

uDeploy[™] User Guide
uDeploy 4.4.0
Document Version 4.4.0.1

Contents

Chapter 1: About This Guide.....	7
Part I: Introduction.....	9
Overview.....	10
Important Concepts.....	15
Architecture and Technology.....	19
Conventions.....	21
Part II: Hands-On.....	23
Getting Started.....	24
Creating Components.....	24
hello_world Component Version.....	25
Hello World Component Process.....	29
hello_world Component Process Design.....	30
Hello World Application.....	34
Part III: Using uDeploy.....	45
Components.....	46
Creating Components.....	47
Resources.....	49
Resource Groups.....	51
Applications.....	52
Deployments.....	53
Advanced Deployments.....	53
Schedule Deployments.....	59
Work Items.....	61
Part IV: Administration.....	63
Installation.....	64
System Requirements.....	64
Download UrbanDeploy.....	66
Database Installation.....	66
Server Installation.....	68
Agent Installation.....	70
Running uDeploy.....	71
Security.....	72
Authentication.....	73
Authorization.....	76
Default Permissions.....	78
Role Configuration.....	82
User Interface Security.....	85
System Security.....	88
Settings.....	92
Licenses.....	94
Network Settings.....	96

Notification Schemes.....	98
Properties.....	100
System Settings.....	102

Part V: Reference..... 105

Plug-in Integration.....	106
Ant Plug-in.....	107
Groovy Plug-in.....	107
IIS_AppCmd Plug-in.....	107
JBOSS Plug-in.....	108
SQL/JDBC Plug-in.....	108
SQLPLUS Plug-in.....	108
Tomcat Plug-in.....	108
WebSphere Plug-in.....	109
WLDeploy Plug-in.....	110
Advanced Automation Steps.....	110
Plug-in Automation.....	110
Source Configuration Reference.....	111
AntHillPro.....	111
PVCS Version Manager.....	111
Perforce.....	112
Luntbuild.....	112
Maven.....	112
Jenkins.....	112
File System.....	112
Team Forge.....	113
Team City.....	113
Subversion.....	113
Team Foundation Server (TFS).....	113
Notifacations.....	113
Configuration.....	116
Application Configuration.....	117
Component Configuration.....	119
Environment Configuration.....	122
Inventory.....	124
CLI Reference.....	126
addActionToRoleForApplications.....	126
addActionToRoleForComponents.....	126
addActionToRoleForEnvironments.....	127
addActionToRoleForResources.....	127
addActionToRoleForUI.....	128
addComponentToApplication.....	128
addGroupToRoleForApplication.....	128
addGroupToRoleForComponent.....	129
addGroupToRoleForEnvironment.....	129
addGroupToRoleForResource.....	130
addGroupToRoleForUI.....	130
addLicense.....	130
addNameConditionToGroup.....	131
addPropertyConditionToGroup.....	131
addResourceToGroup.....	132
addRoleToResource.....	132
addRoleToResourceWithProperties.....	132
addUserToGroup.....	133
addUserToRoleForApplication.....	133

addUserToRoleForComponent.....	134
addUserToRoleForEnvironment.....	134
addUserToRoleForResource.....	135
addUserToRoleForUI.....	135
addVersionFiles.....	135
addVersionStatus.....	136
createApplication.....	136
createApplicationProcess.....	137
createComponent.....	137
createComponentProcess.....	138
createDynamicResourceGroup.....	138
createEnvironment.....	139
createGroup.....	139
createMapping.....	139
createResourceGroup.....	140
createRoleForApplications.....	140
createRoleForComponents.....	141
createRoleForEnvironments.....	141
createRoleForResources.....	141
createRoleForUI.....	142
createSubresource.....	142
createUser.....	142
createVersion.....	143
deleteGroup.....	143
deleteResourceGroup.....	143
deleteResourceProperty.....	144
deleteUser.....	144
exportGroup.....	144
getApplication.....	145
getApplicationProcess.....	145
getApplicationProcessRequestStatus.....	145
getApplications.....	146
getComponent.....	146
getComponentProcess.....	146
getComponents.....	147
getComponentsInApplication.....	147
getEnvironment.....	147
getEnvironmentsInApplication.....	148
getMapping.....	148
getResource.....	148
getResourceGroup.....	149
getResourceGroups.....	149
getResourceProperty.....	149
getResources.....	150
getResourcesInGroup.....	150
getRoleForApplications.....	150
getRoleForComponents.....	151
getRoleForEnvironments.....	151
getRoleForResources.....	151
getRoleForUI.....	152
getUser.....	152
importGroup.....	152
importVersions.....	153
login.....	153
logout.....	153
removeActionFromRoleForApplications.....	154

removeActionFromRoleForComponents.....	154
removeActionFromRoleForEnvironments.....	154
removeActionFromRoleForResources.....	155
removeActionFromRoleForUI.....	155
removeGroupFromRoleForApplication.....	156
removeGroupFromRoleForComponent.....	156
removeGroupFromRoleForEnvironment.....	156
removeGroupFromRoleForResource.....	157
removeGroupFromRoleForUI.....	157
removeResourceFromGroup.....	158
removeRoleForApplications.....	158
removeRoleForComponents.....	158
removeRoleForEnvironments.....	159
removeRoleForResources.....	159
removeRoleForUI.....	159
removeRoleForResource.....	160
removeUserFromGroup.....	160
removeUserFromRoleForApplication.....	161
removeUserFromRoleForComponent.....	161
removeUserFromRoleForEnvironment.....	161
removeUserFromRoleForResource.....	162
removeUserFromRoleForUI.....	162
repeatApplicationProcessRequest.....	163
requestApplicationProcess.....	163
setComponentEnvironmentProperty.....	163
setComponentProperty.....	164
setResourceProperty.....	164
updateUser.....	165

Chapter 1

About This Guide

uDeploy™ automates software deployment. This guide describes how to install, use, and administer uDeploy.

This guide consists of the following sections.

Introduction

- **Overview**
- **Concepts**
- **Architecture and Technology**
- **Conventions**

Hands-On

- **Introduction**
- **Hands-on**
- **Using uDeploy**
- **Administration**
- **Reference**

Using uDeploy

- Introduction
as
- Hands-on
ss
- Using uDeploy
sd
- Administration
sd
- Reference
sd

Administration

- Introduction
as
- Hands-on
ss
- Using uDeploy

sd

- Administration

sd

- Reference

sd

Reference

- Introduction

as

- Hands-on

ss

- Using uDeploy

sd

- Administration

sd

- Reference

sd

Part

I

Introduction

Topics:

- *Overview*
 - *Important Concepts*
 - *Architecture and Technology*
 - *Conventions*
-

Overview

uDeploy is an application release automation tool that enables you to dramatically reduce deployment times. With its easy to use drag-and-drop interface, you can efficiently model N-tiered applications and create transparent deployment processes for any environment—regardless of the build tools you use.

