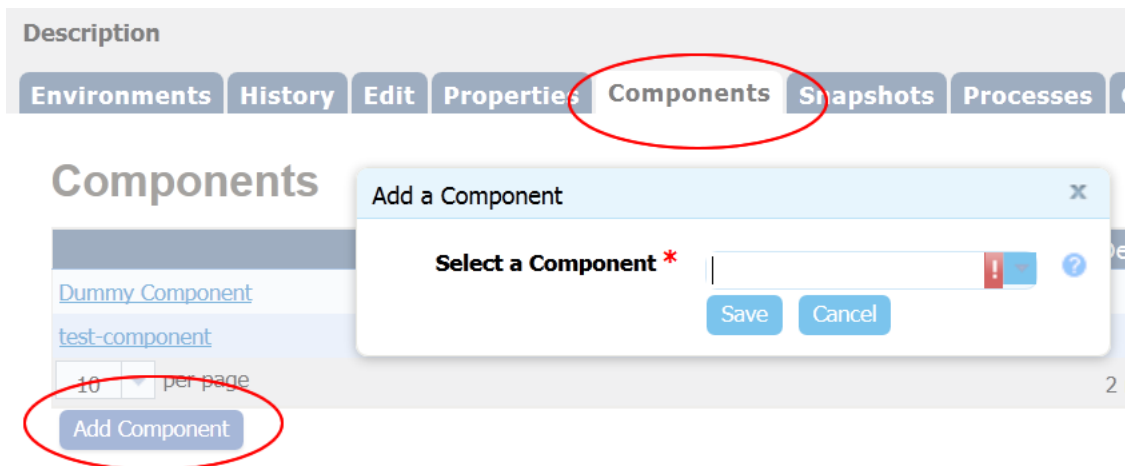
The screenshot shows a 'Create New Application' dialog box with the following fields:

- Name \***: hello\_world
- Description**: Hello world app
- Notification Scheme**: A dropdown menu with 'None' selected and 'Default Notification Scheme' as an option.

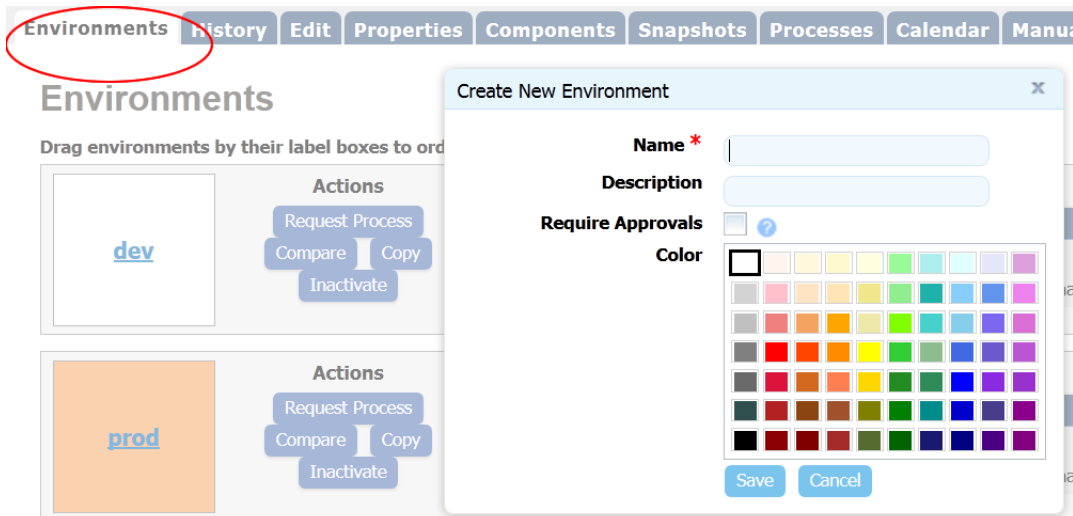
Notification Scheme. UrbanDeploy includes integrations with LDAP and e-mail servers that enable it to send out notifications based on events. For example, the Default Notification scheme will send out an e-mail when an Application Deployment fails or succeeds. Notifications also play a role in approving deployments: UrbanDeploy can be configured to send out an e-mail to either a single individual or to a group or people (based on their security role) notifying them that they need to approve a requested deployment. See [Notifications](#) on page 103.

2. Next, add at least one Component to the Application. If you have not already configured at least one Component, you will not be able to set up an Application. Before continuing, see Components. In UrbanDeploy, Applications are where you bring the different Components (including Versions and Processes) together so they can be deployed as a single unit. For example, a typical web application may have three tiers (WEB, APP, DB), or Components, required for the Application to run. It is at the Application level that these Components are brought together.

Name and Description. Typically the name and description correspond to the application you plan on deploying.



3. Configure Application Environments. If not already done so, install an agent, or agents, in your target Environments (e.g., a machine in SIT, UAT, etc.). If no agents are available, you must install at least one before continuing (see Agent Installation for instructions). Before you can run a deployment, you must also define at least one Environment for the Application and associate the Component with a machine (e.g., an agent on the target machine) in the Environment. This initial Environment is typically "uncontrolled," meaning that no Approvals are required to deploy to the Environment. This uncontrolled environment is where a Snapshot of the Application will be taken, and it is this Snapshot which will be moved to the other, secured, Environments. Once you set up the unsecured Environment, it may be a good idea to create an additional, secure Environment as well (see also Create Controlled Application Environments and Approvals Process). This is done on the Environments tab.

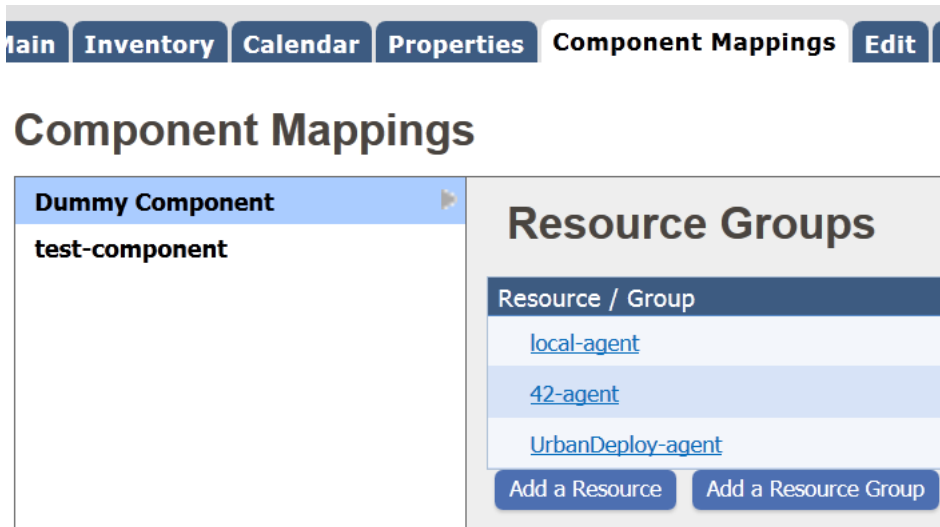


**Name and Description.** The name you give here will be used as part of the deployment process, and typically corresponds to the target Environment. For example, if you are deploying to an integration environment, SIT would be appropriate.

**Require approvals.** If you check the box, UrbanDeploy will enforce an approvals process before the deployment can be successfully executed in the Environment you are setting up. Here, we are assuming you are configuring the first deployment to an uncontrolled Environment, which is the most common approach. Once the deployment has been successful, you can configure an approvals process as the application moves along the development pipeline. If you are setting up more than one Environment, consider creating an Approvals Process for at least one of them.

**Color.** The color enables you to create a visual identifier for each Environment. Typically, every Environment will be assigned its own color.

- Map the Component to a Resource in the Environment. Once you have added the Environment to the Application, UrbanDeploy needs to know where Component artifacts should be sent to. This is determined by selecting a Resource Group that has already been set on the agent (or Resource). If not already done so, go to Resources > Groups and set up at least one Resource Group. See [Resources](#) on page 57.



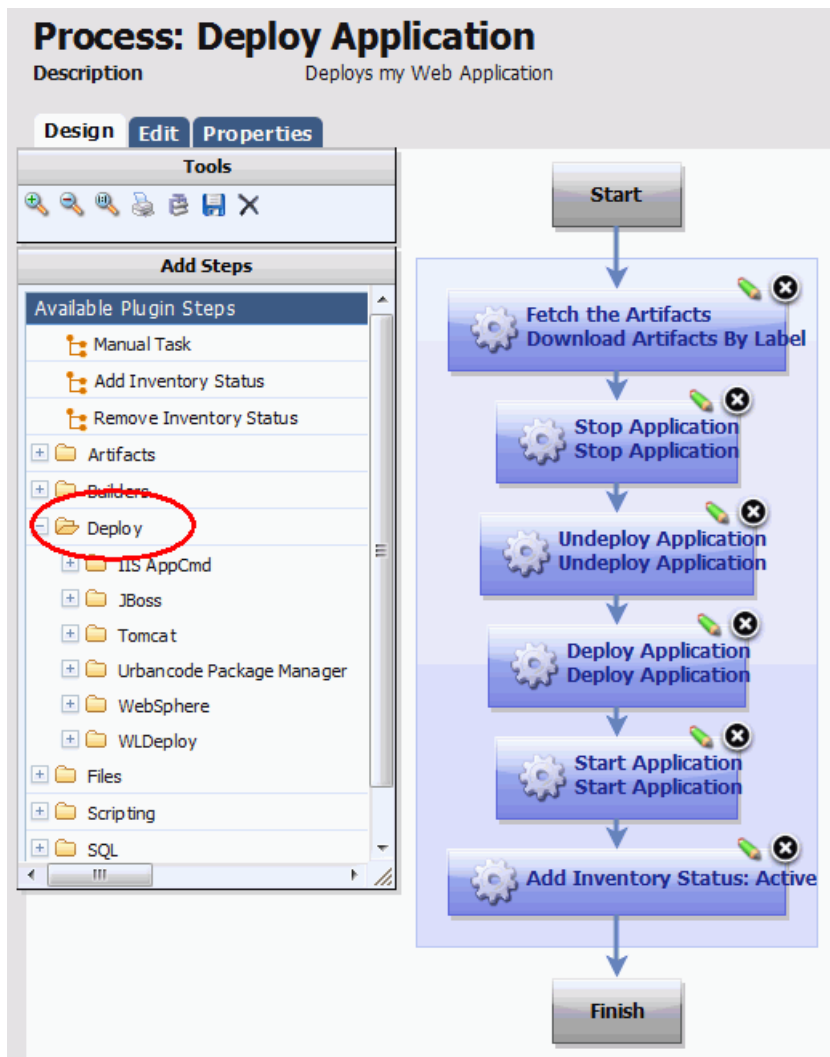
## Deployments

---

uDeploy includes integrations with the most common tools used for web applications. To go beyond a basic deployment, you can configure UrbanDeploy to run tool-specific commands on the target machine. For example, if you are deploying the Application tier to a web server, your Deploy Process could be designed to do the following (all integrations include similar steps):

- Download Artifacts By Label.
- Stop Application. Based on the configuration, this step will stop your application prior to deploying it.
- Undeploy Application. This step is responsible for removing the application from the target machine. This can help ensure a clean install when one is desired.
- Deploy Application. Sends the exact Component Version to the target server and installs the artifacts in the appropriate location.
- Start Application. Once the artifacts have been transferred, UrbanDeploy will automatically restart the application server.
- Add Inventory Status.





**Figure 33: Deploy Application**

Aside from the first and last steps of the Deploy Process, UrbanDeploy allows you to introduce as much automation as is needed for a deployment. For a discussion on the individual integrations, and what each step does, please see the individual integrations listed in the Plugins section.

**Note:** You can set up UrbanDeploy to use the exact same Component Deployment Process for every Application Environment that the Component moves through on its way to Production. For more, see Applications.

In addition to deploying content and interacting with the applications, a Deployment Process can also perform other tasks, including running a SQL script as part of the process, for example when upgrading the database.

Execute SQL Script as Part of Database Update

UrbanDeploy includes an integration that enables you to run a database SQL script when you are deploying a Database Component. You can either use the standard SQL step or, if you are using Oracle, you can use the tool-specific step.

To configure a Deployment Process:

1. Go to the Components, select the Component and then select the Processes tab.
2. Create the Deployment Process.

Name the Process and give it an optional description. The name and description will typically reflect the contents of the Component (e.g., database, application, etc.) as well as the process type (in this case Deploy or Install).

If the Process, and the underlying Component, is to be used by numerous Applications, you can include that information in the description.

**Default working directory.** This is the location that UrbanDeploy will use when executing the steps in the Deploy Process. For most processes, accepting the default value is advisable. The default, which uses a property to determine the directory, enables this process to work in different environments. If you change the default, and add an absolute path, etc., you may not be able to use the same Process as the Component moves through the production pipeline.

**Requires a version.** Check this box if you want the user to enter the version number when running the process. If checked, the version will be passed to the process during runtime.

**Required component role.** This option enables you to restrict who can run this Component Process. The available options are derived from the UrbanDeploy Security System. For example, if you select "Admin" from the drop-down, only users that have been assigned that role in the Security System are able to run this Deployment Process. This can help you enforce who can do what in UrbanDeploy.

3. Once you save the new Process, select it from the table. This will take you to the Process design tool. To set up your process, grab the appropriate steps on the left and drag them onto the canvas.
4. Add the Download Artifacts By Label step. This step is responsible for fetching the artifacts from the UrbanDeploy artifact repository (CodeStation) and should always be the very first step included in a Deployment Process.

**Name.** You can either accept the default name or give a new name.

**Repository URL.** You *must* change this value. You will need to give the URL used to access UrbanDeploy. This value was set during installation and is the one used to log into the server. When changing the URL, ensure that the trailing /vfs is included: this specifies the location of CodeStation, where the Artifacts are being fetched from. For example: `http://urbandeploy.yourcompany.com:8080/vfs`.

**Repository ID.** For most configurations, you should accept the default value, which is a property automatically set by UrbanDeploy. This property tells the system where the Component is stored in the repository.

**Label.** The default property set here references the Label that was applied to the artifacts when they were uploaded into CodeStation. It is advisable to accept the default value.

**Directory offset.** This is the directory UrbanDeploy will use when executing the command. Using the default value (signified by the period) means use the current directory. If you would like to change the directory, for example if a script is looking for files in a specific directory, etc. When changing, the value you give is relative to the working directory. giving "offset/directory" (without the quotes) will switch the working directory to the "directory" folder within the "offset" folder.

**Include and Exclude.** You can tell UrbanDeploy to include or exclude any files stored in CodeStation when the fetch-artifact step is run. The following wild cards are used in addition to specifying a specific file (enter each statement on a new line):

- \*\* Indicates include every directory within the base directory.
- \* Used to include every file. So, if you use \*.zip, the files matching this pattern will be included.
- \*\*/\* Tells UrbanDeploy to retrieve the entire file tree underneath the base directory.

**Allow failure.** Check the box if you would like the step to continue even if a failure is detected.

**Working Directory.** If using the default directory, leave this blank. Other, you will need to specify an absolute path (e.g., C:\path\to\working\directory).

**Use Impersonation.** If the step must run as a different user (as the one UrbanDeploy uses) give the credentials.

5. Add Inventory Status step. This step, which should always come at the end of any Deployment Process, is responsible for updating the Inventory. This will allow UrbanDeploy to track where and when the artifacts have been deployed. Without this step it will be difficult to tell if what is in a desired Application Environment is what you actually intend to be there. Selecting the hard-coded Status of Active will ensure that the Component Deploy Process is correctly identified.
6. Add additional automation to your deployment by inserting the appropriate steps BETWEEN the beginning and ending steps. Please see the Plugin section for the specific steps, if any, you can include in your Component

Deploy Process. By adding additional steps, in the order that they must be executed, you can build a fully automated deployment.



**Note:** You have the option of configuring multiple Components (including versions and processes) before assembling the application. Many users have found that configuring a single Component and then adding it to the Application is the simplest process. This makes it easier to track down errors, etc., when testing the Component Deployment Process. Once the initial component has been successfully deployed throughout the application lifecycle, you can come back and configure the other components and then add them to the application.

If you want to prove out your Deployment Process, you can now configure an Application that uses the Component Deployment Process. Many users have found that configuring a single Component and then adding it to the Application is the simplest process. This makes it easier to track down errors, etc., when testing the Component Deployment Process. Once the initial component has been successfully deployed throughout the application lifecycle, you can come back and configure the other components and then add them to the application. You can always come back and set additional Components at a later time.

## Next Steps

**Resources.** Once you have configured a Component, you will need to ensure that at least one agent has been installed in the target environment and that the agent has been associated with a Resource Group. Go to **Resources > Groups**. If you do not see anything under the "All Resource Groups" folder, you will need to add at least one Resource Group before configuring an Application. See [Resources](#) to continue. If the agent has been associated with a Resource Group, you can configure an Application.

**Add additional automation.** uDeploy integrates with numerous tools used for web applications. These integrations enable you to add tool-specific automation steps to any Component Process. For example, the plug-in system has built-in steps that enable UrbanDeploy to automatically stop, undeploy, deploy, and run servers such as Tomcat, JBoss, and WebSphere. See [Plug-in Integration](#) on page 98

## Reports

---

uDeploy provides deployment- and security-type reports:

- **Deployment reports** contain historical information about deployments. Data can be filtered in a variety of ways and reports can be printed and saved. In addition, you can save search criteria for later use. See [Deployment Reports](#) on page 68
- **Security reports** provide information about user roles and privileges. See [Security Reports](#) on page 77

For information about saving and printing reports, see [Saving and Printing Reports](#) on page 79

The following tables summarize the out-of-the-box reports.

**Table 1: Deployment Reports**

Report	Description
Deployment Detail	Provides information about deployments executed during a user-specified reporting period. Each report row represents a deployment that executed during the reporting period and matched the filter conditions.  See <a href="#">Deployment Detail</a> on page 68.
Deployment Average Duration	Average deployment times for applications executed during a user-specified reporting period.  See <a href="#">Deployment Average Duration</a> on page 73.
Deployment Total Duration	Total deployment times for applications executed during a user-specified reporting period.

Report	Description
	See <a href="#">Deployment Total Duration</a> on page 75
Deployment Count	Provides information about the number of deployments executed during a user-specified reporting period.  See <a href="#">Deployment Count</a> on page 70

**Table 2: Security Reports**

Report	Description
Application Security	Provides information about user roles and privileges defined for uDeploy-managed applications.  See <a href="#">Application Security</a> on page 77
Component Security	Information about user roles and privileges defined for components.  See <a href="#">Component Security</a> on page 77
Environment Security	Information about user roles and privileges defined for environments.  See <a href="#">Environment Security</a> on page 78
Resource Security	Information about user roles and privileges defined for resources.  See <a href="#">Resource Security</a> on page 78

## Deployment Reports

Deployment reports contain historical information about deployments, such as the total number executed and their average duration. Data can be filtered in a variety of ways and reports can be printed and saved. In addition, you can save search criteria for later use. See [Saving and Printing Reports](#) on page 79

### Deployment Detail

The Deployment Detail Report provides information about deployments executed during a user-specified reporting period. Each report row represents a deployment that executed during the reporting period and matched the filter conditions.

Reports can be filtered in a variety of ways (discussed below), and columns selectively hidden. Reports can be saved and printed. See [Saving and Printing Reports](#) on page 79.

When selected, the report runs automatically for the default reporting period--current month--and with all filters set to Any. The default report represents all deployments that ran during the current month.

### Deployment Detail Fields

Initially, all fields are displayed.

Field	Description
Application	Name of the application that executed the deployment.
Environment	Target environment of the deployment.
Date	Date and time when the deployment was executed.
User	Name of the user who performed the deployment.
Status	Final disposition of the deployment. Possible values are:

Field	Description
	<ul style="list-style-type: none"> <li>• Success</li> <li>• Failure</li> <li>• Running</li> <li>• Scheduled</li> <li>• Approval Rejected</li> <li>• Awaiting Approval</li> </ul>
Duration	Amount of time the deployment ran. For a successful deployment, the value represents the amount of time taken to complete successfully. If deployment failed to start, no value is given. If a deployment started but failed to complete, the value represents the amount of time it ran before it failed or was cancelled.
Action	This field provides links to additional information about the deployment. The <a href="#">View Request</a> link displays the <b>Application Process Request</b> pane. See <a href="#">Applications</a> on page 61.

## Running the Deployment Detail Report

To run a report:

1. Use the **Date Range** date-picker to set the report's start- and end-dates.

Value	Effect
Current, Prior Week	Start day is either Sunday or Monday, depending on what is defined in your system.
Current, Prior Month	Start day is first day of the month.
Current, Prior Quarter	Quarters are bound by calendar year.
Current, Prior Year	Current year includes today's date.
Custom	Displays the <b>Custom</b> pop-up which enables you to pick an arbitrary start- and end-date.

The start-time is automatically set to 00:00 (24-hour clock) for the selected date in the reporting period.

The end-time is automatically set to 24:00 for the selected date.

2. Set the report filters.

Filters are set with the properties drop-down list boxes. To set a filter, select it from the corresponding property list box.

Filter	Effect
Application	Only deployments executed by the selected application appear in the report. Default value: Any.
Environment	Only deployments executed by the application selected with the <b>Application</b> list box that also used this environment appear in the report. If the application value is Any, the available value is Any; otherwise, environments defined for the selected application are listed.
User	Only deployments executed by the selected user appear in the report. Default value: Any.

Filter	Effect
Status	Only deployments with the selected status appear in the report. Default value: Any.
Plugin	Only deployments that used the selected plug-in appear in the report. Default value: Any. Note: the Any value also includes deployments that did not use a plug-in.

### 3. Run the report.

Click the **Run** button to apply your filter conditions to the data and produce the report.

By default, the report is sorted by `Application`. You can sort the report on any field by clicking on the column header.

For information about saving and printing reports, see [Saving and Printing Reports](#) on page 79.

## Sample Reports

The following table contains examples of reports that can be produced using the Deployment Detail Report.

Desired Output	Filter / Value
<p><b>Show me:</b> <i>All failed deployments that occurred on July 4th during the previous year.</i></p>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Any</li> <li>• <b>Status:</b> Failure</li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the start- and end-dates to July 4th.</li> </ul>
<p><b>Show me:</b> <i>Deployments for an application that used a specific environment.</i></p>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select the value from the drop-down list box.</li> <li>• <b>Environment:</b> Select the environment from the drop-down list box.</li> </ul> <p>When an application is selected, only environments defined for it are available in the <b>Environment</b> drop-down list box.</p>
<p><b>Show me:</b> <i>Failed deployments that used a specific plug-in yesterday.</i></p>	<ul style="list-style-type: none"> <li>• <b>Status:</b> Failure</li> <li>• <b>Plugin :</b> Select the value from the drop-down list box.</li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the start- and end-dates to the previous day.</li> </ul>
<p><b>Show me:</b> <i>My deployments that used a specific application during the past month.</i></p>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select the value from the drop-down list box.</li> <li>• <b>User:</b> Select your user ID.</li> <li>• <b>Date Range:</b> Select <code>Prior Month</code>.</li> </ul>

## Deployment Count

The Deployment Count Report provides information about the number of deployments executed during a user-specified reporting period. The report provides both a tabular presentation and line graph of the data. Each table row represents an environment used by an applications for the reporting period and interval.

The line graph elements are:

- y-axis represents the number of deployments
- x-axis represents reporting intervals

- plot lines represent environments used by applications

The units along the y-axis are scaled to the number of records reported. The units along the x-axis represent the reporting interval, which can be: months, weeks, or days. Each color-coded plot line represents a single environment used by the deployment during the reporting period.

When selected, the report runs automatically for the default reporting period (current month) and reporting interval (days), and with all filters set to *Any*. The default report provides a count of all deployments that ran during the current month.

### Deployment Count Table Fields

Field	Description
Application	Name of the application that executed the deployment.
Environment	Name of the environment used by the application.
Reporting Interval	The remaining columns display the number of the deployments for the selected reporting interval.

### Running the Deployment Detail Report

To run a report:

1. Set the reporting period.

Use the **Date Range** date-picker to set the report's start- and end-dates. The selected value(s) determines the columns in the tabular report, and the coordinate interval on the graph's x-axis. Default value: *Current Month*.

Value	Effect
Current, Prior Week	Start day is either Sunday or Monday, depending on what is defined in your system. Reporting interval is set to days.
Current, Prior Month	Start day is first day of the month. Reporting interval is set to days.
Current, Prior Quarter	Quarters are bound by calendar year. Reporting interval is set to weeks.
Current, Prior Year	Reporting interval is set months.
Custom	Displays the <b>Custom</b> pop-up which enables you to pick an arbitrary start- and end-date.

The start-time is automatically set to 00:00 (24-hour clock) for the selected date in the reporting period.

The end-time is automatically set to 24:00 for the selected date.

2. Set the report filters.

Filters are set with the properties drop-down list boxes. To set a filter, select it from the corresponding property list box.

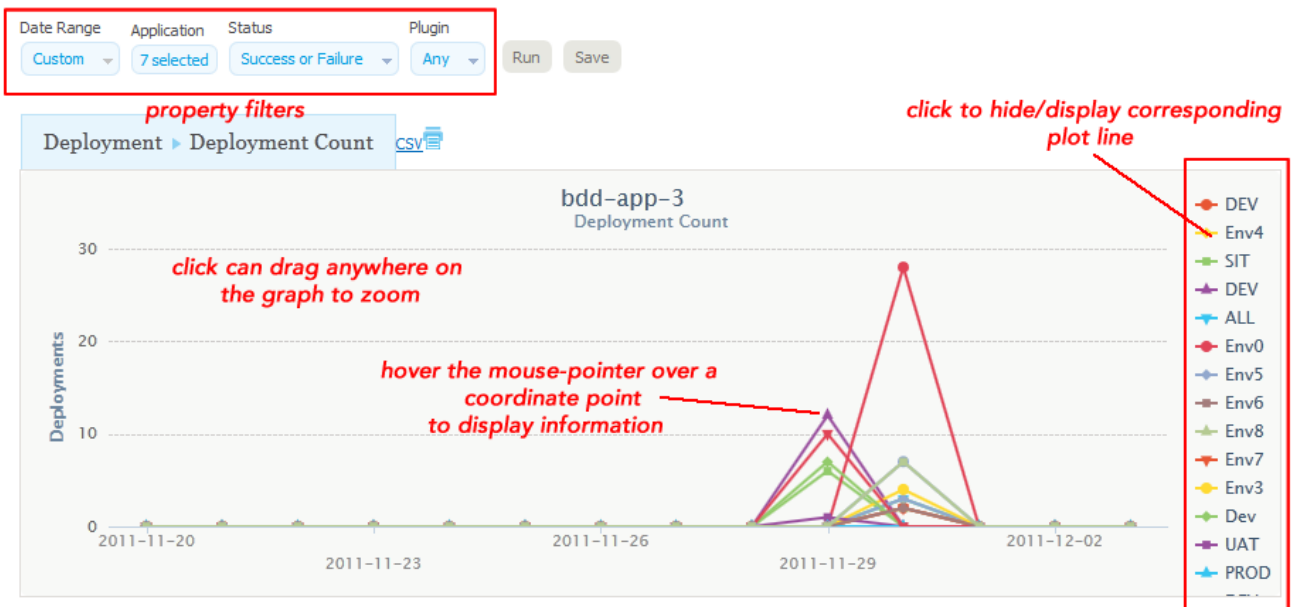
Filter	Effect
Application	<p>Only deployments executed by the selected application(s) appear in the report. .</p> <p>To select applications:</p> <ol style="list-style-type: none"> <li>1. Click the <b>Application</b> button.</li> <li>2. To include an application in the report, click the corresponding check box.</li> </ol>

Filter	Effect
	<p>If a large number of applications are listed, type the first few letters of the application's name in the text box to scroll the list. Multiple applications can be selected.</p> <p>3. Click <b>OK</b>.</p>
Status	Only deployments with the selected status appear in the report. Default value: <code>Success</code> or <code>Failure</code> , which means all deployments.
Plugin	Only deployments that used the selected plug-in appear in the report. Default value: <code>Any</code> . Note: the <code>Any</code> value also includes deployments that did not use a plug-in.

3. Run the report.

Click the **Run** button to apply your filter conditions to the data and produce the report.

A tabular report and line graph are produced.



**Figure 34: Deployment Count Graph**

Each environment used by a reporting application is represented by an individual plot line and table row. You can hide a plot line by clicking the corresponding item in the graph legend. To see information about a graph coordinate, hover the mouse over the graph point.

You can zoom a graph area by dragging the mouse over the area.

For information about saving and printing reports, see [Saving and Printing Reports](#) on page 79.

**Sample Reports**

The following table contains examples of reports that can be produced using the Deployment Count Report.

Desired Output	Filter / Value
<p><b>Show me:</b></p> <p><i>The number of successful deployments for two specific</i></p>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select both applications from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> <code>Success</code></li> <li>• <b>Plugin:</b> Select the plug-in from the drop-down list box.</li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the ten-day range.</li> </ul>



Desired Output	Filter / Value
<i>applications during the past ten days that used a particular plug-in.</i>	.
<b>Show me:</b> <i>The number of failed deployments for a given application during the past month</i>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select the value from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> Failure</li> <li>• <b>Date Range:</b> Select <code>Prior Month</code>.</li> </ul>
<b>Show me:</b> <i>The number of failed deployments that used a specific plug-in yesterday.</i>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select the applications from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> Failure</li> <li>• <b>Plugin:</b> Select the value from the drop-down list box.</li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to select the previous day.</li> </ul>

### Deployment Average Duration

The Deployment Average Duration Report provides average deployment times for applications executed during a user-specified reporting period. The report provides both a tabular presentation and line graph of the data. Each table row represents an environment used by an application for the reporting period and interval.

The line graph elements are:

- y-axis represents deployment duration average times
- x-axis represents reporting intervals
- plot lines represent environments used by the applications

The units along the y-axis are scaled to the number of records reported. The units along the x-axis represent the reporting interval, which can be: months, weeks, or days. Each color-coded plot line represents a single environment used by the deployment during the reporting period.

When selected, the report runs automatically for the default reporting period (current month) and reporting interval (days), and with all filters set to `Any`. The default report provides average deployment times for all deployments that ran during the current month.

### Deployment Average Duration Fields

Field	Description
Application	Name of the application that executed the deployment.
Environment	Name of the environment used by the application.
Reporting Interval	The remaining columns display the average deployment times for the reporting interval.

### Running the Deployment Average Duration Report

To run a report:

1. Set the reporting period.

Use the **Date Range** date-picker to set the report's start- and end-dates. The selected value(s) determines the columns in the tabular report, and the coordinate interval on the graph's x-axis. Default value: `Current Month`.

Value	Effect
Current, Prior Week	Start day is either Sunday or Monday, depending on what is defined in your system. Reporting interval is set to days.
Current, Prior Month	Start day is first day of the month. Reporting interval is set to days.
Current, Prior Quarter	Quarters are bound by calendar year. Reporting interval is set to weeks.
Current, Prior Year	Reporting interval is set months.
Custom	Displays the <b>Custom</b> pop-up which enables you to pick an arbitrary start-and end-date.

The start-time is automatically set to 00:00 (24-hour clock) for the selected date in the reporting period.

The end-time is automatically set to 24:00 for the selected date.

## 2. Set the report filters.

Filters are set with the properties drop-down list boxes. To set a filter, select it from the corresponding property list box.

Filter	Effect
Application	<p>Only deployments executed by the selected application(s) appear in the report.</p> <p>To select applications:</p> <ol style="list-style-type: none"> <li>1. Click the <b>Application</b> button.</li> <li>2. To include an application in the report, click the corresponding check box.</li> </ol> <p>If a large number of applications are listed, type the first few letters of the application's name in the text box to scroll the list. Multiple applications can be selected.</p> <ol style="list-style-type: none"> <li>3. Click <b>OK</b>.</li> </ol>
Status	Only deployments with the selected status appear in the report. Default value: <code>Success</code> or <code>Failure</code> , which means all deployments.
Plugin	Only deployments that used the selected plug-in appear in the report. Default value: <code>Any</code> . Note: the <code>Any</code> value also includes deployments that did not use a plug-in.

## 3. Run the report.

Click the **Run** button to apply your filter conditions to the data and produce the report.

A tabular report and line graph are produced. Each environment used by a reporting application is represented by an individual plot line and table row. You can hide a plot line by clicking the corresponding item in the graph legend. To see information about a graph coordinate, hover the mouse over the graph point.

You can zoom a graph area by dragging the mouse over the area.

For information about saving and printing reports, see [Saving and Printing Reports](#) on page 79.

## Sample Reports

The following table contains examples of reports that can be produced using the Deployment Average Duration Report.

Desired Output	Filter / Value
<p><b>Show me:</b></p> <p><i>Average durations for two specific applications during the past ten days that used a particular plug-in.</i></p>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select both applications from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> Success or Failure</li> <li>• <b>Plugin:</b> Select the plug-in from the drop-down list box.</li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the ten-day range.</li> </ul>
<p><b>Show me:</b></p> <p><i>Average durations for successful deployments for a given application during the past six months.</i></p>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select the application from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> Success</li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the range to the previous six months.</li> </ul>

### Deployment Total Duration

The Deployment Total Duration Report provides total deployment times for applications executed during a user-specified reporting period. The report provides both a tabular presentation and line graph of the data. Each table row represents an environment used by one of the selected applications for the reporting period and interval.

The line graph elements are:

- y-axis represents deployment duration times
- x-axis represents reporting intervals
- plot lines represent environments used by the applications

The units along the y-axis are scaled to the number of records reported. The units along the x-axis represent the reporting interval, which can be: months, weeks, or days. Each color-coded plot line represents a single environment used by an application during the reporting period.

When selected, the report runs automatically for the default reporting period (current month) and reporting interval (days), and with all filters set to Any. The default report provides total deployment times for all deployments that ran during the current month.

### Deployment Total Duration Fields

Field	Description
Application	Name of the application that executed the deployment.
Environment	Name of the environment used by the application.
Reporting Interval	The remaining columns display the total deployment times for the reporting interval.

### Running the Deployment Total Duration Report

To run a report:

1. Set the reporting period.

Use the **Date Range** date-picker to set the report's start- and end-dates. The selected value(s) determines the columns in the tabular report, and the coordinate interval on the graph's x-axis. Default value: Current Month.