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 deployments synchronized with source control tools
- auto-trigger smoke tests
- automatic deployment roll-back
- integration with authentication systems such as LDAP
- artifact repository
- tight SCM integration
- build tool integration with such tools as Ant, and Maven

uDeploy-managed applications are usually deployed into N-tiered environments that consist of many machines, systems, and networks. Typically, the deployable artifacts for each tier are logically combined into what UrbanCode calls a *component*. Components are then combined into applications that coordinate and perform the deployment. Complex deployments can contain numerous components. Components can also remain independent of one another, which enables incremental or targeted deployments. uDeploy is flexible and works the way you work.

Of course, you can model your components as you see fit. uDeploy provides the tools to manage components for any type of environment, such as:

- QA—automated test environment
- User Acceptance Testing—business requirements testing
- Stage—final testing environment
- Production—live environment with customer access
- Disaster Recovery—production clone used for application failure

uDeploy Software Elements

- **Server.** The server provides critical services such as the user interface, component configuration tools, work flow engine, and security service, among others. See xxxx.
- **Agent.** An agent is a lightweight process that communicates with the uDeploy server. Because agents perform the actual deployment work, each machine participating in a deployment should have its own agent installed on it. When not performing a deployment, agents run in the background with minimal overhead. See xxxx.
- **Repository.** The artifact repository—CodeStation— provides secure and tamper-proof artifact storage. The repository uses content-addressable storage to maximize efficiency. It tracks artifact versions as they occur, and maintains an artifact archive. See xxxx.

Important Terms

- **Component.** Components contain the content that gets deployed, such as: application code, file system files, database updates, middle ware configurations, etc. The deployable artifacts are combined with processes that define deployments, rollbacks, or run book processes. See xxxx.
- **Resource.** A resource represents a deployment target along with its configuration and security information. While most resources are configured for a single machine/environment, they can be configured for multiple environments. Resources are created and managed through the user interface. See xxxx.
- **Application.** An application is a user-defined model that coordinates multi-component deployments. Coordination is achieved by defining the components, processes, and environments used by the deployment. See xxxx.

uDeploy Server

The uDeploy server provides critical services such as the user interface, component and application configuration tools, work flow engine, security service, and artifact repository, among others. The server also provides tools to determine whether a deployment succeeds or fails, as well as the information needed to correct the problem or roll it back.

See xxxx.

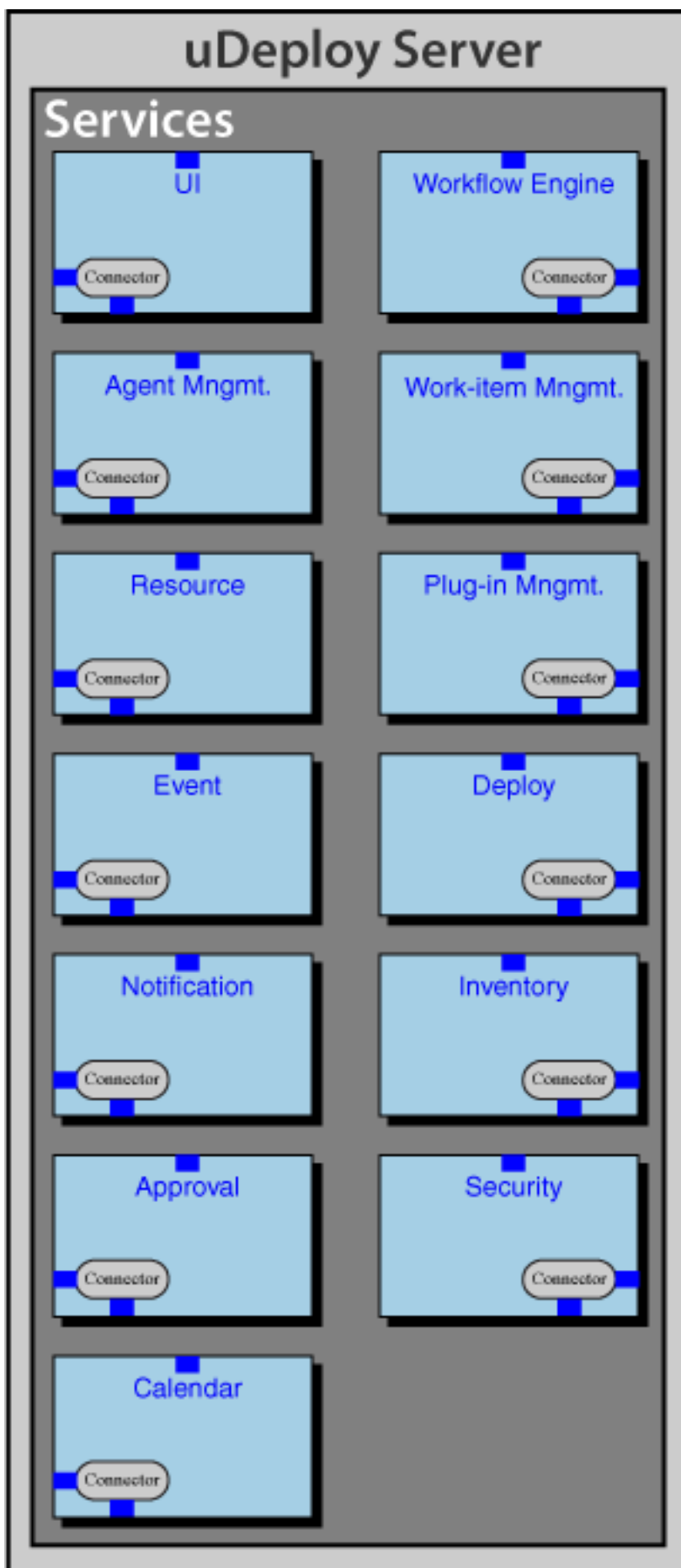


Figure 1: uDeploy Server Services

Agents

An agent is a lightweight process that communicates with the uDeploy server. Each machine that participates in a deployment usually has an agent deployed on it. Sometimes a deployment will involve a large number of agents. A deployment requires at least one agent.

Agents are unobtrusive and secure. Agent communications use SSL encryption and mutual key-based authentication. For added security, the agent process does not listen for connections on any ports: instead it opens a connection to the server.

An agent has minimal impact on the performance of the host machine. When not performing a deployment, the agent process runs in the background with almost no CPU or memory usage.

See xxxx.

Relay Servers

Many deployments involve multiple networks or data centers. uDeploy supports cross-network deployments with relay servers. Relay servers enable network-to-network communications by using only a single hole in the firewall on each network. Relay servers simplify setup, configuration, maintenance, and security. See xxxx.

Components

Components model the content—the artifacts--that get deployed, such as:

- application code
- static content
- database updates
- middleware configurations

A web application, for example, might consist of a web component containing static content served by the HTTP server (perhaps a large number of files and images); a middleware component containing an EAR file that gets deployed to a J2EE container; and a database component that contains database changes.

As a component changes, it is assigned a unique version ID. Every time a component is imported, it is assigned a new version ID. Versions can be full or incremental. A full version contains all component files; an incremental version only contains the files modified since the previous version.

The actual artifacts modeled by a component are stored in a repository, such as CodeStation or Maven. The component contains references to the artifacts.