Value	Effect
Current, Prior Week	Start day is either Sunday or Monday, depending on what is defined in your system. Reporting interval is set to days.

Value	Effect
Current, Prior Month	Start day is first day of the month. Reporting interval is set to days.
Current, Prior Quarter	Quarters are bound by calendar year. Reporting interval is set to weeks.
Current, Prior Year	Reporting interval is set months.
Custom	Displays the <b>Custom</b> pop-up which enables you to pick an arbitrary start- and end-date.

The start-time is automatically set to 00:00 (24-hour clock) for the selected date in the reporting period.

The end-time is automatically set to 24:00 for the selected date.

## 2. Set the report filters.

Filters are set with the properties drop-down list boxes. To set a filter, select it from the corresponding property list box.

Filter	Effect
Application	<p>Only deployments executed by the selected application(s) appear in the report.</p> <p>To select applications:</p> <ol style="list-style-type: none"> <li>1. Click the <b>Application</b> button.</li> <li>2. To include an application in the report, click the corresponding check box.</li> </ol> <p>If a large number of applications are listed, type the first few letters of the application's name in the text box to scroll the list. Multiple applications can be selected.</p> <ol style="list-style-type: none"> <li>3. Click <b>OK</b>.</li> </ol>
Status	Only deployments with the selected status appear in the report. Default value: <code>Success</code> or <code>Failure</code> , which means all deployments.
Plugin	Only deployments that used the selected plug-in appear in the report. Default value: <code>Any</code> . Note: the <code>Any</code> value also includes deployments that did not use a plug-in.

## 3. Run the report.

Click the **Run** button to apply your filter conditions to the data and produce the report.

A tabular report and line graph are produced. Each environment used by a reporting application is represented by an individual plot line and table row. You can hide a plot line by clicking the corresponding item in the graph legend. To see information about a graph coordinate, hover the mouse over the graph point.

You can zoom a graph area by dragging the mouse over the area.

For information about saving and printing reports, see [Saving and Printing Reports](#) on page 79.

## Sample Reports

The following table contains examples of reports that can be produced using the Deployment Total Duration Report.

Desired Output	Filter / Value
<b>Show me:</b>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select both applications from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> <code>Success</code> or <code>Failure</code></li> <li>• <b>Plugin:</b> Select the plug-in from the drop-down list box.</li> </ul>

Desired Output	Filter / Value
<i>Total duration times for two specific applications during the past ten days that used a particular plug-in.</i>	<ul style="list-style-type: none"> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the ten-day range.</li> <li>•</li> </ul>
<p><b>Show me:</b></p> <p><i>Total duration times for successful deployments for a given application during the past six months.</i></p>	<ul style="list-style-type: none"> <li>• <b>Application:</b> Select the application from the <b>Applications</b> dialog.</li> <li>• <b>Status:</b> Success</li> <li>• <b>Time Unit:</b> Months</li> <li>• <b>Date Range:</b> Use the <b>Custom</b> pop-up to set the six-month range.</li> <li>•</li> </ul>

## Security Reports

Security reports provide information about user roles and privileges defined with the uDeploy security system.

### Application Security

The Application Security Report provides information about user roles and privileges defined for uDeploy-managed applications. Each report row represents an individual application. When selected, the report runs automatically for all applications.

#### Application Security Fields

Field	Description
Application	Name of the application.
Run Component Processes	Users who have component process execution privileges. For information about component processes, see <a href="#">Creating Components</a> on page 44.
Execute	Users who have application execution privileges. For information about applications, see <a href="#">Applications</a> on page 61.
Security	Users who can define privileges for other users. For information about security, see <a href="#">Security</a> on page 90.
Read	Users who can review information about the application but not change it.
Write	Users who can access and edit the application.

The report is sorted by *Application*. You can change the sort order by clicking on the column header.

For information about saving and printing reports, see [Saving and Printing Reports](#) on page 79.

### Component Security

The Component Security Report provides information about user roles and privileges defined for components. Each report row represents an individual component. When selected, the report runs automatically for all components.

#### Component Security Fields

Fields are:

Field	Description
Component	Name of the component.

Field	Description
Execute	Users who have component process execution privileges. For information about component processes, see <a href="#">Creating Components</a> on page 44.
Security	Users who can define privileges for other users. For information about security, see <a href="#">Security</a> on page 90.
Read	Users who can review information about the component but not change it.
Write	Users who can access and edit the component.

The report is sorted by Component. You can change the sort order by clicking on the column header.

For information about saving and printing reports, see [Saving and Printing Reports](#) on page 79.

### Environment Security

The Environment Security Report provides information about user roles and privileges defined for environments. Each report row represents an individual environment. When selected, the report runs automatically for all environments.

#### Environment Security Fields

Field	Description
Application	Name of the application.
Environment	Name of the environment.
Execute	Users who have execution privileges for the environment. For information about environments, see <a href="#">Applications</a> on page 61.
Security	Users who can define privileges for other users. For information about security, see <a href="#">Security</a> on page 90.
Read	Users who can review information about the environment (but not change it).
Write	Users who can access and edit the environment.

The report can be sorted by by Application or Environment. By default, it is sorted by Application. You can change the sort order by clicking on the column header.

For information about saving and printing reports, see [Saving and Printing Reports](#) on page 79.

### Resource Security

The Resource Security Report provides information about user roles and privileges defined for resources. Each report row represents an individual resource. When selected, the report runs automatically for all resources.

#### Resource Security Fields

Fields are:

Field	Description
Resource	Name of the resource.

Field	Description
Execute	Users who have execution privileges for the resource. For information about resources, see <a href="#">Resources</a> on page 57.
Security	Users who can define privileges for other users. For information about security, see <a href="#">Security</a> on page 90.
Read	Users who can review information about the resource but not change it.
Write	Users who can access and edit the resource.

The report is sorted by **Resource**. You can change the sort order by clicking on the column header.

For information about saving and printing reports, see [Saving and Printing Reports](#) on page 79.

## Saving and Printing Reports

You can print and save report data for all report types. In addition, you can save filter and sort order information for deployment-type reports.


### Saving Report Data

uDeploy saves report data in CSV files (comma separated value).

#### To save report data:

1. Set the filters (if any) and run the report.
2. Click the **CSV** button.
3. Use the **Opening File** dialog.

You can save the data to file, or open the data with an application associated with CSV-type files on your system.

 **Note:** Sort-order and hidden/visible column information is not preserved in the CSV file.

### Saving Report Filters

You can save filter and sort-order settings for deployment reports. Saved reports can be retrieved with the **My Reports** menu on the **Reports** pane.

#### To save a report:

1. Set the filter conditions.
2. Define the reporting period.
3. Run the report.
4. Optionally, set the sort order.

You can change the sort order for any column by clicking the column header.

5. Optionally, change column visibility.

Click the **Edit** button to display the Select Columns dialog. By default, all columns are selected to appear in a report. To hide a column, click the corresponding check box.

6. Click the **Save** button.

The **Save Current Filters** dialog is displayed.

7. Enter a name for the file, and save your work.

To run your report, click the report name in the **My Reports** menu.

To delete your report, click the **Delete** button.

## Printing Reports

### To print a report:

1. Set the filter conditions.
2. Define the reporting period.
3. Run the report.
4. Optionally, set the sort order.

Your changes are reflected in the printed report.

5. Optionally, change column visibility.

By default, all columns are selected to appear in the printed report. Hidden columns will not appear in the output.

6. Click the **Print** button to print your report.

## Schedule Deployments

---

uDeploy has a built-in deployment scheduling system for setting regular deployments, or even black-out dates, for your Deployments. Deployments for an individual Application are scheduled on a per-environment basis, set when you run a deployment of a Snapshot or Deployment Process. Black-out dates are set within the individual Environments.

### Creating a Schedule

To set up a Scheduled Deployment, go to Application > Environment > Run Process. If you are scheduling a Snapshot deployment, you would go to Application > Snapshots > Run Process instead. Regardless of the type of deployment you are scheduling, configuration is the same.

After you check the Schedule Deployment box, uDeploy will prompt you to give the date and time you want the deployment to run. The Make Recurring setting will deploy the Application on a regular schedule. For example, if you are practicing Continuous Delivery, the Daily option will deploy the Application to the target Environment every day.

Once you have scheduled the deployment, it will be added to the Calendar. There, if you click on the Scheduled Deployment, you can edit, delete, or investigate the deployment.

### Set Blackouts

Blackouts are set per-environment, per-application. Once set, no deployments (nor Snapshots) can be scheduled to occur in that Environment. Any previously scheduled deployments to the Environment will fail if they fall within the blackout date you set. To set up a blackout, go to Application > Environments > Calendar > Add Blackout. If you need to set blackouts for more than one Environment, you must do this for each individual one. uDeploy will prompt you to give the dates and times for the blackout.



---

# Part IV

---

## Administration

---

Topics:

- [Installation](#)
  - [Security](#)
-

## Installation

---

An installation of uDeploy consists of the uDeploy server (with a supporting database), and at least one agent. Typically, the server, database, and agents are installed on separate machines. For a simple evaluation they can all be installed on the same machine. In addition, Java must be installed on all agent and server machines.



**Note:** For evaluation purposes, the supplied Derby database should be adequate and can be installed on the machine where the server is located. If you are installing uDeploy in a production environment, UrbanCode recommends the use one of the supported databases--Oracle Database (all versions), SQL Server, or MySQL.

### Installation Steps

1. Review the system requirements. See [System Requirements](#) on page 82.
2. Ensure that Java is installed on both the server and agent machines. Server and agent machines require Java JRE 5 or greater. Set the JAVA\_HOME environment variable to point to the directory you intend to use. You can also use the JDK.
3. Download both the uDeploy server and agent installation files. If you are installing an evaluation version, the license is included with the downloaded files.
4. If you are not installing an evaluation version, install one of the supported databases. The database should be installed before the server and on a separate machine. See [Database Installation](#) on page 84
5. Complete database installation by configuring the appropriate JDBC driver (typically supplied by the database vendor).
6. Create an empty database for uDeploy and at least one dedicated user account.
7. Install the server. See [Server Installation](#) on page 86.
8. Finally, install at least one agent. See [Agent Installation](#) on page 88.

## System Requirements

uDeploy will run on Windows and UNIX-based systems. While the minimum requirements provided below are sufficient for an evaluation, you will want server-class machines for production deployments.

### Server Minimum Installation Requirements

- Windows: Windows 2000 Server (SP4) or later.
- Processor: Single core, 1.5 GHz or better.
- Disk Space: 300 MB or more.
- Memory: 2 GB, with 256 MB available to uDeploy.
- Java version: JRE 5 or greater.

### Recommended Server Installation

- **Two server-class machines**

UrbanCode 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 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 uDeploy's artifact repository (CodeStation), the more storage needed.



**Note:** CodeStation is installed when the uDeploy server is installed.

For production environments, use the following guidelines to determine storage requirements:

- 10-20 GB of database storage should be sufficient for most environments.
- To calculate CodeStation storage requirements:

average build artifact size \* number of builds per day \* average number of days before cleanup

For further assistance in determining storage requirements, contact UrbanCode support.

- **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. For a simple evaluation, the agent can be installed on the same physical machine as the server. In production environments, agents should be installed on separate machines.

### 32- and 64-bit JVM Support

The uDeploy server must use the 32-bit JDK for the Windows 2003 64-bit server; the 64-bit JDK can be used for agents. Because uDeploy does not require a multi-gigabyte heap, there is little advantage to using a 64-bit JVM. For 64-bit Windows installations, uDeploy uses a 32-bit JVM; for other 64-bit platforms, uDeploy uses a 64-bit JVM, as the following table illustrates:

Operating System	JVM 32-bit	JVM 64-bit
Windows 32-bit	yes	NA
Windows 64-bit	yes	no
Non-Windows 32-bit	yes	NA
Non-Windows 64-bit	yes	yes

### Performance Recommendations

Since the uDeploy agent performs most of the processing, optimal agent configuration is important. Except when evaluating uDeploy, an agent should not be installed on the same machine where the server is located.

By following these recommendations, you should avoid most performance-related issues:

- **Install the server as a dedicated user account.** The server should be installed as a dedicated system account whenever possible. However, uDeploy runs well as a root user (or local system user on Windows), and running this way is the easiest method to avoid permission errors.
- **Install the agent as dedicated system account.** Ideally, the account used should be dedicated to uDeploy. Because uDeploy agents are remote command-execution engines, it is best to create a user just for the agent and grant it only the appropriate privileges.
- **Do not install an agent on the uDeploy server machine.** Because the agent is resource intensive, installing one on the server machine will degrade server performance whenever a large deployment runs.
- **Install one agent per machine.** Several agents on the same machine can result in significant performance reduction, especially when they are running at the same time.

## Download UrbanDeploy

The installation package is available from the UrbanCode support portal--Supportal. If you are evaluating uDeploy, the Supportal account where you download uDeploy also enables you to create support tickets.

1. Navigate to the UrbanCode Support Portal `support.urbancode.com/tasks/login/LoginTasks/login`.

If you do not have an account, please create one.



**Note:** You must have a license in order to download the product. For an evaluation license, go to `urbancode.com/html/products/deploy/default.html`.

2. Click the **Products** tab and select the uDeploy version you want to download.
3. Select the appropriate package for your environment for both the server and agent. The contents of the zip and tar packages are the same.

uDeploy enables you to install agents on any supported platform, regardless of the operating system where the server is installed.

4. Download the license. If you do not see a license, ensure that you are the Supportal account administrator. Licenses are not available to all Supportal users.

## Database Installation

Currently, uDeploy supports Derby, Oracle, SQL Server, and MySQL.

### Installing Oracle

Before installing the uDeploy server, install an Oracle database. If you are evaluating uDeploy, you can install the database on the same machine where the uDeploy server will be installed.

When you install uDeploy, you will need the Oracle connection information, and a user account with table creation privileges.

#### uDeploy supports the following editions:

- Oracle Database Enterprise
- Oracle Database Standard
- Oracle Database Standard One
- Oracle Database Express

Version 10g or later is supported for each edition.

#### To install the database

1. Obtain the Oracle JDBC driver. The JDBC jar file is included among the Oracle installation files. The driver is unique to the edition you are using.
2. Copy the JDBC jar file to `uDeploy_installer_directory\lib\ext`.
3. Begin server installation, see [Server Installation](#) on page 86. When you are prompted for the database type, enter `oracle`.
4. Provide the JDBC driver class uDeploy will use to connect to the database.

The default value is `oracle.jdbc.driver.OracleDriver`.

5. Provide the JDBC connection string. The format depends on the JDBC driver.

Typically, it is similar to:

```
jdbc:oracle:thin:@[DB_URL]:[DB_PORT]
```

For example:

```
jdbc:oracle:thin:@localhost:1521.
```

6. Finish by entering the database user name and password.

## Installing MySQL

Before installing the uDeploy server, install MySQL. If you are evaluating uDeploy, you can install the database on the same machine where the uDeploy server will be installed.

When you install uDeploy, you will need the MySQL connection information, and a user account with table creation privileges.

### To install the database

1. Create a database:

```
CREATE DATABASE urbandeploy;

GRANT ALL ON urbandeploy * TO 'urbandeploy'@'%'
IDENTIFIED BY 'password' WITH GRANT OPTION;
```

2. Obtain the MySQL JDBC driver. The JDBC jar file is included among the installation files. The driver is unique to the edition you are using.
3. Copy the JDBC jar file to *uDeploy\_installer\_directory*\lib\ext.
4. Begin server installation, see [Server Installation](#) on page 86. When you are prompted for the database type, enter `mysql`.
5. Provide the JDBC driver class uDeploy will use to connect to the database.

The default value is `com.mysql.Driver`.

6. Next, provide the JDBC connection string.

Typically, it is similar to:

```
jdbc:mysql:[DB_URL]:[DB_PORT]:[DB_NAME]
```

For example:

```
jdbc:mysql://localhost:3306/urbandeploy.
```

7. Finish by entering the database user name and password.

## Installing Microsoft SQL Server

Before installing the uDeploy server, install a SQL Server database. If you are evaluating uDeploy, you can install the database on the same machine where the uDeploy server will be installed.

When you install uDeploy, you will need the SQL Server connection information, and a user account with table creation privileges.

Before installing the uDeploy server, install an SQL Server database. If you are evaluating uDeploy, you can install the database on the same machine where the uDeploy server will be installed:

```
CREATE DATABASE udeploy;

USE udeploy;

CREATE LOGIN udeploy WITH PASSWORD = 'password';

CREATE USER udeploy FOR LOGIN udeploy WITH DEFAULT_SCHEMA = udeploy;

CREATE SCHEMA udeploy AUTHORIZATION udeploy;

GRANT ALL TO udeploy;
```

1. Obtain the SQL Server JDBC driver. The JDBC jar file is included among the installation files.
2. Copy the JDBC jar file to *uDeploy\_installer\_directory*\lib\ext.
3. Begin server installation, see [Server Installation](#) on page 86. When you are prompted for the database type, enter `sqlserver`.

4. Provide the JDBC driver class uDeploy will use to connect to the database.

The default value is `com.microsoft.sqlserver.jdbc.SQLServerDriver`.

5. Next, provide the JDBC connection string. The format depends on the JDBC driver.

Typically, it is similar to:

```
jdbc:sqlserver://[DB_URL]:[DB_PORT];databaseName=[DB_NAME]
```

For example:

```
jdbc:sqlserver://localhost:1433;databaseName=udeploy.
```

6. Finish by entering the database user name and password.

## Server Installation

The server provides services such as the user interface used to configure application deployments, the work flow engine, the security service, and the artifact repository, among others



**Note:** If you are installing the server in a production environment, install and configure the database you intend to use before installing the server. See [Database Installation](#) on page 84.

### Windows Server Installation

1. Download and unpack the installation files to the *installer\_directory*.
2. From the *installer\_directory*, run `install-server.bat`.



**Note:** Depending on your Windows version, you might need to run the batch file as the administrator.

The uDeploy Installer is displayed and prompts you to provide the following information:

3. **Enter the directory where the uDeploy Server will be installed.**

Enter the directory where you want the server located. If the directory does not exist, enter Y to instruct the Installer to create it for you. The default value is Y.



**Note:** Whenever the uDeploy Installer suggests a default value, you can press ENTER to accept the value.

4. **Please enter the home directory of the JRE/JDK used to run the server.**

If Java has been previously installed, uDeploy will suggest the Java location as the default value. To accept the default value, press ENTER, otherwise override the default value and enter the correct path.

5. **Enter the IP address on which the Web UI should listen.** UrbanCode suggests accepting the default value `all available to this machine`.
6. **Do you want the Web UI to always use secure connections using SSL?**

Default value is Y.

If you use SSL, turn it on for agents too, or the agents will not be able to connect to the server. This also applies if using mutual authentication. If you change the port numbers for agent communication, you need to provide the port numbers when installing the agents.

7. **Enter the port where uDeploy should listen for secure HTTPS requests.**

The default value is 8443.

8. **Enter the port on which the uDeploy should redirect unsecured HTTP requests.**

The default value is 8080.

9. **Enter the URL for external access to the web UI.**

10. **Enter the port to use for agent communication.**

The default value is 7918.

11. **Do you want the Server and Agent communication to require mutual authentication?**

If you select Y, a manual key must be exchanged between the server and each agent. The default value is N.

**12. Enter the database type UrbanDeploy should use.**

The default value is the supplied database Derby. The other supported databases are: `mysql`, `oracle`, and `sqlserver`.

If you enter a value other than `derby`, the uDeploy Installer will prompt you for connection information, which was defined when you installed the database. See [Database Installation](#) on page 84.

**13. Enter the database user name..** The default value is `urbandeploy`.

Enter the user name you created during database installation.

**14. Enter the database password..** The default value is `password`.

**15. Do you want to install the Server as Windows service?.** The default value is N.



**Note:** When installed as a service, uDeploy only captures the value for the PATH variable. Values captured during installation will always be used, even if you make changes later. For recent Windows versions, you will need to execute the command as Administrator.

## UNIX/LINUX Installation

**1.** Download and unpack the installation files to the *installer\_directory*.



**Note:** If you are installing uDeploy on Solaris, UrbanCode recommends the Korn shell (ksh).

**2.** From the *installer\_directory* run `install-server.sh`.

The uDeploy Installer is displayed and prompts you to provide the following information:

**3. Enter the directory where the uDeploy Server will be installed.**

If the directory does not exist, enter Y to instruct the Installer to create it for you. The default value is Y.



**Note:** Whenever the uDeploy Installer suggests a default value, you can press ENTER to accept the value.

**4. Please enter the home directory of the JRE/JDK used to run the server.**

If Java has been previously installed, uDeploy will suggest the Java location as the default value. To accept the default value, press ENTER, otherwise override the default value and enter the correct path.

**5. Enter the IP address on which the Web UI should listen.** UrbanCode suggests accepting the default value `all available to this machine`.

**6. Do you want the Web UI to always use secure connections using SSL?**

Default value is Y.

If you use SSL, turn it on for agents too, or the agents will not be able to connect to the server. This also applies if using mutual authentication. If you change the port numbers for agent communication, you need to provide the port numbers when installing the agents.

**7. Enter the port where uDeploy should listen for secure HTTPS requests.**

The default value is `8443`.

**8. Enter the port on which the uDeploy should redirect unsecured HTTP requests.**

The default value is `8080`.

**9. Enter the URL for external access to the web UI.**

**10. Enter the port to use for agent communication.**

The default value is `7918`.

**11. Do you want the Server and Agent communication to require mutual authentication?**

If you select Y, a manual key must be exchanged between the server and each agent. The default value is N.

**12. Enter the database type UrbanDeploy should use.**

The default value is the supplied database `Derby`. The other supported databases are: `mysql`, `oracle`, and `sqlserver`.

If you enter a value other than `derby`, the uDeploy Installer will prompt you for connection information, which was defined when you installed the database. See [Database Installation](#) on page 84.

**13. Enter the database user name..** The default value is `urbandeploy`.

Enter the user name you created when you installed the database.

**14. Enter the database password..** The default value is `password`.

## Agent Installation

For production environments, UrbanCode recommends creating a user account dedicated to running the agent on the machine where the agent is installed.

For simple evaluations, the administrative user can run the agent on the machine where the server is located. But if you plan to run deployments on several machines, a separate agent should be installed on each machine. If, for example, your testing environment consists of three machines, install an agent on each one. Follow the same procedure for each environment the application uses.

Each agent needs the appropriate rights to communicate with the uDeploy server.


At a minimum, each agent should have permission to:

- **Create a cache.** By default, the cache is located in the home directory of the user running the agent. The cache can be moved or disabled.
- **Open a TCP connection.** The agent uses a TCP connection to communicate with the server's JMS port.
- **Open a HTTP(S) connection.** The agent must be able to connect to the uDeploy user interface in order to download artifacts from the CodeStation repository.
- **Access the file system.** Many agents need read/write permissions to items on the file system.

### Installing an Agent

After downloading and expanding the installation package, open the *installer\_directory*.


From the *installer\_directory* run `install-server.bat` (Windows) or `install-server.sh` (UNIX-LINUX).

 **Note:** If you are installing Windows, you might need to run the batch file as the administrator.

The uDeploy Installer is displayed and prompts you to provide the following information:

**1. Enter the directory where agent should be installed..** For example: `C:\Program Files\urban-deploy\agent` (Windows) or `/opt/urban-deploy/agent` (UNIX).

If the directory does not exist, enter `Y` to instruct the Installer to create it for you. The default value is `Y`.

 **Note:** Whenever the uDeploy Installer suggests a default value, you can press `ENTER` to accept the value.

**2. Please enter the home directory of the JRE/JDK used to run the agent.**

If Java has been previously installed, uDeploy will suggest the Java location as the default value. To accept the default value, press `ENTER`, otherwise override the default value and enter the correct path.

**3. Will the agent connect to a agent relay instead of directly to the server?**

The default value is `N`.

**4. Enter the host name or address of the server the agent will connect to.** The default value is `localhost`.

**5. Enter the agent communication port for the server.**

The default value is `7918`.



## 6. Does the server agent communication use mutual authentication with SSL?

Default value is `Y`.

If you use SSL, turn it on for server too or the agent will not be able to connect to the server. This also applies if using mutual authentication. If you change the port numbers for agent communication, you need to provide them when installing the agents.

**7. Enter the name for this Agent.** Enter a unique name; the name will be used by uDeploy to identify this agent.

**8. Do you want to install the Agent as Windows service?** (Windows only).

The default value is `N`. When installed as a service, uDeploy only captures the value for the `PATH` variable.

Values captured during installation will always be used, even if you make changes later. For recent Windows versions, you will need to execute the command as Administrator.

## Running uDeploy

Both UNIX-based and Windows installation require the uDeploy server and at least one agent. Before you continuing, ensure that you have the correct JVM/JDK for the server. If you are using a Oracle or MySQL database, make sure you have installed and configured the appropriate driver, see [Database Installation](#) on page 84.

### Running the Server

1. Navigate to the `server_installation_directory\bin` directory
2. Run the `run_server.cmd` batch file (Windows), or `start_server.cmd` (UNIX/LINUX).

### Running an Agent

After the server has successfully started:

1. Navigate to the `agent_installation_directory\bin` directory
2. Run the `run_udagent.cmd` batch file (Windows), or `start_udagent.cmd` (UNIX/LINUX).
3. Once the installer is done, start the agent. Go to the UrbanDeploy agent directory created during installation. For example, `C:\Program Files\urban-deploy\agent`. (Windows) or `/opt/urban-deploy/agent` (UNIX-like system). Enter the `bin` directory. Run: `run_udagent.cmd` (Windows) or `"udagent run"` (UNIX-like systems, without the quotes).
4. When the agent has finished starting up, go to the UrbanDeploy UI and select the Resources tab. You should see the agent in the list. If the agent is not visible, ensure that you used the correct connection ports; if using SSL, ensure it is turned on for both the server and the agent; that there is no firewall blocking communication; and that the license is activated. If the agent still can't establish a connection to the server, please contact support.
5. To install another agent, repeat the previous steps. Note that you can use the same agent installer for both Windows and UNIX-like systems.

### Accessing uDeploy

1. Open a web browser and navigate to the external URL you entered during installation.
2. Log onto the server by using the default credentials.

**User name:** `admin admin`

**Password:** `admin admin`


You can change these later by using the **Settings** pane, see [Database Installation](#) on page 84

3. Activate license. A license is required for the agents to connect to the server. Without a license, UrbanDeploy will be unable to run deployments, etc. If not already done so, go to Supportal and retrieve the license. Go to the Setting tab and either upload or past the license to activate it.
4. To install an agent, see Agent Installation.

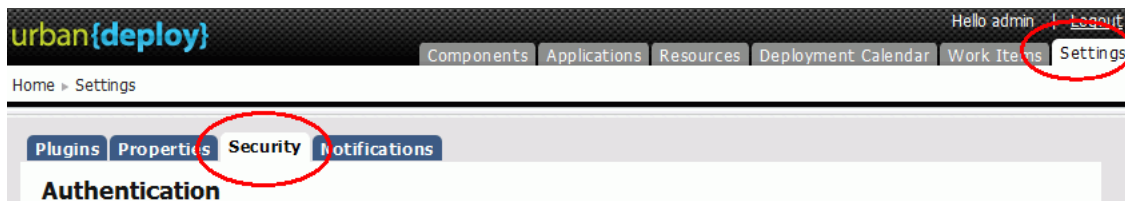
## Security

In uDeploy, you have detailed control over what users can see and do. The Security System maps to your organizational structure by teams, activities, etc. For example, you can set up uDeploy so that team members only see the Applications or Components they work with; or if a manager just needs to approve a deployment, etc., you can set up uDeploy so all they see are the assigned Work Items.

uDeploy includes both an internal database to store Security information as well as an integration with LDAP. The internal security database enables you to set up who can access a resource (Component, Application, Environment, etc.) via the UI as well as who can approve a deployment or other Process. If you are rolling out a production instance of uDeploy, it is recommended to use the LDAP integration.

 **Note:** If you are evaluating uDeploy, it is not necessary to set up the LDAP integration: full security is configured and enforced by the server. However, if you want to send out notifications you will need to set up the LDAP integration.

When setting up Security in uDeploy, you can either use the default configurations or create your own Security setting (unless you are configuring the LDAP integration, both options use the internal database for storage).



**Figure 35: Security Pane**

For both the LDAP integration and the standard security system, Security configuration is performed on the Settings > Security page, and consists of the following:

- **Authorization.** Authorization Realms are used by Authentication Realms to associate Users with Roles and to determine user access to uDeploy. There are two basic Authorization Realms in uDeploy: the default Realm and the LDAP realm. When setting up Security, the first step is to configure Authorization.
- **Authentication.** The Authentication Realm is used to determine a user's identity within an Authorization Realm. The User authentication is determined following the hierarchy of realms displayed on the Authentication Realms. When a User attempts to log in, uDeploy will poll all the configured Authentication Realms for matching credentials.
- **Schemas.** The Security Schemas are visual representations of the different parts of uDeploy that may be secured. Each Schema interacts with Users indirectly, through the Role. To configure security for any of the schemas, you configure what are called Roles. In uDeploy, a Role is used to determine the type of permissions a user is assigned (execute, read, security, write). For example, if a user is assigned the "admin" Role, and the "admin" role has complete access to view, configure, and run Application Processes, that User will have access to that page. In addition, the Role will also need to be assigned to additional schemas so the User can configure Applications, etc. Typically, you will need to add new Roles to a schema on initial setup, and then occasionally as need dictates.
- **Dynamic Roles.** These give you a quick way to grant all users a specific set of permissions at once, regardless of the User's assigned Group or Role, corresponding to the selected Schema. For example, creating a Dynamic Role for Applications grants the selected permissions to every user in the system.
- **UI security.** Corresponding to the Roles created in the UI Security Schema, use this section to quickly assign a user permissions to the different areas of uDeploy. For those with the Security permission (i.e., they see the Security tab in the UI) you can easily add an individual user or a Group of users to any resource. In the example below, an "admin" user is assigning new users, based on the Group they have been assigned, to an Application.

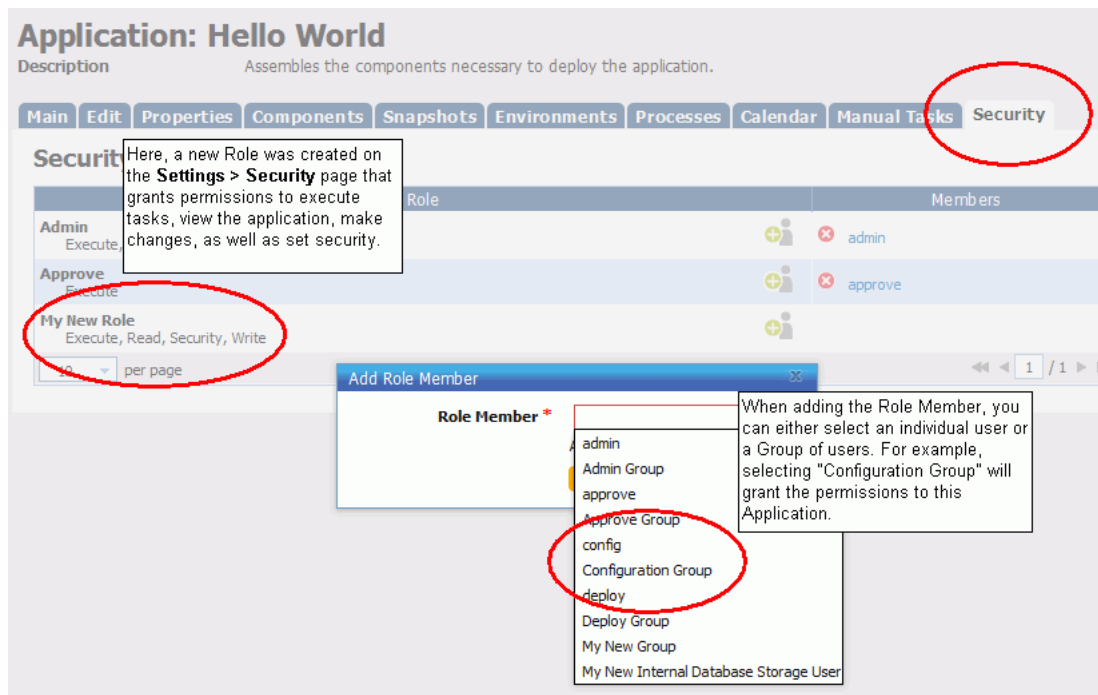


Figure 36: Security Pane

## Authentication

The LDAP integration enables you to map uDeploy Groups and Roles to your existing infrastructure. Once the integration is configured, when a user logs into uDeploy using their LDAP credentials, the system will automatically add them as a user.

You will need to first set up a dedicated Authorization Realm for LDAP. The LDAP Authorization Realm uses an external LDAP server for authorization. If User Roles are defined in LDAP as an attribute of the User, the LDAP Role Attribute configuration must be used. If User Roles are defined elsewhere in LDAP and reference the Users that belong to them, a LDAP Role Search needs to be performed. Once you have the Authorization set up, configure the Authentication Realm, which enables uDeploy to determine a User's identity as defined by LDAP.

### Configuring LDAP Integration

1. Go to Settings > Security > Create New Authorization Realm

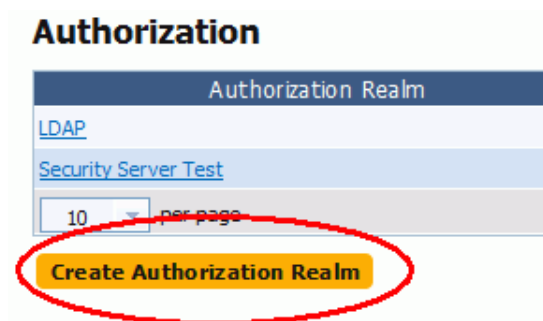


Figure 37: Create Authorization Realm

2. Now, provide uDeploy with information about your LDAP set up. You can either use the User Group Attribute, Group Search Base, or a combination of settings to import. Once configured, and you set up the Authentication Realm (covered in the next section) uDeploy will import your LDAP information.