Component Processes

As used by uDeploy, a *process* is a series of user-configured steps that instruct uDeploy's automation (or workflow) engine to perform some action. Processes are modeled with uDeploy's drag-and-drop editor and can be as complex as needed--steps can run sequentially or in parallel. Components are versioned during configuration.

Process steps are either manual or automatic. Manual steps are managed by the server's work-item list service. Automated steps are provided by *plug-ins*. Plug-ins provide integration with external systems such as middleware, databases, and other systems that receive deployments, or uDeploy interacts with in some manner. See xxxx.

For example, deploying a J2EE EAR file to WebSphere typically consists of the following automated steps: .

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

In uDeploy, each item is an automated, transparent, and easily configurable step.

Repository

The uDeploy artifact repository—CodeStation-- provides secure and tamper-proof artifact storage. The repository uses content-addressable storage to maximize efficiency. It tracks artifact versions as they change and maintains an archive for each artifact. Associations between repository files and components is built-in and automatic.

How It Is Populated

The artifact repository can be populated in a number of ways, including but not limited to:

- middleware code taken automatically from a build tool
- database updates pulled directly from a source control tool
- front-end web content retrieved from a network drop-box

Resources

A resource represents an agent along with its configuration and security information. While most agents are configured for a single machine/environment, agents can be configured for multiple environments. Resources are created and managed through the user interface.

Resource Groups

A resource group is a logical collection of resources. Resource groups are used to manage agents installed in different environments.

Proxy Resources

A *proxy resource* is a resource effected by an agent on a machine other than the one where the resource is located. If a deployment does not require direct interaction with the file system or with process management on the resource's machine, 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.

See xxxx.

Applications

An application is a user-defined process that coordinates multi-component deployments. Coordination is achieved by defining the components, resources, processes, and environments used by the deployment.

Applications run deployments.

Rollbacks

Application deployments, especially production deployments, require built-in rollback support. Applications can be configured with an automatic rollback step.

Environments

An environment is a collection of resources that host an application. Environments typically include host machines and uDeploy agents. A deployment always runs in an environment. While environments are collections of resources, resources can vary per environment.

Snapshots

A snapshot is a collection of specific components versions, usually versions that are known to work together. A snapshot represents the current state of an application in the environment.

Deployments

Deployments are run by applications. Deployments can be run manually through the uDeploy user interface, automatically by some trigger condition, or on a predetermined schedule.

Complex Deployments

Deployments supporting hundreds of agents without the need for high availability or load balancing can be performed with a single server. uDeploy supports robust deployments that require both high availability and load balancing. This requires distributing the server services onto multiple processes and machines.

Plug-in System

A plug-in provides automatic process-steps and integration with third-party applications. The plug-in system enables you to download and install any of our numerous process and integration tools. Hundreds of out-of-the-box plug-ins are available. See xxxx.

Plug-in Development

The plug-in system enables anyone to easily create their own plug-in. Plug-ins can be developed in a language of choice (including scripting languages such as perl, python, and Ruby). The UrbanCode community has a Plug-in Exchange where third-party plug-ins can be found.

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 xxxx.

Configuration Engine

TBD See xxxx.

Package Engine

TBD See xxxx.

Important Concepts

deployment is the process of moving software from various testing and preproduction stages to final deployment

easily visualize and model and automate current workflow--big picture; architectural big picture planning and design

visibility into the entire release process and all environments; logical view of your applications, components, and environments

collaboration

coordinate change tickets, approval, manual, tasks, and user-defined scripts

reporting performance metrics helps resolve conflicts

calendar

templates

developer self-service and rapid problem detection

track the configurations of QA, pre-production, and production systems in one place

tracks history too

full, end-to-end picture of application release status

coordinate the interaction of automated processes

captures, logs, and reports all activities of the release process

snapshot capture configuration state

virtualization has exacerbated challenges due to proliferation of images across data center environment

troubleshooting a deployment is just as much a part of the process as any other step
 deployments are dynamic with growing and unpredictable life-cycle
 deployments are inherently dynamic because each environment can be different and in flux
 each deployment is different and tool must be flexible
 also, might develop in one environment and deploy in another
 understand changes per environment
 points of control with optional or manual processes
 support virtual, physical, cloud-based environments
 flexible extension points

Big picture

- WHAT payload, artifact, binaries, along with configuration information. Abstracted from and independent of target multiple items delivered at the same time)
- HOW application that deploys artifacts (payload) to target and generate artifacts from the target; configure and abstracted target model
- WHERE environment-specific target to higher-criticality environment (dev to QA to production)
- VALIDATE ensure success and compliance

modeling enables straightforward config and models but generates artifacts (payloads) from targets

deployment from development to QA to preproduction to production

at deploy time payload of binaries and XML config are feed to deploy process which on the fly translates environment config info for the different target environments; mechanism then interrogates the target server and only makes changes necessary to match the abstract model--any point the process can be interrupted and rolled back

quickly troubleshoot and compare from known good version

create inventory of existing configuration

service-based solution

messaging middleware, applications servers, web servers, databases, authorization services

each component must be configured to work with the current version of every other component

model-driven configuration management

role-based access

configuration data model captures a snapshot of an application's environment with its details and interdependencies

automated discovery model existing infrastructure

UrbanDeploy is a complete, extensible platform for deployment automation and management. To accomplish this, UrbanDeploy's conceptual model allows you to easily gather, organize, and store the files that you need to deploy. This is done using what is called a Component. In addition, the Component associates the containing files with the configuration (such as versions and automation processes) necessary to move the files from environment to environment.

In turn, UrbanDeploy uses what are called Applications that are used to assemble the different Components into a group. At the Application level, UrbanDeploy allows you to map your deployments to existing environments, as well as provides a way for you to run a deployment.

Both Components and Applications utilize Processes, which can be thought of as the basic units of automation. At the Component level, the process is responsible for carrying out the tasks needed to move the components from one environment. For Applications, the Process generally consists of assembling the components and performing other, higher-level activities.

All deployments and rollbacks activities are carried out on agents, called Resources. Each resource is uniquely identified based on information you give.

Components

A component represents the artifacts that uDeploy deploys. A component wraps and combines artifacts that have some relationship to one another. For example, a component might represent all the artifacts that are deployed to a web application.

Also processes.

Components map to the existing tiers of an Application. They contain the "content" that is to be deployed, which can be a single file, images, a database, etc.: the contents of a Component are called artifacts. Components can represent configurations, not just code or infrastructure. For example, a Component may hold the application-specific configuration for WebSphere.

Components are used to import the artifacts you want to deploy. For example, an Application may consist of a WEB component containing the static content served by the HTTP server for your application; a MID component for your EAR file deployed to your J2EE container; and a DB component that contains database changes. In this case, UrbanDeploy treats the contents of each Component as grouping of artifacts.

Components are the point of contact between UrbanDeploy and your build artifacts: the Component is responsible for pulling in the artifacts that make up an Application. To keep track of changes in the artifacts, Components are assigned a Version, based on your current versioning scheme. Components also have a second role: they are responsible for deploying the artifacts that have been imported into UrbanDeploy. This is done via the Process.

Versions. Components change over time: as development continues on the Application, new builds are created and made ready for deployment. When you import the Components into UrbanDeploy, a Version is created. Versions are unique and never change. For example, Version 1.0 will include the exact same artifacts as the Component moves through the production pipeline. Every time a new build is imported into UrbanDeploy, a new Version is created. It is possible for the Component files to come from sources other than a build: they can come from some other system or from a user manually uploading files into the repository. UrbanDeploy ships with its own Artifact Repository as well as with integrations to all leading open-source and commercial repositories.

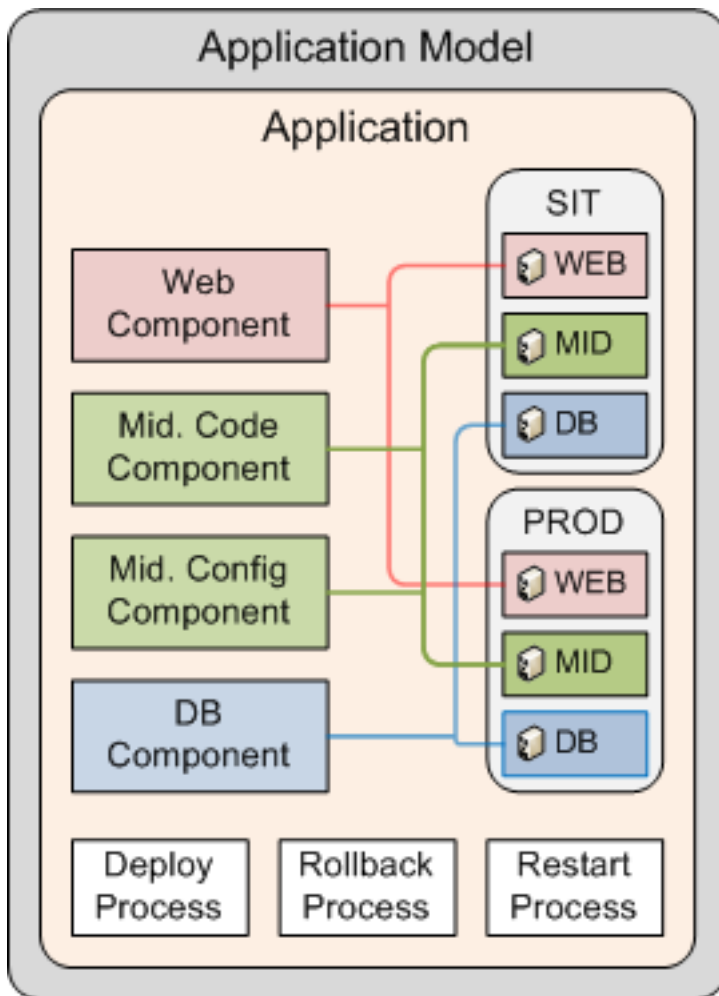
Versions come in two flavors: Full and Incremental. A Full Version contains all files for a Component, while Incremental Components contain only the files that have been modified since the previous Version was created.

Processes. Processes are composed of a series of automated Steps that are run when deploying a Component. Authoring of Processes is performed using a visual drag-n-drop editor, using standard Steps that implement functionality. The Steps within the Process are designed to replace what is typically performed manually or via a series of scripts. Deploying a J2EE EAR file to WebSphere typically consists of the following automated steps: (a.) Transfer the EAR file to the target machine; (b.) stop the WebSphere server instance; (c.) invoke wsAdmin with the location of the EAR file and appropriate deployment properties; (d.) start the WebSphere instance and verify that the deployment succeeded by hitting a specified URL. This is a plain-vanilla, out-of-box deployment Process.

In addition to running deployments, Processes can also be used to rollback an Application. The system keeps track of the history of each Versions it has deployed, so when you create a uninstall Process you typically reverse the order of a deployment.

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.

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.

Plugins

integrations are provided as Plugins. The Plugin framework allows custom integrations, the loading an unloading of integrations and updates to one integration without impacting others. Unlike other vendors, UrbanCode does not charge additional fees for each Plugin. Plugins are a technical solution to the integration problem, not a sales ploy.

Typically, an external tool exposes more than one type of behavior. For this reason, a single integration between The DevOps Platform and an external tool is made up of one or more Step Types where each Step Type knows how to invoke a single specific behavior in the external tool and/or extract specific data from the tool or inject specific data into the tool. For example, an integration with source control tool Subversion would have one Step Type that can extract source code from Subversion; a second Step Type that can apply a label to Subversion; a third Step Type that can extract a change log from Subversion; as well as other Step Types. The The DevOps Platform integration with Subversion is consists of all the Step Types that interact with Subversion.

Every mature process automation tool allows a user to enter the command line to execute and the expected exit code or tokens within the output that signal success and failure. This functionality exists within The DevOps Platform as well, but we do not consider this an integration since it does not embody any built-in relationship between the tools.

Command line integrations have a number of clear limitations. They have are restricted to command line work; wrapper scripts must be used in cases where the integration calls a web service or COM API. Command line integrations also typically have very limited abilities to capture data from the external tool. The automation tool may be able to capture a report and make it available for download, but it will be unable to parse that.

a Control Integration prompts the user for information that is required in order to interact with the external tool. Using that user supplied information, the DevOps Platform knows will how to invoke behavior in the external tool. Knowing how to determine success or failure of the invocation of the external tool is also part of the Control Integration and does not have to be configured by the user.

In a Data Integration, The DevOps Platform and the target application exchange data. The DevOps Platform retrieves data from that application's data silo, or passes information to the target application. Data integrations seem to be relatively uncommon amongst enterprise automation tools, despite being extremely powerful.

Data integrations require a database designed to store information from extracted from the externals tool via the integrations. The DevOps Platform includes a data warehouse where this type of data is stored. Data integrations help the team assemble data in a single place and boost communication in the enterprise.

Source Control Tools

Most of our source control integrations leverage the existing command line clients provided by the SCM vendor. We've found that the error messages from the command line are more easily understood than those from the APIs. Easier error messages keep things easier for everyone. Some of our Source Control Integrations utilize the APIs or direct database access for complex queries.

Creating Custom Plugins

Web Services API

Architecture and Technology

network; these are the distributed agents, or Resources. The Plug-in Integrations provide what are called Steps: which is typically a discrete series of commands, etc., that drive functionality of a third-party tool. It is at this level that

UrbanDeploy replaces scripts with hard-coded steps. The artifact repository (CodeStation) guarantees that the bits that were deployed to Production are identical to the bits that were tested and approved in a lower Environment. The package management system simplifies the deployment process: UrbanDeploy stores and provides the knowledge of how to perform a specific deployment, freeing up team members to perform other tasks. UrbanDeploy is operating-system agnostic, and will run on any system that supports Java 5. The server also runs on top of a database (MySQL or Oracle). UrbanDeploy also includes its own artifact repository, called CodeStation, to track deployable Components across Environments. The repository provides artifact security and is the basis of the traceability across environments.