**Authentication Realm: LDAP**

Description

Edit Users

Name \* LDAP

Description

Authorization Realm \* LDAP

Type \* LDAP

Context Factory \* com.sun.jndi.ldap.LdapCtxFactory

LDAP URL \* ldap://ldap.yourcompany.com/lda

User DN Pattern

User Search Base ou=employees,dc=yourcompany

User Search Filter uid={0}

Search User Subtree

Search Connection DN

Search Connection Password

Name Attribute givenName

Email Attribute mail

All fields marked with \* are required.

Save Cancel

**Figure 38: Authorization Realm Dialog**

**Name and description.** The name you give here will be used when configuring the Authentication Real. M the Authorization Realm.

**Type.** Select LDAP from the drop down.

**User Group Attribute.** Give the name of the attribute that contains role names in the user directory entry. If User Groups are defined in LDAP as an attribute of the User, the Group Attribute configuration must be used.

**Group Search Base.** Give the base directory to execute Group searches in (e.g., ou=groups,dc=mydomain,dc=com). The will determine the Group that Urban Deploy will add any new users to.

**Group Search Filter.** Provide the LDAP filter expression to use when searching for user Group entries. The user name will be put in place of {1} in the search pattern and the full user DN will be put in place of {0} (e.g., member={0}). The will determine the Group that Urban Deploy will add any new users to.

**Group Name.** Give the name of the entry that contains the user's Group names in the directory entries returned by the Group Search. If left blank, no search will take place.

**Search Group Subtree.** Check the box to have uDeploy search the Group Subtree for the Users. Leave blank to not search the Subtree.

3. Next, you need to configure an Authentication Realm. The Authentication Realm is used to determine a users identity within an Authorization Realm, based on LDAP. The User authentication is determined following the hierarchy of realms displayed on the Authentication Realms. When a Users attempts to log in, uDeploy will poll all the configured Authentication Realms for matching credentials.

When configuring the LDAP Authentication Realm, you need to give uDeploy the location of your LDAP server, as well as provide information similar that given for the Authentication Realm.

**Name and description.** The name you give here will be used when configuring the Authentication Real.

**Authorization Realm.** Select the Real created in the previous step.

**Type.** Select LDAP from the drop down.

**Context Factory.** Give the context factory class used to connect. This may vary depending upon your specific Java implementation. The default for Sun Java implementations: com.sun.jndi.ldap.LdapCtxFactory

**LDAP URL.** Provide the full URL to the LDAP server, beginning with ldap:// (e.g., ldap://ldap.mydomain.com:389)

**User DN Pattern.** Give the user directory entry pattern. The user name will be put in place of {0} in the pattern (e.g., cn={0},ou=employees,dc=yourcompany,dc=com).

**User Search Base.** Give the base directory to execute Group searches in (e.g., ou=employees,dc=mydomain,dc=com).

**User Search Filter.** Provide the LDAP filter expression to use when searching for user entries (e.g., uid={0}).

**Search User Subtree.** Check the box to have uDeploy search the User Subtree for the entries. Leave blank to not search the Subtree.

**Search Connection DN.** Give the directory name to use when binding to the LDAP for searches (e.g., cn=Manager,dc=mycompany,dc=com). If not specified, an anonymous connection will be made. Connection Name is required if the LDAP server cannot be anonymously accessed.

**Search Connection Password.** Give the password uDeploy should use when connecting to LDAP to perform searches.

**Name Attribute.** Give the attribute that contains the user's name, as set in LDAP.

**Email Attribute.** Give the attribute that contains the user's email address, as set in LDAP.

Once the configuration is complete, when a new user logs into uDeploy using their LDAP credentials, they will be listed on the Authentication Realm User tab. Since uDeploy relies on LDAP for authentication, it is best practice not to manage user passwords nor remove users from the list. If an active user is removed from uDeploy, they will still be able to log onto the server as long as their LDAP credentials are valid. If this happens, you may also need to set up UI and other permissions for the user.

4. Assign Group to Role. When a User has logged into uDeploy using LDAP credentials, uDeploy automatically assigns the new User to a Group, based on the information pulled from LDAP. In the example below, when "New LDAP User" logged on to uDeploy, they were automatically added to the LDAP Default group. If a user logs on to uDeploy and they are part of a mapping that is not currently associated with a Group, uDeploy will create a new Group based on the information fetched from LDAP. Conversely, if a user logs onto uDeploy and their LDAP credentials map to an existing Group, they will be automatically added to that Group.

User	Name	Email	
miw			<a href="#">Reset Password</a> <a href="#">Remove</a>
pab			<a href="#">Reset Password</a> <a href="#">Remove</a>

10 per page 2 records - [Refresh](#)

**Figure 39: Edit Users**

Once the new user has been successfully added to a Group, you may need to configure additional permissions. This may happen when the new User is mapped to a Group that has limited permissions (e.g., the User has UI permissions but not access to view any Components, Applications; the user was added to a Group that can only access the Work Items and they need to be able to deploy an application, etc.). When this is the case, you will need to set up security for the user, as outlined in the previous section.

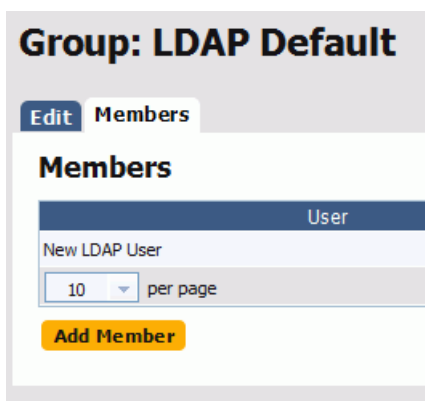



Figure 40: Group Dialog

## Authorization

When setting up Security, there is no optimal process to follow; however using the following order presented below can help you find your way. For most evaluations, starting out with the default Security settings should be adequate and require minimal configuration. What is presented below assumes you are setting up a custom Security System from scratch. In all likelihood.

 **Note:** If you are using the LDAP integration set that up first before continuing. See Configure LDAP Integration.

1. Go to Settings > Security > Create New Authorization Realm. You will select this Authorization Realm in the next step. This Realm is used to ensure people attempting to log on to the server are allowed to.
2. Next, configure an Authentication Realm and add Users. The Authentication Realm is used to determine a users identity within an Authorization Realm. The User authentication is determined following the hierarchy of realms displayed on the Authentication Realms. When a Users attempts to log in, uDeploy will poll all the configured Authentication Realms for matching credentials.

When adding a new user, the user name and password is what the individual will use when logging into uDeploy. The user name will also be displayed when setting up additional Security. Unless you are using the LDAP Integration, uDeploy, which does not have its own e-mail server, will not be able to send notifications to the e-mail address.

3. Add new Group and assign a User (member) to the Group. A Group is a logical identifier for that similar Users are identified with. It is at the Group level that individual Users are manually added to uDeploy. Once the Group container is created, select it from the list and then manually enter the new User.
4. Next, create a new Role. The purpose of the Role is to assign permission that allows Users with that Role to use uDeploy. For example, if you are setting up a new user that must access every page in uDeploy, you must add a new Role to each Schema. Most users will only be required to add Roles on initial set up, and then occasionally as needs arise. Since the Schemas work independently of each other, you will need to create a new Role for each, defining the permissions that you want the role to have for the individual Schema.

**Role: Test Role**  
Description

Name \*

Description

Execute

Read

Security

Write

All fields marked with \* are required.

**Figure 41: Role Pane**

5. Finally, go to the specific Applications, Components, Environments, etc., and add either individual Users or the Group they participate in. If you have many different individuals that must access a resource, say an Application, the most efficient way to give them access is to add the Group that they are assigned to. If this is done, when future users are added to uDeploy, you will not need to manually add them to the resources they need access to.





---

# Part

# V

---

## Reference


---

### Topics:

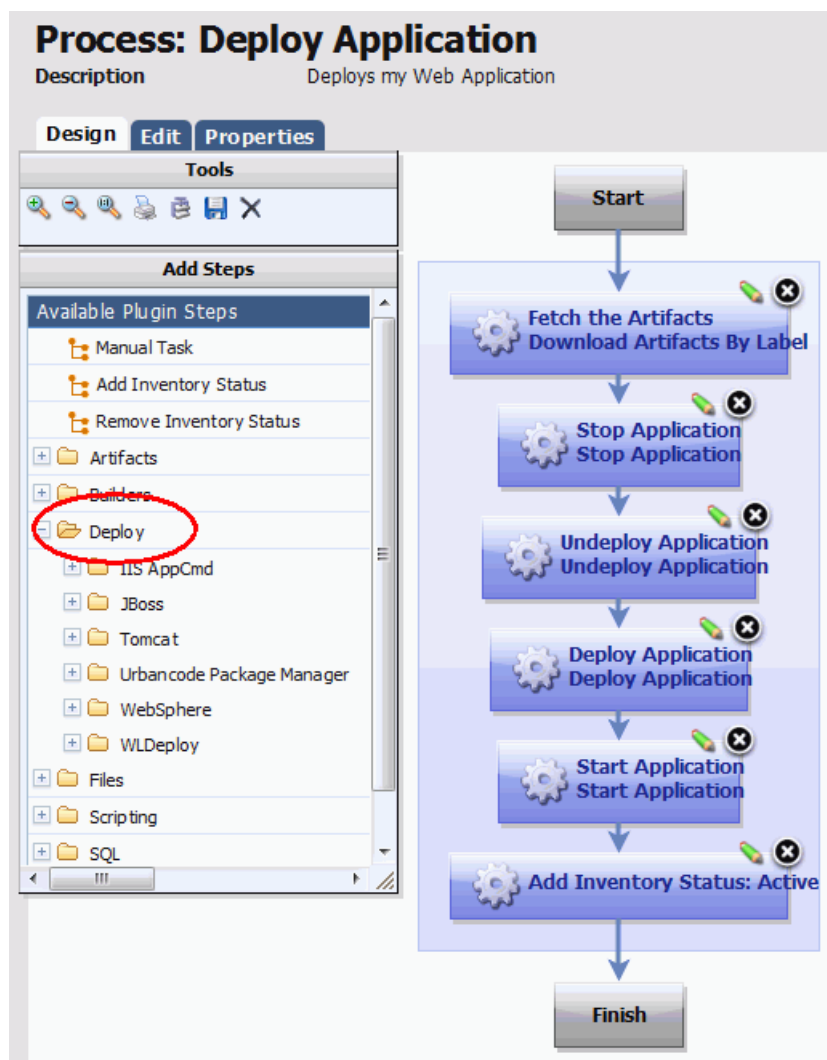
- [Plug-in Integration](#)
  - [Notifications](#)
  - [Configuration](#)
  - [Inventory](#)
  - [CLI Reference](#)
-

## Plug-in Integration

uDeploy plug-ins provide deployment capabilities with many of the common tools used for deployments, as well as application servers, etc. Each integration has at least one "step," which can be thought of as a distinct piece of automation. By stringing these individual steps together, you create a fully automated Process that replaces many of your existing deployment scripts and manual deployment processes. For example, the integrations with Tomcat, WebSphere, etc., are able to start and stop servers, install and uninstall applications, as well as perform other tool-specific tasks.

 **Note:** Before using one of the integrations, it is recommended that you understand what a Component Process is and how a deployment is actually run in uDeploy. If not already done so, you can review the Components section to see how a deployment is set up; then, the Applications section takes you through the steps necessary to actually run a deployment.

The integration steps, which automate distinct deployment tasks, are added to a deploy Process at the Component level (i.e., when setting up a Component Process). As you create a deployment, you start out with the basic deployment configuration (the Download Artifacts By Label step first; the Add Inventory Status last) and then add the integration steps between the steps. In the illustration, the process shows configuration for deploying an application. The Process (a.) stops a running instance of the application; (b.) removes the application from the machine; (c.) installs the new version of the application; and (d.) restarts the application to finish the deployment.



Your deploy jobs will vary, depending on your existing processes. Most users can will end up with a process similar to the one in the illustration, regardless of the integration they use. Because there is no way to predict how your processes are set up, you may need to mix and match steps from each scenario.

uDeploy also includes a number of tools for automating other processes that don't fit neatly into the integration steps, or when it is impossible to completely replace an existing script. For example, your deployment may require running a Ant task, a Groovy script, or even execute SQL statements.

### Plug-in Integrations at Runtime

Because the integrations drive other tools, you will need to ensure that, when you run a deployment, uDeploy is actually able to execute the steps you configured.

Typically this will require you to install agents (Resources) on particular machines in the target Environments. Unless otherwise stated, the following guidelines applies to all the integrations:

1. The agents (Resources) selected to run an integration step must be installed on the same physical machine as the Application. For example, if your deploy jobs includes the step "Stop WebSphere Application," the agent (Resource) must be on the target server to run the command.
2. The Resources running the step must be installed as a user with appropriate permissions to both execute commands as well as access the tool. This typically entails granting permissions on the machine if the external tool is installed as a different user; installing the agent as a service; or, in some cases, installing the agent as ROOT (which should be avoided is possible).
3. The required minimum version of the external tool must be used. If stated, some of integrations require a minimum version of a third-party tool (e.g., WebSphere 5.1 or above). While it may be possible to use the integrations with older versions of the third-party tool, UrbanCode can't guarantee that it will work.

If you need to install new agents of modify Resources, or need to gather more information before using one of the integrations, the Resources and Getting Started section may be helpful.

### Ant Plug-in

The Ant integration consists of a single step that you can include in any deployment process or other process. The most common use case is running Ant Tasks on the target machine. If the step is used within a larger process, ensure that you set the order correctly. For example, if you have to run an Ant script prior to executing another process, you will need to add the Ant step above the other step.

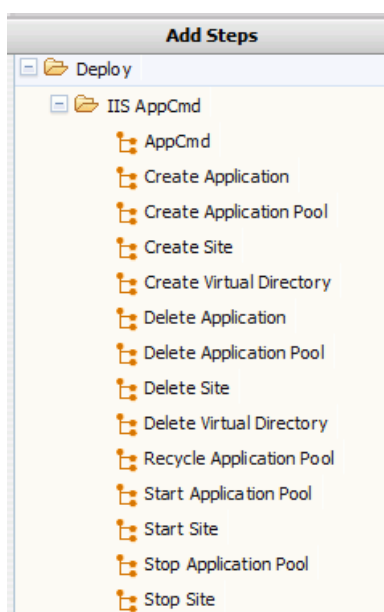
### Groovy Plug-in

The Groovy integration consists of a single step that you can include in any deployment process or other process. The most common use case is running a Groovy script on the target machine. If the step is used within a larger process, ensure that you set the order correctly. For example, if you have to run Groovy prior to executing another process, you will need to add the Groovy step above the other step.

### IIS\_AppCmd Plug-in

Use the integration to add IIS to your deploy processes and run deployments using MSDeploy. The integration enables uDeploy to run a MSDeploy command; start, stop and recycle applications in IIS; as well delete and synchronize IIS objects.

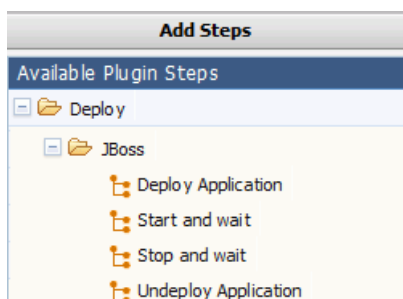
Please note that you will need to select the appropriate Resource: i.e., ensure that you use the agent installed on the same machine as the application/configuration you are syncing. You will also need to provide the path to the msdeploy.exe.



## JBoss Plug-in

Use the integration to add JBoss to your deploy processes. The integration enables uDeploy to run commands to start, stop, deploy and undeploy an application on JBoss. To start using the integration, you will need to configure a deploy process that uses the JBoss steps. How you configure your deploy job will depend on your existing JBoss processes. Generally, you will need to order the job steps to:

1. Stop the application
2. Undeploy the application
3. Deploy the application
4. Start the application



Before setting up the integration, ensure the Resource has access to the deploy directory the JBoss manages.

## SQL/JDBC Plug-in

The SQL-JDBC integration consists of a single step that you can include in any deployment process or other process. The most common use case opening and running a SQL statement when updating a database. If the step is used within a larger process, ensure that you set the order correctly. For example, if you have to run a SQL statement prior to executing another process, you will need to add the step above the other step.

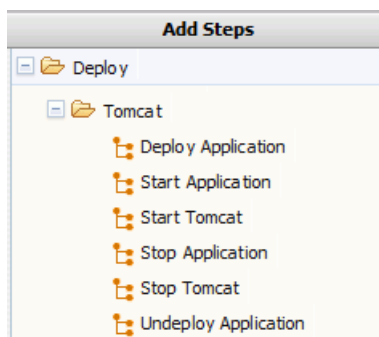
## SQLPLUS Plug-in

The Oracle SQL-Plus integration consists of a single step that you can include in any deployment process or other process. The most common use case opening and running a SQL statement when updating a database. If the step is used within a larger process, ensure that you set the order correctly. For example, if you have to run a SQL statement prior to executing another process, you will need to add the step above the other step.

## Tomcat Plug-in

Use the integration to add Tomcat to your deployment processes. The integration enables uDeploy to run commands to start, stop, deploy and undeploy an application on Tomcat. To start using the integration, you will need to configure a deploy process that uses the Tomcat steps. How you configure your deploy process will depend on your existing Tomcat processes. Generally, you will need to order the job steps to:

1. Stop the application
2. Undeploy the application
3. Deploy the application
4. Start the application

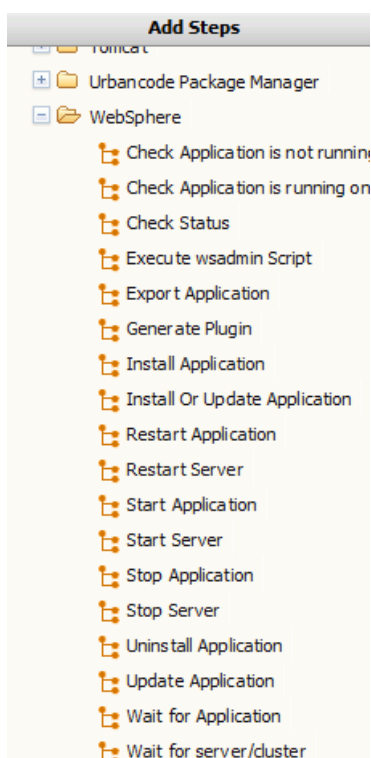


When running the process, ensure that the Resource running the step has access to the Tomcat full client jar, that uDeploy has a user and password to log to connect with, and that the full path to the Tomcat full client jar is available.

## WebSphere Plug-in

Use the integration to run commands that start and stop your WebSphere server and applications; install a new application; update an application; as well as execute a wsadmin script. To start using the integration, in your WebSphere properties files you need to add the user name and password uDeploy will use when connecting. Once this is done, you can then set up your WebSphere deploy jobs. How you configure your deploy job will depend on your existing WebSphere processes. Generally, you will need to order the job steps to:

1. Resolve artifacts
2. Stop the application/sever
3. Update/uninstall the application
4. Start the application/server

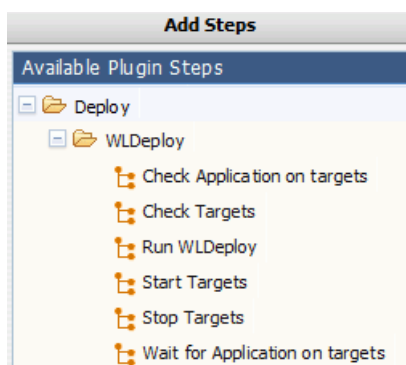


When setting up your deployment, you select one of the pre-defined steps and add it to your process. Step configuration is straightforward: you generally need to give connection information as well as the location to any executables.

## WLDeploy Plug-in

Use the integration to add WLDeploy to your deployment processes. The integration enables uDeploy to run commands to start, stop, deploy and undeploy an application on Tomcat. To start using the integration, you will need to configure a deploy process that uses the Tomcat steps. How you configure your deploy process will depend on your existing Tomcat processes. Generally, you will need to order the job steps to:

1. Stop the application
2. Undeploy the application
3. Deploy the application
4. Start the application



When running the process, ensure that the Resource running the step has access to the Tomcat full client jar, that uDeploy has a user and password to log to connect with, and that the full path to the Tomcat full client jar is available.

## Standard Component Process Steps

uDeploy also includes a standard set of automation steps that can be used to add additional automation to any process. These will typically be used for advanced processes or where there is no standard integration step available from one of the integrations.

### Shell

The Shell integration consists of a single step that you can include in any deployment process or other process. The most common use case opening and running a shell script on the target machine. If the step is used within a larger process, ensure that you set the order correctly. For example, if you have to run a shell script prior to executing another process, you will need to add the Shell step above the other step.

### UrbanCode Package Manager

This is for advanced usage. The steps work in conjunction with uDeploy to create and manage application packages for deployments. These steps will not generally be used as part of a regular deployment.

### uDeploy

These advanced automation steps will retrieve properties and environments from uDeploy.

## Notifications

uDeploy can send notifications to users based on a number of events that occur. Most commonly, uDeploy is configured to send an e-mail regarding the state of a deployment (success or failure) or when an Approval is required. The recipient list of these notifications must be tied to the LDAP integration, etc., (see Security for more), allowing you an easy way to integrate uDeploy with your existing infrastructure. If you have not already done so, set up uDeploy with LDAP prior to configuring Notifications: uDeploy relies on LDAP, and the associated e-mail server, to send notifications. When setting up notifications, you select both the events and the Role, which is inherited from the Security System, to determine which users will be notified and when. For example, it is common for an administrator or environment owner to be notified when a Work Item (as part of the Approvals Process) has been generated. The Default Notification Scheme, which sends out notifications to the Application and Admin default Roles (see Security for more), can be edited or you can create your own Notification Scheme.



**Note:** Once a Notification Scheme is created, it will be used when setting up your Applications (see here for an example).

To set up your own notifications, go to Settings > Notifications > Create New Notification Scheme page.

The screenshot shows the uDeploy web interface. At the top right, the user is logged in as 'admin' with a 'Logout' link. The main navigation bar includes 'Components', 'Applications', 'Resources', 'Deployment Calendar', 'Work Items', and 'Settings'. The 'Settings' link is circled in red. Below the navigation bar, the breadcrumb path is 'Home > Settings'. The 'Notifications' tab is selected and circled in red. The main content area is titled 'Notification Schemes' and contains a table with the following data:

Notification Scheme	Description	Actions
<a href="#">Default Notification Scheme</a>		
<a href="#">My Notification Scheme</a>		

Below the table, there is a pagination control showing '10 per page' and '2 records - Refresh'. A yellow button labeled 'Create New Notification Scheme' is circled in red.

Figure 42: Notification Schemes

Configure the new Scheme. Here, you will be setting up the who/when for notifications. Once configured, you can come back add additional Entries to the Scheme or edit existing one.

**Notification Type.** The process type is determined mainly by the type of recipient. For example, a deployment engineer would be interested in being notified about a failed deployment.

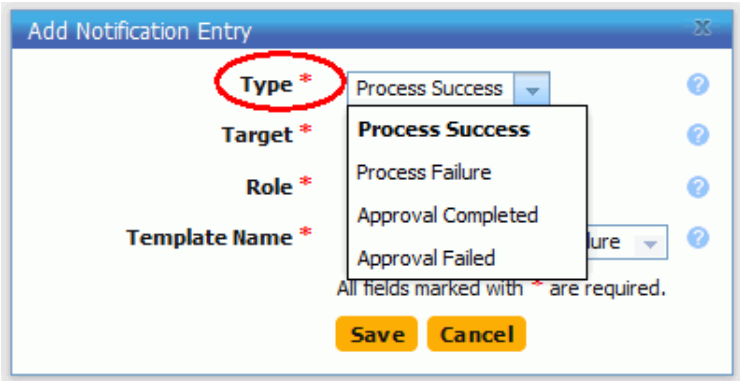


Figure 43: Notification Type

**Notification Target.** When setting the target, the application option will only send out notifications when the event selected above corresponds to an Application. For example, the "Process Success" event, when paired with the "Application" Target would trigger a notification when a Process (an application deployment) is successful. Similarly, the same event type, when used with the "Environment" target would instigate a notification when a successful deployment has been run in an Environment (e.g., SIT, PROD).

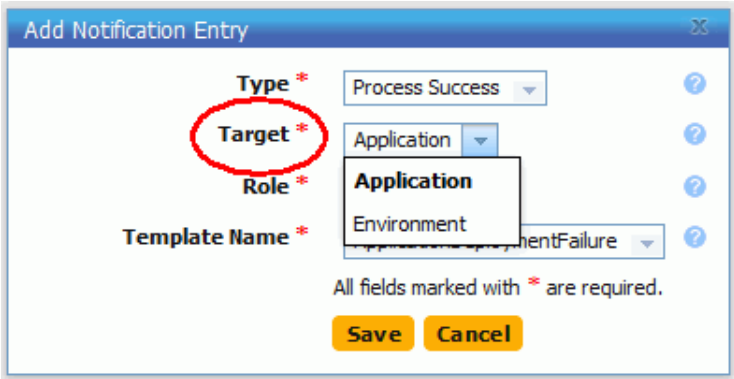


Figure 44: Notification Target

**Notification Role.** The Role corresponds to those set in the Security System. Any individual assigned the Role you select will receive an e-mail.

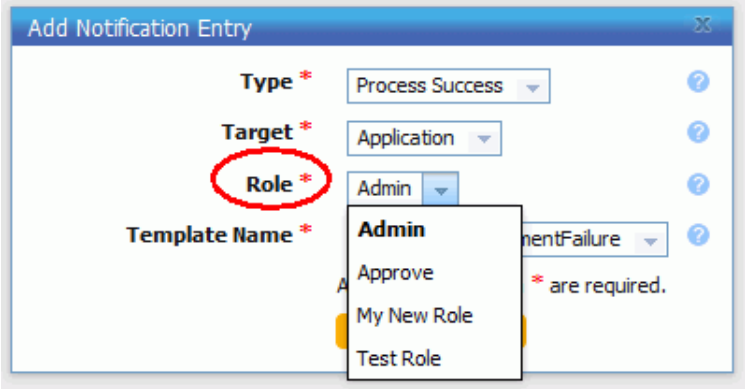
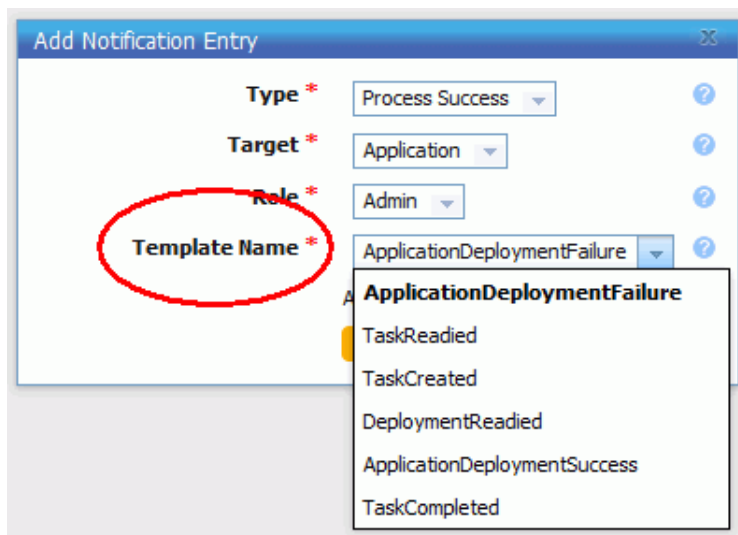


Figure 45: Notification Role



**Template Name.** The available templates are provided by default and should suffice for all your needs; they format the e-mail being sent. Which template you use is based on why you want to set up a notification and the recipients of the notification.



**Figure 46: Template**

Application deployment failure / success. Sends out notifications about a specific Application deployment to the specified users, based on the Role setting above.

Task readied / created / completed. This template is used to report back on the state of manual tasks.

Deployment readied. A specialized e-mail template for letting people know a deployment has been prepared.

Once you have the Entry done, add other Entries to the Scheme following the same process. Note that if you want to use the new Notification Scheme with existing Applications, you will need to modify the Application settings.

## Configuration

---

The Urban Deploy Configuration tool enables you to manipulate configuration data, such as Tomcat or JBoss property files.

Configuration data is manipulated at the application, component, and environment levels:

- **Component**

A component refers to any file that you want to include in the build process; components are associated with the configuration data required to deploy them.

- **Application**

Applications represent a group of components deployed together by component version and environment. Applications also map the hosts and machines (called resources) components require within every environment.

- **Environment**

An environment is a collection of resources that host an Urban Deploy application.

The screenshot shows the UrbanDeploy Configuration Tab. The top navigation bar includes 'urban(deploy)' and 'Hello admin | Help | Logout'. The main navigation tabs are 'Dashboard', 'Components', 'Applications', 'Configuration', 'Resources', 'Deployment Calendar', 'Work Items', and 'Settings'. The breadcrumb trail is 'Home > Configuration'.

The left sidebar shows a tree view of the configuration structure:

- Application / Component / Environment
  - JPetStore application
    - JPetStore-APP component
      - SIT environment
      - UAT environment
      - France1 environment
    - JPetStore-DB component
      - SIT
      - UAT
      - France1
    - JPetStore-WEB component
      - SIT
      - UAT
      - France1
  - stressTestNov1 application

**Figure 47: Configuration Tab**

Access the Configuration Tool by clicking on the Configuration tab.

## Application Configuration

You attach properties to an application by using the Configuration Tool's **Application: Add Property** button.

Typical application-level properties include items that are the same in all environments, such as base-install paths.

The screenshot shows the UrbanDeploy Configuration Tool interface. The top navigation bar includes 'urban{deploy}' logo, 'Hello admin', 'Help', and 'Logout'. The main navigation menu contains 'Dashboard', 'Components', 'Applications', 'Configuration', 'Resources', 'Deployment Calendar', 'Work Items', and 'Settings'. The breadcrumb trail is 'Home > Configuration'. On the left, a tree view under 'Application / Component / Environment' shows a hierarchy: JPetStore (circled in red), JPetStore-APP, SIT, UAT, France1, JPetStore-DB, SIT, UAT, France1, JPetStore-WEB, SIT, UAT, France1, and stressTestNov1. The main panel is titled 'Application: JPetStore' and contains a 'Properties' section with an 'Add Property' button. Below this is a table with columns 'Name', 'Value', 'Description', and 'Actions'. The table contains one row with 'Name' 'someProp', 'Value' empty, 'Description' empty, and 'Actions' 'Edit Delete'. Below the table is a pagination control showing '10 per page', '1 record - Refresh', and '1 / 1'. Another 'Add Property' button is located below the pagination control.

**Figure 48: Application Properties panel**

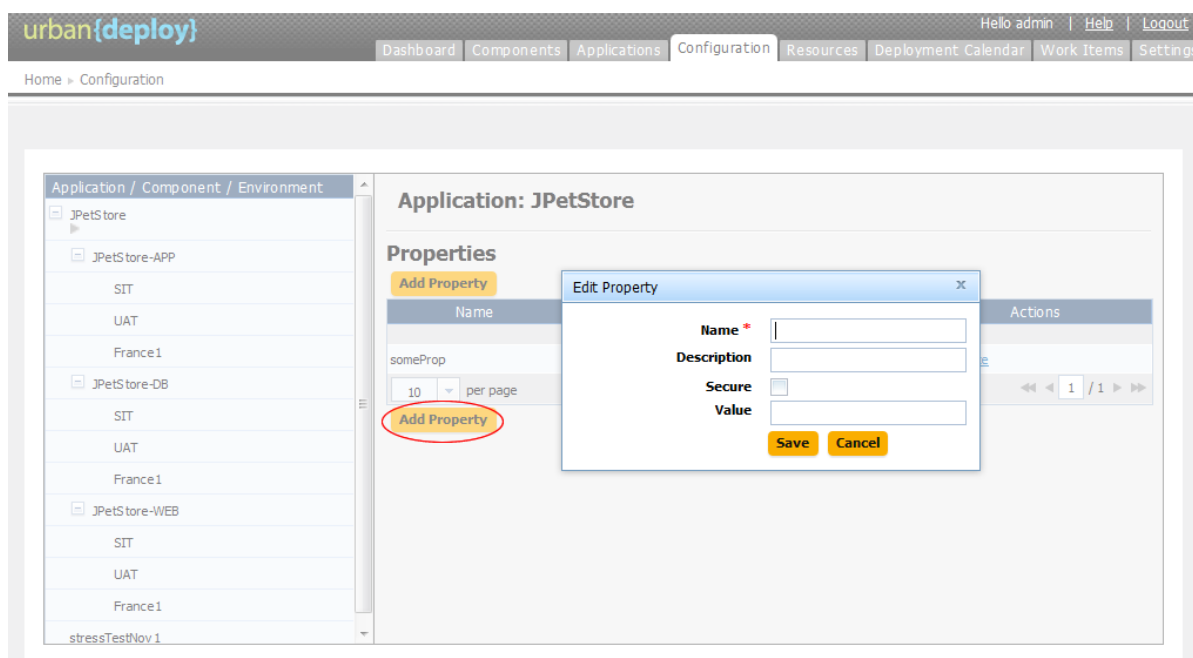
Access the Configuration Tool Application panel by clicking on an application in the **Application/Component/Environment** list box.

### Adding Application Configuration Properties

To add a property to the selected application:

1. Use the **Add Property** button.

The Edit Property pop-up is displayed.



**Figure 49: Edit Property pop-up**

2. Enter the property's name in the **Name** field.

While component fields can be of any size, configuration properties are restricted to 4,000 characters.

3. Enter a description of the property in the **Description** field.
4. Specify whether the property is secure by using the **Secure** check box.

Secure properties are stored encrypted and displayed obscured in uDeploy's user interface.

5. Enter a value for the property in the **Value** field.
6. Save the property by using the **Save** button.
7. To discard your work, use the **Cancel** button.

### Modifying Application Configuration Properties

To modify a previously created property, use the **Edit** link in the Action column to display the Edit Property pop-up.

### Deleting Application Configuration Properties

To delete a property, use the **Delete** link in the Action column.

## Component Configuration

The Urban Deploy Configuration tab enables you to configure applications and their components from a single location.

Configuration data is manipulated at the application, component, and environment levels:

- component

A component refers to any file that you want to include in the build process; components are associated with the configuration data required to deploy them.

- application

Applications represent a group of components deployed together by component version and environment.