Distributed Server and Agent Architecture. The server provides services such as the user interface to configure application deployments; the configuration store; the work-flow engine; security service; the artifact repository; and many other services. To run a deployment on a machine, UrbanDeploy requires an agent (also called a Resource). The agent is a small application that runs on the machine and communicates with the server. When not performing a deployment, the agent process runs in the background with almost no CPU or memory usage. Since an agent may need to perform work on an external network, UrbanDeploy provides a way to perform remote deployments and other tasks while maintaining security:

- Proxy agents. Technically, not every machine to which UrbanDeploy deploys an application or component requires an agent. Whether a target machine requires an agent or not really depends on the type of deployments that needs to be performed. If the deployment involves direct interaction with the file system (placing new or modified files on the file system or moving files or deleting them) on the target machine, then an agent on the target machine is the easiest way to accomplish this goal. Also, if the deployment involves direct interaction with process management on the target machine (such as starting or stopping processes), then an agent on the target machine provides the simplest path to accomplish this. If the deployment neither requires direct interaction with the file system (or with process management on the target machine) but instead interacts with a network service, then a proxy agent can be used. A proxy agent is located on a machine other than the target machine. When the deployment needs to interact with a service exposed on the network (such as a database, J2EE server, or another service with a network API), then the interaction with the networked service can happen from any machine on the network that has access to the networked service. For this reason, the proxy agent may reside on any machine on the network that can access the target networked service.
- Rollouts. Smaller rollouts supporting up to hundreds of agents and without the need for high availability or load balancing can be accomplished with a single UrbanDeploy server. But, UrbanDeploy architecture also allows for more robust rollouts that provide high availability and load balancing. Accomplishing this requires distributing the services provided by the UrbanDeploy server into multiple processes and multiple machines.

Plug-in Integrations. Plug-in Integrations automate common tasks. 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. The integration steps 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.

Artifact Repository (CodeStation). The artifact repository, called CodeStation, is an integral part of UrbanDeploy. It is the piece that guarantees the bits that were deployed to Production are identical to the bits that were tested and approved in a lower environment. CodeStation provides a secure and temper-proof storage for the binary artifacts that are deployed. (UrbanDeploy can integrate with an external repository, effectively bypassing CodeStation, and pull deployment artifact out of there; however, this reduces visibility and tracking.)

The artifact repository uses content addressable storage to minimize the amount of disk space utilized. CodeStation tracks Versions of files as they change and also holds a full history of all file versions for each component (a component includes the files that are deployed as well as the deployment process configuration; for more, see Components). Maximizing efficiency is important, since the artifact repository stores files that are much larger than source files. Association of files with Components is built into the system. Without any configuration, each Component gets its own area of the repository for its files. There is no chance of confusion or mix-up of files to

Components. And, each Component is mapped to a specific set of files and versions corresponding to the Component. The artifact repository has built-in security that limits the users that can access an area of the repository and what actions the users can take. Areas of the repository correspond to Components and use the security settings on the components. The artifact repository stores cryptographic hashes of all content in the repository and verifies that the stored files have not been tampered with (by recomputing the cryptographic hashes and comparing the calculated hash against the expected value that was stored when the file was added to the repository). The storage of the original, or base value, cryptographic hash takes place at the time that a file is added to the repository. The verification of the file content against the expected cryptographic hash value takes place every time the file is requested from the repository.

Configuration Engine. The Configuration Engine allows configuration changes to be made in a declarative manner. This means that the changes have to be described, but the logic used to apply the change is provided by the underlying Configuration Engine. This enables you to forego deployment and other configuration scripts. Changes are declaratively described using XML in a well-documented format defined by the Configuration Engine. Under the covers, the Configuration Engine provides the following features:

- Deploying/applying configuration changes. A configuration can be deployed to a target server or cluster. The deployment of a configuration change will examine the existing configuration on the target server/cluster and make the required changes. The required changes may be deleting configuration items (such as Data Sources) that are no longer required, creating new configuration items, or making changes to the configuration of existing items (e.g., changing the database host name in a Data Source).
- Parameterization for different environments. Configurations can be easily parameterized so that they can be used across multiple environments, even if the environments have different deployment topologies (some environments may have a single server while others may contain a cluster of servers). When the configuration is deployed to an environment, it is supplied with the parameter values that are to be used when deploying the configuration to that environment.

Databases

uDeploy stores configuration and run-time data in a relational database. At deployment time, uDeploy interrogates the database for changes and applies updates.

uDeploy supports the most commonly used relational databases: DB2, Oracle, SQL Server, and MySQL.

uDeploy provides the easy to use and configure database, Derby. This database is only recommended for evaluations.

Conventions

UrbanDeploy automates deployments of Applications -- most typically web applications. The server also provides tracking and management for every deployment it runs: providing complete visibility back to the point the Application was imported into UrbanDeploy.

To accomplish this, UrbanDeploy provides a secure, tamper-proof mechanism for fetching the different parts of an Application and storing them. These "different parts" are called Components (you may know them as application tiers, etc.), and represent the first point where UrbanDeploy interacts with the Applications it deploys. Once UrbanDeploy is up and running, the first step is to create what is called a Version. To do this, UrbanDeploy need to know where to pick up the contents of the Component (called the Artifacts). Typically, the source for a Component is located under source control or resides on a file share, etc. The Version (sometimes called Component Version or simply Component) is created by pointing UrbanDeploy to the base directory. When the configuration is complete, UrbanDeploy imports the artifacts and creates the first Version. Until removed from UrbanDeploy by an administrator, the Version never changes: the artifacts contained in the Version will be pushed, or deployed, through the various Environments as the Version is assembled, along with other Component Versions, into an Application.

After a Version is created, an automated deployment is designed. This is called a Process. Processes are composed of a series of automated Steps that are run when deploying a Component. Authoring of Processes is performed using a visual drag-n-drop editor, using standard Steps that implement functionality. The Steps within the Process are designed to replace what is typically performed manually or via a series of scripts.

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 Processes concerned with the Components and Resources necessary for deployment, etc. Applications also introduce Snapshots to manage the different versions of each Component. For most users, the Snapshot is pushed through the pipeline.

These elements make up the UrbanDeploy system. The system is designed to provide pure automation for Application deployments, as well as managing the resources they rely on. In addition, UrbanDeploy was designed with a flexible security and approvals system that can map to your requirements and processes.

With UrbanDeploy you can:

- Deploy multi-tier and service oriented applications. Many modern applications are multi-tiered and thus require a coordinated deployment of different components to corresponding tier servers. Service-oriented application deployment requires the coordination of different versions of services, which are modeled as Component in UrbanDeploy.
- Perform incremental deployments. Application deployments may be incremental, where only the changed files or configuration changes are deployed. Supporting incremental deployments allows deployments to be performed quickly and efficiently.
- Rollback applications. Application deployments, especially production deployments, require built-in rollback support. Unlike lower environments that can be rebuilt and even reimaged, production environments require less invasive and faster fixes.
- Simplify configuration changes. Application deployments typically include configuration as a significant aspect of the deployments. Whether the configuration adds new DataSources, message queues, or just changes settings in a flat file, these types of changes are typically part of application deployments. Built-in integrations, along with support for declarative configurations, mean you don't have to spend time writing and maintaining complex deployment scripts.
- Promote across multiple environments. With application deployments, multiple environments are typically involved. When deploying application changes, it is important that exactly the same bits and configuration changes get deployed to each environment while taking into account environment specific configurations.