Applications also map the hosts and machines (called resources) components require within every environment.

- environment

An environment is a collection of resources that host an Urban Deploy application.

The screenshot shows the Urban Deploy Configuration interface. The top navigation bar includes 'urban{deploy}' logo, 'Hello admin', 'Help', and 'Logout'. Below the navigation bar are tabs for 'Dashboard', 'Components', 'Applications', 'Configuration', 'Resources', 'Deployment Calendar', 'Work Items', and 'Settings'. The 'Configuration' tab is active, and the breadcrumb path is 'Home > Configuration'.

The main content area is divided into two sections. On the left is a tree view titled 'Application / Component / Environment'. It shows a hierarchy: 'JPetStore' (expanded) containing 'JPetStore-APP' (circled in red), 'SIT', 'UAT', and 'France1'; 'JPetStore-DB' containing 'SIT', 'UAT', and 'France1'; and 'JPetStore-WEB' containing 'SIT', 'UAT', and 'France1'. At the bottom of the tree is 'stressTestNov 1'.

The right section is titled 'Component: JPetStore-APP' and contains three main sections:

- Properties:** Includes an 'Add Property' button and a table with columns 'Name', 'Value', 'Description', and 'Actions'. The table is currently empty, with a message 'No properties found. - Refresh' and a 'Show Filters' link.
- Environment Property Definitions:** Includes an 'Add Property' button and a table with columns 'Name', 'Label', 'Required', 'Default Value', 'Description', and 'Actions'. The table is currently empty, with a message 'No properties have been defined. - Refresh'.
- Configuration Templates:** Includes a 'Create New Configuration Template' button and a table with columns 'Name' and 'Actions'. The table is currently empty, with a message 'No configuration templates found. - Refresh'.

**Figure 50: Configuration Tab**

Access the Configuration Tool by clicking on the Configuration tab.

## Environment Configuration

The Urban Deploy Configuration tab enables you to configure applications and their components from a single location.

Configuration data is manipulated at the application, component, and environment levels:

- component

A component refers to any file that you want to include in the build process; components are associated with the configuration data required to deploy them.

- application

Applications represent a group of components deployed together by component version and environment.

Applications also map the hosts and machines (called resources) components require within every environment.

- environment

An environment is a collection of resources that host an Urban Deploy application.

The screenshot shows the UrbanDeploy web interface. The top navigation bar includes 'urban{deploy}' logo, 'Hello admin', 'Help', and 'Logout'. Below the navigation bar are tabs for 'Dashboard', 'Components', 'Applications', 'Configuration', 'Resources', 'Deployment Calendar', 'Work Items', and 'Settings'. The 'Configuration' tab is active, and the breadcrumb 'Home > Configuration' is visible.

The main content area is titled 'Configuration of JPetStore-APP in SIT'. On the left, a tree view shows the hierarchy: Application / Component / Environment. Under 'JPetStore', there is a sub-component 'JPetStore-APP' which contains three environments: 'SIT' (highlighted with a red circle), 'UAT', and 'France1'. Below this, there are sub-components 'JPetStore-DB' and 'JPetStore-WEB', each with 'SIT', 'UAT', and 'France1' environments. At the bottom of the tree is 'stressTestNov 1'.

The main panel displays the configuration for the selected 'SIT' environment. It is titled 'Configuration of JPetStore-APP in SIT' and has a sub-section 'Resources and Groups'. Below this is a table:

Resource / Group	Type	# Resources	Actions
/SIT/APP	static	1	<a href="#">Remove</a>
urbandeploy-agent			<a href="#">Remove</a>

Below the table are two buttons: 'Add a Resource' and 'Add a Resource Group'. Underneath is a section titled 'Environment Properties' with the text: 'No environment properties have been defined by this component.'

**Figure 51: Environment Configuration Tab**

Access the Configuration Tool by clicking on the Configuration tab.

## Inventory

The Inventory shows what Applications and Components have been deployed, including the current Versions that are running on the Resource within an Environment. The inventory provides complete visibility into the different Versions of your Applications which can be tracked back to the original artifacts imported into UrbanDeploy. There are different views of the current inventory, depending on where in UrbanDeploy you are. Inventory information is available on the individual Components, for every Application Environment, as well as for each Resource (agent).

### Resources Inventory

If you want to see what Components are sitting on the SIT Environment, go to Resources and select the agent that is running in the Environment. From here, selecting either the Component or its Version will take you to the Component's page if you need more information.

urban{deploy}

Components Applications **Resources** Deployment Calendar

Home » Resources » SIT Environment

## Resource: SIT Environment

Description Agent installed in SIT.

Main Edit Roles Properties Custom Properties Security

### Current Inventory

Component	Version	Date	Status
<a href="#">My Application (WEB)</a>	<a href="#">1.0</a>	8/18/11 10:43 AM	Active
<a href="#">My Application (APP)</a>	<a href="#">1.0</a>	8/18/11 10:43 AM	Active
<a href="#">My Application (DB)</a>	<a href="#">1.0</a>	8/18/11 10:36 AM	Staged

10 per page 3 records - [Refresh](#)

**Figure 52: Resource inventory**

### Component Inventory

Unlike the Resource Inventory, the Component Inventory tells you what Version of the Component is running on a Resource. For example, if the Component is currently deployed to multiple machines, they would all be displayed. For here, you can go navigate to the Resource.

urban{deploy}

Home ▶ Components ▶ My Application (WEB)

**Component: My Application (WEB)**  
Description: The web tier of My Application.

Main Edit **Inventory** Calendar Properties Versions P

**Resources With This Component**

Resource	Version	
<a href="#">SIT Environment</a>	<a href="#">1.0</a>	8/18/11 10:43

10 per page 1 record - !

Figure 53: Component inventory

### Environment Inventory

For any given Application Environment, the Inventory tells you both what version of any given Component is running on a particular Resource. If multiple Versions are running on different Resources, they will all be listed.

urban{deploy}

Home ▶ Applications ▶ Hello World ▶ Environments ▶ Environment: SIT

**Environment: SIT for Hello World**  
Description: Unrestricted testing environment

Main **Inventory** Calendar Component Mappings Edit Security

**Current Inventory By Version - [View By Resource](#)**

Component / Resource	Version	Date
<a href="#">My Application (APP)</a>	<a href="#">1.0</a>	
<a href="#">SIT Environment</a>		8/18/11 10:43 AM

Figure 54: Environment Inventory



## CLI Reference

---

### addActionToRoleForApplications

Add action to a role for applications.

#### Format

```
udclient [global-args...] [global-flags...]  
addActionToRoleForApplications [args...]
```

#### Options

```
-role, --role  
    Required. Name of the role  
  
-action, --action  
    Required. Name of the action
```

### addActionToRoleForComponents

Add action to a role for components

#### Format

```
udclient [global-args...] [global-flags...]  
addActionToRoleForComponents [args...]
```

#### Options

```
-role, --role  
    Required. Name of the role  
  
-action, --action  
    Required. Name of the action
```

### addActionToRoleForEnvironments

Add action to a role for environments

#### Format

```
udclient [global-args...] [global-flags...]  
addActionToRoleForEnvironments [args...]
```

**Options**

```
-role, --role
    Required. Name of the role

-action, --action
    Required. Name of the action
```

**addActionToRoleForResources**

Add action to a role for resources

**Format**

```
udclient [global-args...] [global-flags...]
addActionToRoleForResources [args...]
```

**Options**

```
-role, --role
    Required. Name of the role

-action, --action
    Required. Name of the action
```

**addActionToRoleForUI**

Add action to a role for the UI

**Format**

```
udclient [global-args...] [global-flags...] addActionToRoleForUI
[args...]
```

**Options**

```
-role, --role
    Required. Name of the role

-action, --action
    Required. Name of the action
```

**addComponentToApplication**

Add a component to an Application.

**Format**

```
udclient [global-args...] [global-flags...] addComponentToApplication
[args...]
```

**Options**

```
-component, --component
    Required. Name of the component to add

-application, --application
    Required. Name of the application to add it to.
```

**addGroupToRoleForApplication**

Add a group to a role for an application

**Format**

```
udclient [global-args...] [global-flags...]
addGroupToRoleForApplication [args...]
```

**Options**

```
-group, --group
    Required. Name of the group

-role, --role
    Required. Name of the role

-application, --application
    Required. Name of the application
```

**addGroupToRoleForComponent**

Add a group to a role for a component

**Format**

```
udclient [global-args...] [global-flags...] addGroupToRoleForComponent
[args...]
```

**Options**

```
-group, --group
```

```

    Required. Name of the group

    -role, --role
        Required. Name of the role

    -component, --component
        Required. Name of the component

```

## addGroupToRoleForEnvironment

Add a group to a role for an environment

### Format

```

    udclient [global-args...] [global-flags...]
    addGroupToRoleForEnvironment [args...]

```

### Options

```

    -group, --group
        Required. Name of the group

    -role, --role
        Required. Name of the role

    -application, --application
        Required. Name of the application

    -environment, --environment
        Required. Name of the environment

```

## addGroupToRoleForResource

Add a group to a role for a resource

### Format

```

    udclient [global-args...] [global-flags...] addGroupToRoleForResource
    [args...]

```

### Options

```

    -group, --group
        Required. Name of the group

    -role, --role
        Required. Name of the role

    -resource, --resource
        Required. Name of the resource

```

## addGroupToRoleForUI

Add a group to a role for the UI

### Format

```
udclient [global-args...] [global-flags...] addGroupToRoleForUI
[args...]
```

### Options

```
-group, --group
    Required. Name of the group

-role, --role
    Required. Name of the role
```

## addLicense

Add a license to the server.

### Format

```
udclient [global-args...] [global-flags...] addLicense [args...]
```

### Options

No options for this command.

## addNameConditionToGroup

Add a name condition to a resource group. Only works with dynamic groups.

### Format

```
udclient [global-args...] [global-flags...] addNameConditionToGroup
[args...]
```

### Options

```
-comparison, --comparison
    Required. Type of the comparison
```

```
-value, --value
    Required. Value of the comparison

-group, --group
    Required. Path of the parent resource group
```

## addPropertyConditionToGroup

Add a property condition to a resource group. Only works with dynamic groups.

### Format

```
udclient [global-args...] [global-flags...]
addPropertyConditionToGroup [args...]
```

### Options

```
-property, --property
    Required. Name of the property

-comparison, --comparison
    Required. Type of the comparison

-value, --value
    Required. Value of the comparison

-group, --group
    Required. Path of the parent resource group
```

## addResourceToGroup

Add a resource to a resource group. Only works with static groups.

### Format

```
udclient [global-args...] [global-flags...] addResourceToGroup
[args...]
```

### Options

```
-resource, --resource
    Required. Name of the resource to add

-group, --group
    Required. Path of the resource group to add to
```

## addRoleToResource

Add a role to a resource.

### Format

```
udclient [global-args...] [global-flags...] addRoleToResource
[args...]
```

### Options

```
-resource, --resource
    Required. Name of the parent resource.

-role, --role
    Required. Name of the new resource.
```

## addRoleToResourceWithProperties

Add a role to a resource. This command takes a JSON request body. Use the `-t` flag to view the template for the data required for this command.

### Format

```
udclient [global-args...] [global-flags...]
addRoleToResourceWithProperties [args...] [-] [filename]

-
    Read JSON input from the stdin. See command for requirements.

filename
    Read JSON input from a file with the given filename. See command
for
    requirements.
```

### Options

```
No options for this command.
```

## addUserToGroup

Add a user to a group

### Format

```
udclient [global-args...] [global-flags...] addUserToGroup [args...]
```

**Options**

```
-user, --user
    Required. Name of the user

-group, --group
    Required. Name of the group
```

**addUserToRoleForApplication**

Add a user to a role for an application

**Format**

```
udclient [global-args...] [global-flags...]
addUserToRoleForApplication [args...]
```

**Options**

```
-user, --user
    Required. Name of the user

-role, --role
    Required. Name of the role

-application, --application
    Required. Name of the application
```

**addUserToRoleForComponent**

Add a user to a role for a component

**Format**

```
udclient [global-args...] [global-flags...] addUserToRoleForComponent
[args...]
```

**Options**

```
-user, --user
    Required. Name of the user

-role, --role
    Required. Name of the role

-component, --component
    Required. Name of the component
```



## addUserToRoleForEnvironment

Add a user to a role for an environment

### Format

```
udclient [global-args...] [global-flags...]
addUserToRoleForEnvironment [args...]
```

### Options

```
-user, --user
    Required. Name of the user

-role, --role
    Required. Name of the role

-application, --application
    Required. Name of the application

-environment, --environment
    Required. Name of the environment
```

## addUserToRoleForResource

Add a user to a role for a resource

### Format

```
udclient [global-args...] [global-flags...] addUserToRoleForResource
[args...]
```

### Options

```
-user, --user
    Required. Name of the user

-role, --role
    Required. Name of the role

-resource, --resource
    Required. Name of the resource
```

## addUserToRoleForUI

Add a user to a role for the UI

**Format**

```
udclient [global-args...] [global-flags...] addUserToRoleForUI
[args...]
```

**Options**

```
-user, --user
    Required. Name of the user

-role, --role
    Required. Name of the role
```

**addVersionFiles**

Upload files to a version

**Format**

```
udclient [global-args...] [global-flags...] addVersionFiles [args...]
```

**Options**

```
-component, --component
    Optional. Name/ID of the component (Only required if not using
    version ID)

-version, --version
    Required. Name/ID of the version

-base, --base
    Required. Local base directory for upload. All files inside this
    will be sent.

-offset, --offset
    Optional. Target path offset (the directory in the version files
to
    which these files should be added)
```

**addVersionStatus**

Add a status to a version

**Format**

```
udclient [global-args...] [global-flags...] addVersionStatus [args...]
```

## Options

```

-component, --component
    Optional. Name/ID of the component (Only required if not using
    version ID)

-version, --version
    Required. Name/ID of the version

-status, --status
    Required. Name of the status to apply

```

## createApplication

Create a new application. This command takes a JSON request body. Use the `-t` flag to view the template for the data required for this command.

### Format

```

    udclient [global-args...] [global-flags...] createApplication
    [args...] [-] [filename]

    -
    Read JSON input from the stdin. See command for requirements.

    filename
    Read JSON input from a file with the given filename. See command
for
    requirements.

```

### Options

No options for this command.

## createApplicationProcess

Create a new application process. This command takes a JSON request body. Use the `-t` flag to view the template for the data required for this command.

### Format

```

    udclient [global-args...] [global-flags...] createApplicationProcess
    [args...] [-] [filename]

    -
    Read JSON input from the stdin. See command for requirements.

    filename
    Read JSON input from a file with the given filename. See command
for
    requirements.

```

**Options**

No options for this command.

**createCommand**

Create a new component. This command takes a JSON request body. Use the `-t` flag to view the template for the data required for this command.

**Format**

```

    udclient [global-args...] [global-flags...] createComponent [args...]
    [-] [filename]

    -
      Read JSON input from the stdin. See command for requirements.

    filename
      Read JSON input from a file with the given filename. See command
for
      requirements.

```

**Options**

No options for this command.

**createCommandProcess**

Create a new component process. This command takes a JSON request body. Use the `-t` flag to view the template for the data required for this command.

**Format**

```

    udclient [global-args...] [global-flags...] createComponentProcess
    [args...] [-] [filename]

    -
      Read JSON input from the stdin. See command for requirements.

    filename
      Read JSON input from a file with the given filename. See command
for
      requirements.

```

**Options**

No options for this command.

## createDynamicResourceGroup

Create a new static resource group.

### Format

```
udclient [global-args...] [global-flags...] createDynamicResourceGroup
[args...]
```

### Options

```
-path, --path
  Required. Path to add the resource group to (parent resource
group
  path).

-name, --name
  Required. Name of the new resource group.
```

## createEnvironment

Create a new environment.

### Format

```
udclient [global-args...] [global-flags...] createEnvironment
[args...]
```

### Options

```
-application, --application
  Required. Application to add the environment to.

-name, --name
  Required. Name of the new environment.

-description, --description
  Optional. Description of the new environment.

-color, --color
  Optional. Color of the new environment.

-requireApprovals, --requireApprovals
  Optional. Does the environment require approvals?
```

## createGroup

Add a new group

### Format

```
udclient [global-args...] [global-flags...] createGroup [args...]
```

### Options

```
-group, --group
    Required. Name of the group
```

## createMapping

Create a new mapping.

### Format

```
udclient [global-args...] [global-flags...] createMapping [args...]
```

### Options

```
-environment, --environment
    Required. The environment for the mapping.

-component, --component
    Required. The component for the mapping.

-resourceGroupPath, --resourceGroupPath
    Required. The resource group for the mapping.

-application, --application
    Optional. The application for the mapping. Only necessary if
    specifying env name instead of id.
```

## createResourceGroup

Create a new static resource group.

### Format

```
udclient [global-args...] [global-flags...] createResourceGroup
[args...]
```

**Options**

```

    -path, --path
    group      Required. Path to add the resource group to (parent resource
               path).

    -name, --name
               Required. Name of the new resource group.

```

**createRoleForApplications**

Create a role for applications

**Format**

```

    udclient [global-args...] [global-flags...] createRoleForApplications
    [args...]

```

**Options**

```

    -role, --role
               Required. Name of the role

```

**createRoleForComponents**

Create a role for components

**Format**

```

    udclient [global-args...] [global-flags...] createRoleForComponents
    [args...]

```

**Options**

```

    -role, --role
               Required. Name of the role

```

**createRoleForEnvironments**

Create a role for environments

**Format**

```

    udclient [global-args...] [global-flags...] createRoleForEnvironments
    [args...]

```

```
udclient [global-args...] [global-flags...] createRoleForEnvironments
[args...]
```

### Options

```
-role, --role
    Required. Name of the role
```

## createRoleForResources

Create a role for resources

### Format

```
udclient [global-args...] [global-flags...] createRoleForResources
[args...]
```

### Options

```
-role, --role
    Required. Name of the role
```

## createRoleForUI

Create a role for the UI

### Format

```
udclient [global-args...] [global-flags...] createRoleForUI [args...]
```

### Options

```
-role, --role
    Required. Name of the role
```

## createSubresource

Create a new subresource.

### Format

```
udclient [global-args...] [global-flags...] createSubresource
[args...]
```



**Options**

```

-parent, --parent
    Required. Name of the parent resource.

-name, --name
    Required. Name of the new resource.

-description, --description
    Optional. Description of the resource.

```

**createUser**

Add a new user This command takes a JSON request body. Use the `-t` flag to view the template for the data required for this command.

**Format**

```

    udclient [global-args...] [global-flags...] createUser [args...] [-]
    [filename]

    -
    Read JSON input from the stdin. See command for requirements.

    filename
    Read JSON input from a file with the given filename. See command
for
    requirements.

```

**Options**

No options for this command.

**createVersion**

Create a new version for a component

**Format**

```

    udclient [global-args...] [global-flags...] createVersion [args...]

```

**Options**

```

-component, --component
    Required. Name/ID of the component

```

```
-name, --name
    Required. Name of the new version
```

## deleteGroup

Delete a group

### Format

```
udclient [global-args...] [global-flags...] deleteGroup [args...]
```

### Options

```
-group, --group
    Required. Name of the group
```

## deleteResourceGroup

null

### Format

```
udclient [global-args...] [global-flags...] deleteResourceGroup
[args...]
```

### Options

```
-group, --group
    Required. Path of the resource group to delete
```

## deleteResourceProperty

Remove a custom property from a resource

### Format

```
udclient [global-args...] [global-flags...] deleteResourceProperty
[args...]
```

### Options

```
-resource, --resource
    Required. Name of the resource to configure
```

```
-name, --name
  Required. Name of the property
```

## deleteUser

Delete a user

### Format

```
udclient [global-args...] [global-flags...] deleteUser [args...]
```

### Options

```
-user, --user
  Required. Name of the user
```

## exportGroup

Add a new group

### Format

```
udclient [global-args...] [global-flags...] exportGroup [args...]
```

### Options

```
-group, --group
  Required. Name of the group
```

## getApplication

Get a JSON representation of an application

### Format

```
udclient [global-args...] [global-flags...] getApplication [args...]
```

### Options

```
-application, --application
  Required. Name of the application to look up
```

## getApplicationProcess

Get a JSON representation of an Application Process

### Format

```
udclient [global-args...] [global-flags...] getApplicationProcess
[args...]
```

### Options

```
-application, --application
    Required. Name of the application

-applicationProcess, --applicationProcess
    Required. Name of the process
```

## getApplicationProcessRequestStatus

Repeat an application process request.

### Format

```
udclient [global-args...] [global-flags...]
getApplicationProcessRequestStatus [args...]
```

### Options

```
-request, --request
    Required. ID of the application process request to view
```

## getApplications

Get a JSONArray representation of all applications

### Format

```
udclient [global-args...] [global-flags...] getApplications [args...]
```

### Options

No options for this command.

## getComponent

Get a JSON representation of a component

### Format

```
udclient [global-args...] [global-flags...] getComponent [args...]
```

### Options

```
-component, --component  
    Required. Name of the component to look up
```

## getComponentProcess

Get a JSON representation of a componentProcess

### Format

```
udclient [global-args...] [global-flags...] getComponentProcess  
[args...]
```

### Options

```
-component, --component  
    Required. Name of the component  
  
-componentProcess, --componentProcess  
    Required. Name of the component
```

## getComponents

Get a JSONArray representation of all components

### Format

```
udclient [global-args...] [global-flags...] getComponents [args...]
```

### Options

```
No options for this command.
```

## getComponentsInApplication

Get all components in an application

### Format

```
udclient [global-args...] [global-flags...] getComponentsInApplication
[args...]
```

### Options

```
-application, --application
    Required. Name of the application to get components for
```

## getEnvironment

Get a JSON representation of an environment

### Format

```
udclient [global-args...] [global-flags...] getEnvironment [args...]
```

### Options

```
-environment, --environment
    Required. Name of the environment to look up
```

## getEnvironmentsInApplication

Get all environments in an application

### Format

```
udclient [global-args...] [global-flags...]
getEnvironmentsInApplication [args...]
```

### Options

```
-application, --application
    Required. Name of the application to get environments for
```

## getMapping

Get a JSON representation of a mapping

### Format

```
udclient [global-args...] [global-flags...] getMapping [args...]
```

### Options

```
-mapping, --mapping  
    Required. ID of the mapping to look up
```

## getResource

Get a JSON representation of a resource

### Format

```
udclient [global-args...] [global-flags...] getResource [args...]
```

### Options

```
-resource, --resource  
    Required. Name of the resource to look up
```

## getResourceGroup

Get a JSON representation of a resource group

### Format

```
udclient [global-args...] [global-flags...] getResourceGroup [args...]
```

### Options

```
-group, --group  
    Required. Path of the resource group to show
```

## getResourceGroups

Get a JSONArray representation of all resource groups

**Format**

```
udclient [global-args...] [global-flags...] getResourceGroups
[args...]
```

**Options**

```
No options for this command.
```

**getResourceProperty**

Get the value of a custom property on a resource

**Format**

```
udclient [global-args...] [global-flags...] getResourceProperty
[args...]
```

**Options**

```
-resource, --resource
    Required. Name of the resource

-name, --name
    Required. Name of the property
```

**getResources**

Get a JSONArray representation of all resources

**Format**

```
udclient [global-args...] [global-flags...] getResources [args...]
```

**Options**

```
No options for this command.
```

**getResourcesInGroup**

Get a JSONArray representation of all resources in a group



**Format**

```
udclient [global-args...] [global-flags...] getResourcesInGroup
[args...]
```

**Options**

```
-group, --group
    Required. Path of the resource group
```

**getRoleForApplications**

Get a JSON representation of a role

**Format**

```
udclient [global-args...] [global-flags...] getRoleForApplications
[args...]
```

**Options**

```
-role, --role
    Required. Name of the role
```

**getRoleForComponents**

Get a JSON representation of a role

**Format**

```
udclient [global-args...] [global-flags...] getRoleForComponents
[args...]
```

**Options**

```
-role, --role
    Required. Name of the role
```

**getRoleForEnvironments**

Get a JSON representation of a role

**Format**

```
udclient [global-args...] [global-flags...] getRoleForEnvironments
[args...]
```

**Options**

```
-role, --role
    Required. Name of the role
```

**getRoleForResources**

Get a JSON representation of a role

**Format**

```
udclient [global-args...] [global-flags...] getRoleForResources
[args...]
```

**Options**

```
-role, --role
    Required. Name of the role
```

**getRoleForUI**

Get a JSON representation of a role

**Format**

```
udclient [global-args...] [global-flags...] getRoleForUI [args...]
```

**Options**

```
-role, --role
    Required. Name of the role
```

**getUser**

Get a JSON representation of a user

**Format**

```
udclient [global-args...] [global-flags...] getUser [args...]
```

**Options**

```
-user, --user
    Required. Name of the user
```

**importGroup**

Add a new group This command takes a JSON request body. Use the -t flag to view the template for the data required for this command.

**Format**

```
udclient [global-args...] [global-flags...] importGroup [args...] [-]
[filename]

-
    Read JSON input from the stdin. See command for requirements.

filename
    Read JSON input from a file with the given filename. See command
for
    requirements.
```

**Options**

```
No options for this command.
```

**importVersions**

Run the source config integration for a component This command takes a JSON request body. Use the -t flag to view the template for the data required for this command.

**Format**

```
udclient [global-args...] [global-flags...] importVersions [args...]
[-] [filename]

-
    Read JSON input from the stdin. See command for requirements.

filename
    Read JSON input from a file with the given filename. See command
for
    requirements.
```

**Options**

```
No options for this command.
```

**login**

Login for further requests

**Format**

```
udclient [global-args...] [global-flags...] login [args...]
```

**Options**

```
No options for this command.
```

**logout**

Logout

**Format**

```
udclient [global-args...] [global-flags...] logout [args...]
```

**Options**

```
No options for this command.
```

**removeActionFromRoleForApplications**

Add action to a role for applications

**Format**

```
udclient [global-args...] [global-flags...]
removeActionFromRoleForApplications [args...]
```

**Options**

```
-role, --role
    Required. Name of the role
```

```
-action, --action
    Required. Name of the action
```

## removeActionFromRoleForComponents

Add action to a role for components

### Format

```
udclient [global-args...] [global-flags...]
removeActionFromRoleForComponents [args...]
```

### Options

```
-role, --role
    Required. Name of the role

-action, --action
    Required. Name of the action
```

## removeActionFromRoleForEnvironments

Add action to a role for environments

### Format

```
udclient [global-args...] [global-flags...]
removeActionFromRoleForEnvironments [args...]
```

### Options

```
-role, --role
    Required. Name of the role

-action, --action
    Required. Name of the action
```

## removeActionFromRoleForResources

Add action to a role for resources

### Format

```
udclient [global-args...] [global-flags...]
removeActionFromRoleForResources [args...]
```

**Options**

```
-role, --role
    Required. Name of the role

-action, --action
    Required. Name of the action
```

**removeActionFromRoleForUI**

Add action to a role for the UI

**Format**

```
udclient [global-args...] [global-flags...] removeActionFromRoleForUI
[args...]
```

**Options**

```
-role, --role
    Required. Name of the role

-action, --action
    Required. Name of the action
```

**removeGroupFromRoleForApplication**

Remove a group to a role for an application

**Format**

```
udclient [global-args...] [global-flags...]
removeGroupFromRoleForApplication [args...]
```

**Options**

```
-group, --group
    Required. Name of the group

-role, --role
    Required. Name of the role

-application, --application
    Required. Name of the application
```

## removeGroupFromRoleForComponent

Remove a group to a role for a component

### Format

```
udclient [global-args...] [global-flags...]
removeGroupFromRoleForComponent [args...]
```

### Options

```
-group, --group
    Required. Name of the group

-role, --role
    Required. Name of the role

-component, --component
    Required. Name of the component
```

## removeGroupFromRoleForEnvironment

Remove a group to a role for an environment

### Format

```
udclient [global-args...] [global-flags...]
removeGroupFromRoleForEnvironment [args...]
```

### Options

```
-group, --group
    Required. Name of the group

-role, --role
    Required. Name of the role

-application, --application
    Required. Name of the application

-environment, --environment
    Required. Name of the environment
```

## removeGroupFromRoleForResource

Remove a group to a role for a resource

**Format**

```
udclient [global-args...] [global-flags...]
removeGroupFromRoleForResource [args...]
```

**Options**

```
-group, --group
    Required. Name of the group

-role, --role
    Required. Name of the role

-resource, --resource
    Required. Name of the resource
```

**removeGroupFromRoleForUI**

Remove a group to a role for the UI

**Format**

```
udclient [global-args...] [global-flags...] removeGroupFromRoleForUI
[args...]
```

**Options**

```
-group, --group
    Required. Name of the group

-role, --role
    Required. Name of the role
```

**removeResourceFromGroup**

Remove a resource from a resource group. Only works with static groups.

**Format**

```
udclient [global-args...] [global-flags...] removeResourceFromGroup
[args...]
```

**Options**

```
-resource, --resource
```



```
Required. Name of the resource to remove
```

```
-group, --group  
Required. Path of the resource group to remove from
```

## removeRoleForApplications

Create a role for applications

### Format

```
udclient [global-args...] [global-flags...] removeRoleForApplications  
[args...]
```

### Options

```
-role, --role  
Required. Name of the role
```

## removeRoleForComponents

Create a role for components

### Format

```
udclient [global-args...] [global-flags...] removeRoleForComponents  
[args...]
```

### Options

```
-role, --role  
Required. Name of the role
```

## removeRoleForEnvironments

Create a role for environments

### Format

```
udclient [global-args...] [global-flags...] removeRoleForEnvironments  
[args...]
```

### Options

```
-role, --role
    Required. Name of the role
```

## removeRoleForResources

Create a role for resources

### Format

```
udclient [global-args...] [global-flags...] removeRoleForResources
[args...]
```

### Options

```
-role, --role
    Required. Name of the role
```

## removeRoleForUI

Create a role for the UI

### Format

```
udclient [global-args...] [global-flags...] removeRoleForUI [args...]
```

### Options

```
-role, --role
    Required. Name of the role
```

## removeRoleFromResource

Remove a role from a resource.

### Format

```
udclient [global-args...] [global-flags...] removeRoleFromResource
[args...]
```

### Options

```
-resource, --resource
```

```

    Required. Name of the parent resource.

    -role, --role
        Required. Name of the new resource.

```

## removeUserFromGroup

Remove a user from a group

### Format

```

    udclient [global-args...] [global-flags...] removeUserFromGroup
    [args...]

```

### Options

```

    -user, --user
        Required. Name of the user

    -group, --group
        Required. Name of the group

```

## removeUserFromRoleForApplication

Remove a user to a role for an application

### Format

```

    udclient [global-args...] [global-flags...]
    removeUserFromRoleForApplication [args...]

```

### Options

```

    -user, --user
        Required. Name of the user

    -role, --role
        Required. Name of the role

    -application, --application
        Required. Name of the application

```

## removeUserFromRoleForComponent

Remove a user to a role for a component

**Format**

```
udclient [global-args...] [global-flags...]
removeUserFromRoleForComponent [args...]
```

**Options**

```
-user, --user
    Required. Name of the user

-role, --role
    Required. Name of the role

-component, --component
    Required. Name of the component
```

**removeUserFromRoleForEnvironment**

Remove a user to a role for an environment

**Format**

```
udclient [global-args...] [global-flags...]
removeUserFromRoleForEnvironment [args...]
```

**Options**

```
-user, --user
    Required. Name of the user

-role, --role
    Required. Name of the role

-application, --application
    Required. Name of the application

-environment, --environment
    Required. Name of the environment
```

**removeUserFromRoleForResource**

Remove a user to a role for a resource

**Format**

```
udclient [global-args...] [global-flags...]
removeUserFromRoleForResource [args...]
```

**Options**

```

-user, --user
    Required. Name of the user

-role, --role
    Required. Name of the role

-resource, --resource
    Required. Name of the resource

```

**removeUserFromRoleForUI**

Remove a user to a role for the UI

**Format**

```

udclient [global-args...] [global-flags...] removeUserFromRoleForUI
[args...]

```

**Options**

```

-user, --user
    Required. Name of the user

-role, --role
    Required. Name of the role

```

**repeatApplicationProcessRequest**

Repeat an application process request.

**Format**

```

udclient [global-args...] [global-flags...]
repeatApplicationProcessRequest [args...]

```

**Options**

```

-request, --request
    Required. ID of the application process request to repeat

```

## requestApplicationProcess

Submit an application process request to run immediately. This command takes a JSON request body. Use the `-t` flag to view the template for the data required for this command.

### Format

```

    udclient [global-args...] [global-flags...] requestApplicationProcess
    [args...] [-] [filename]

    -
      Read JSON input from the stdin. See command for requirements.

    filename
      Read JSON input from a file with the given filename. See command
for
      requirements.

```

### Options

No options for this command.

## setComponentEnvironmentProperty

Set property on component/environment mapping

### Format

```

    udclient [global-args...] [global-flags...]
    setComponentEnvironmentProperty [args...]

```

### Options

```

    -propName, --propName
      Required. Name of the property to set

    -propValue, --propValue
      Required. Value of the property to set

    -component, --component
      Required. Name of the component to look up

    -environment, --environment
      Required. Name or id of the environment to look up

    -application, --application
      Optional. Name of the application to look up

```

## setComponentProperty

Set property on component

### Format

```
udclient [global-args...] [global-flags...] setComponentProperty
[args...]
```

### Options

```
-propName, --propName
    Required. Name of the property to set

-propValue, --propValue
    Required. Value of the property to set

-component, --component
    Required. Name of the component to look up
```

## setResourceProperty

Set a custom property on a resource

### Format

```
udclient [global-args...] [global-flags...] setResourceProperty
[args...]
```

### Options

```
-resource, --resource
    Required. Name of the resource to configure

-name, --name
    Required. Name of the property

-value, --value
    Optional. New value for the property
```

## updateUser

Add a new user This command takes a JSON request body. Use the `-t` flag to view the template for the data required for this command.

### Format

```
udclient [global-args...] [global-flags...] updateUser [args...] [-]
[filename]
```

```
-      Read JSON input from the stdin. See command for requirements.  
filename  
for   Read JSON input from a file with the given filename. See command  
      requirements.
```

### Options

```
-user, --user  
    Required. Name of the user
```