Part II

Hands-On

Topics:

- [*Getting Started*](#)
- [*Creating Components*](#)
- [*hello_world Component Version*](#)
- [*Hello World Component Process*](#)
- [*hello_world Component Process Design*](#)
- [*Hello World Application*](#)

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 64 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 49 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 46 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 52 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

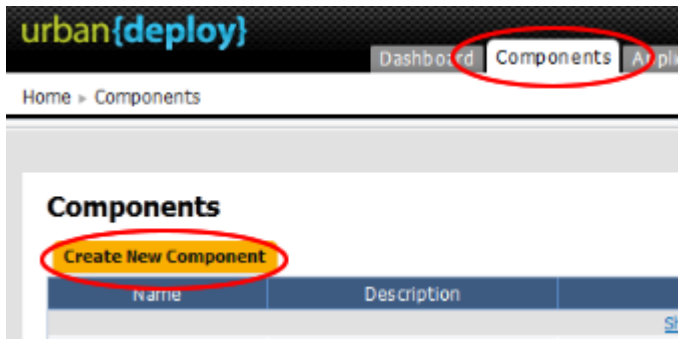
You can read the walk-through without actually performing the steps, or you can perform them as you read along. If you want to actually perform the steps as we go, do the following before starting:

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.

The screenshot shows a 'Create New Component' dialog box with the following fields and options:

- Name ***: Text input field with a red note "same for each type".
- Description**: Text input field.
- Status Plugin ***: Dropdown menu set to "Default".
- Source Config Type ***: Dropdown menu set to "AnthillPro".
- Anthill URL ***: Text input field with a red note "depends on type".
- User ***: Text input field.
- Password ***: Text input field with masked characters (dots).
- Project ***: Text input field.
- Workflow ***: Text input field.
- Status ***: Text input field.
- Import Versions Automatically**: Unchecked checkbox.
- Copy to CodeStation**: Checked checkbox with a red note "same for each type".

Buttons: Save, Cancel.

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.

Component: hello_World

Description Euro Store

History Edit Inventory Calendar Properties Templates Versions P

Versions

Version	Latest Status
1.0	Full

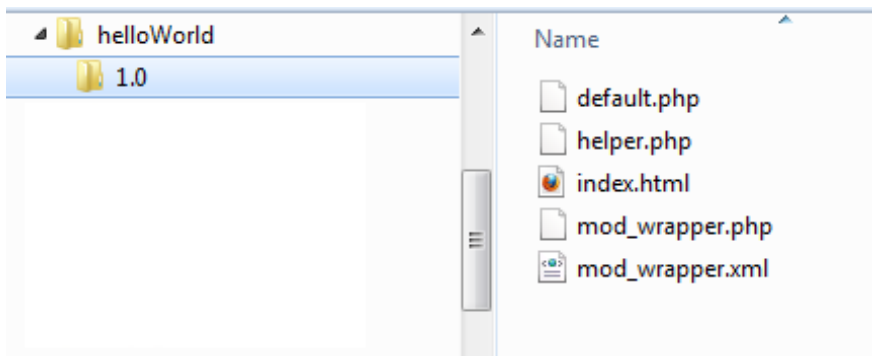
10 per page

Show Inactive Versions

[Import New Versions](#)

Figure 2: 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.

Component Artifacts

[Main](#)
[Edit](#)
[Properties](#)

Statuses

Status	Description	Created	By	Actions
No statuses have been assigned to this version. - Refresh				

[Add a Status](#)

Artifacts

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

Figure 3: 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. - Refresh	

Show Inactive Processes

[Create New Process](#)

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

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 72.

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 124.

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 4: Process Design Pane](#) on page 31. 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	
hello_worldInstall	hello_world remote install	Edit Copy

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

Show Inactive Processes

[Create New Process](#)

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

Process: hello_worldInstall

Description hello_world remote install
Version 4 of 4

Design Edit Properties Changelog

Tools

Add Steps

Available Plugin Steps

- Manual Task
- Add Inventory Status
- Remove Inventory Status
- Artifacts
 - Download Artifacts
 - Download Artifacts By Label
 - Upload Artifacts
 - Verify Local Artifacts
- Scripting
- UrbanDeploy

Start

Finish

Figure 4: 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.

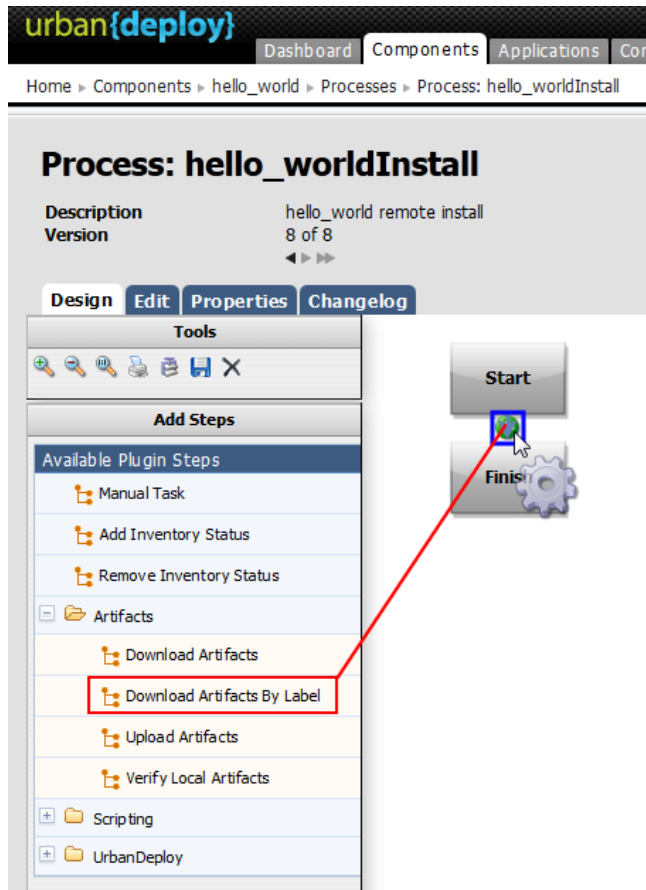



Figure 5: Adding a Step to an Anchor Point

 **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 106. 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 24.

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.

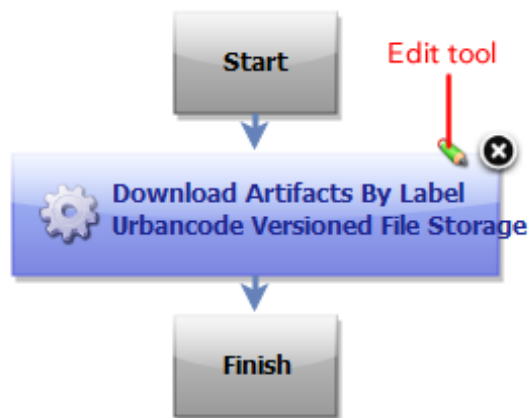


Figure 6: Edit tool

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 29).

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)

A dropdown menu for the Notification Scheme is also visible, showing 'None' and 'Default Notification Scheme'. At the bottom right of the page, the version 'UrbanDeploy 4.3.0-b1.214367' is displayed.

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 72 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 25), select *hello_world* from the **Select a Component** list box.

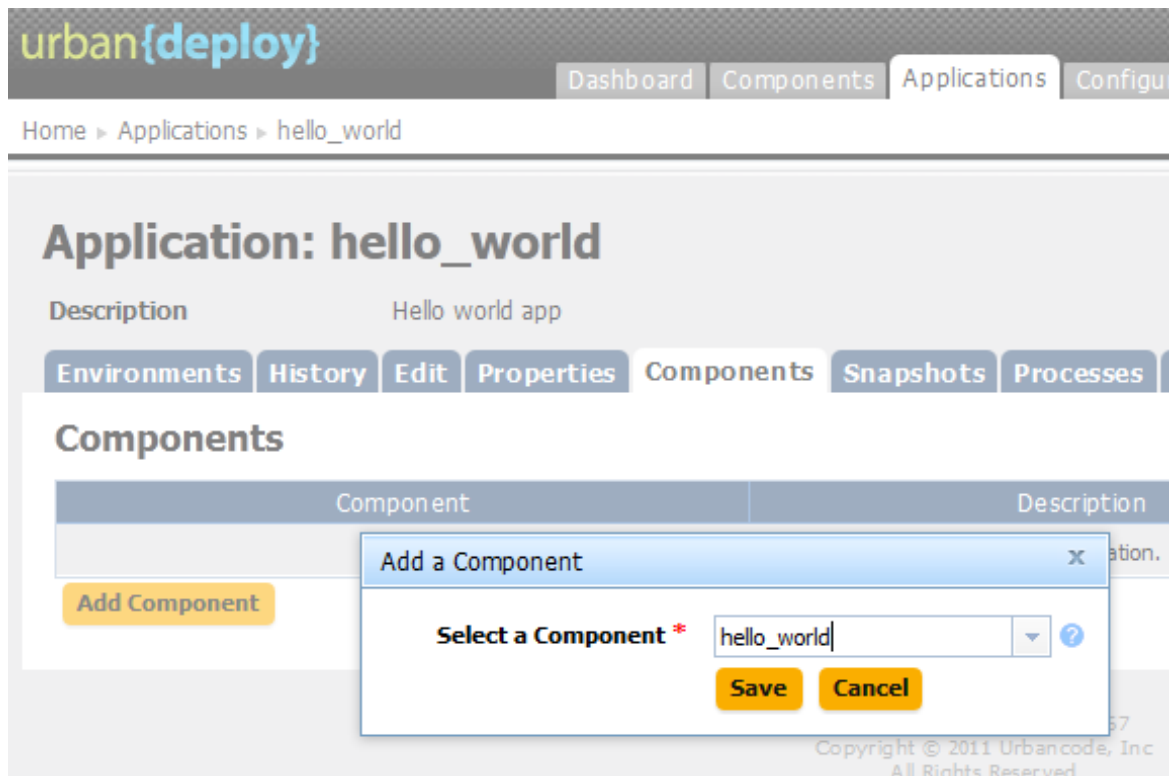


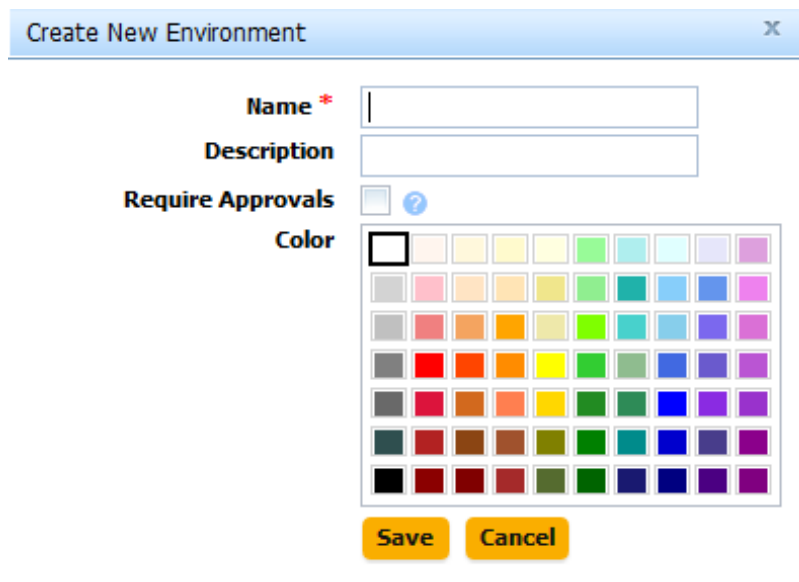
Figure 7: Adding a component to an application

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.

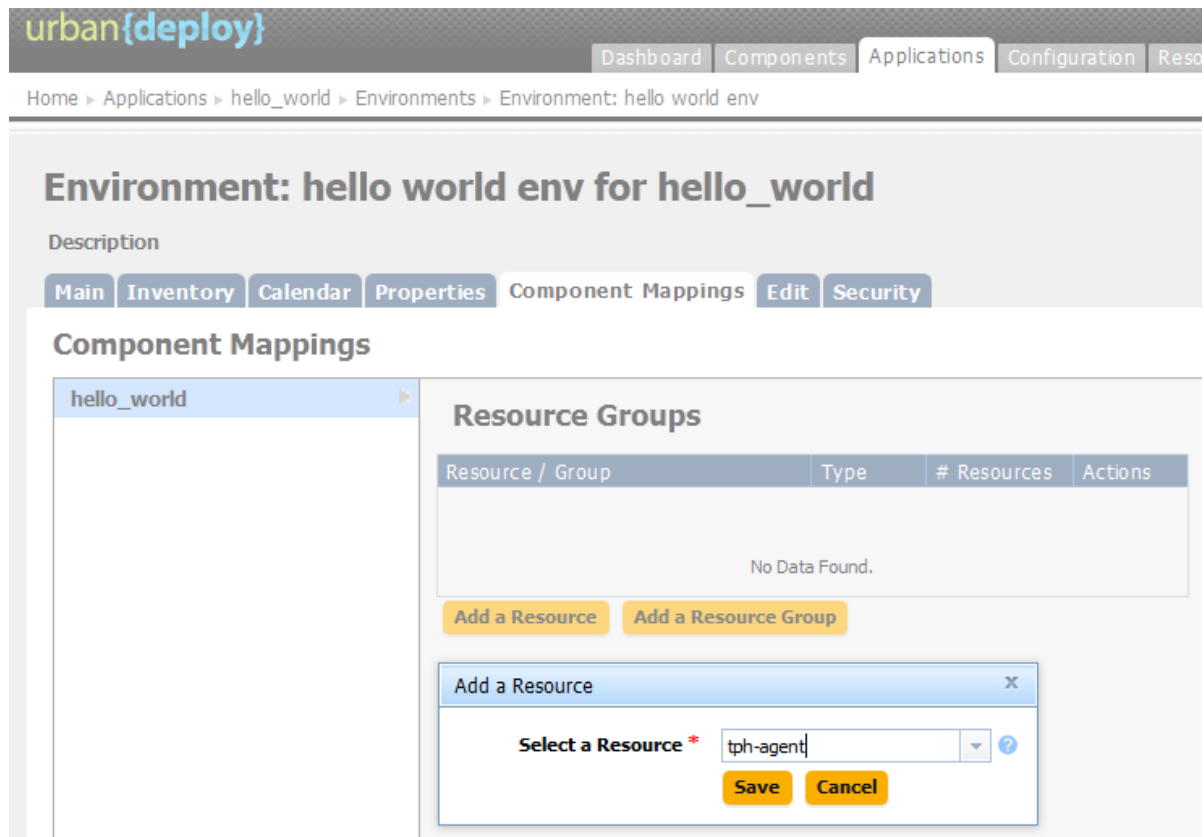


Figure 8: Adding a resource to an environment


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 49 for information about resource groups.

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 9: 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 10: 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 72.

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 124 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 29) 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.

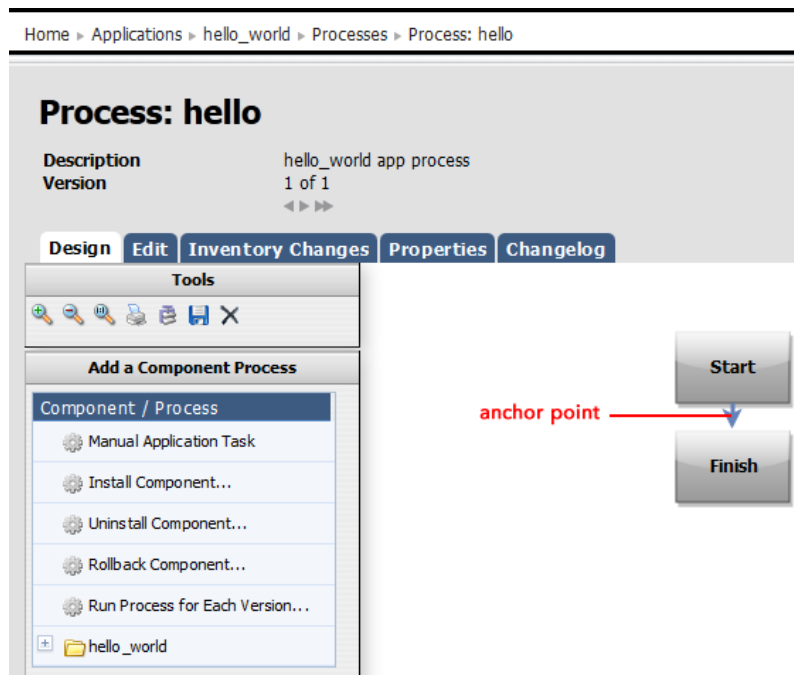


Figure 11: Process Design Pane

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.

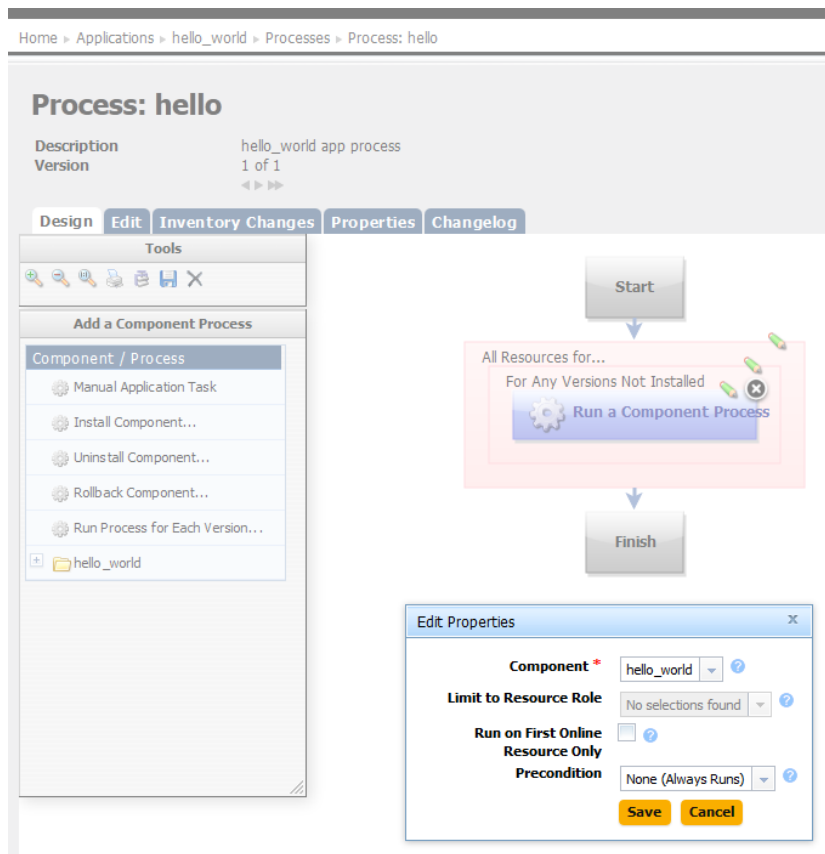


Figure 12: Edit Properties Dialog

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 52 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.

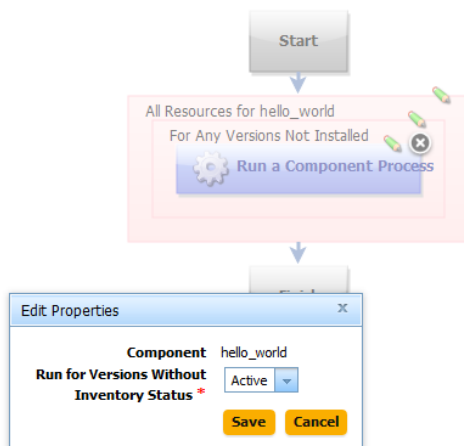
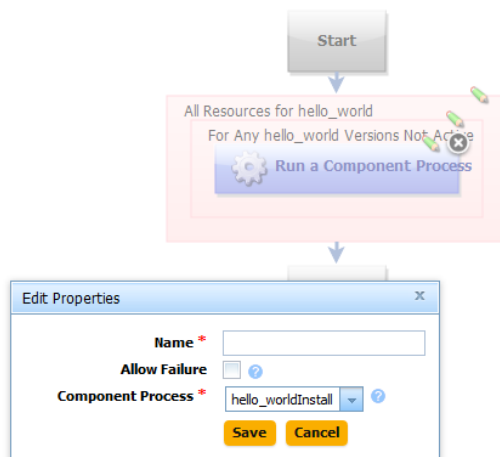


Figure 13: Run for Versions Without Inventory Status field

- Accept the default value `Active` (see [Applications](#) on page 52 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 52 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 52 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.

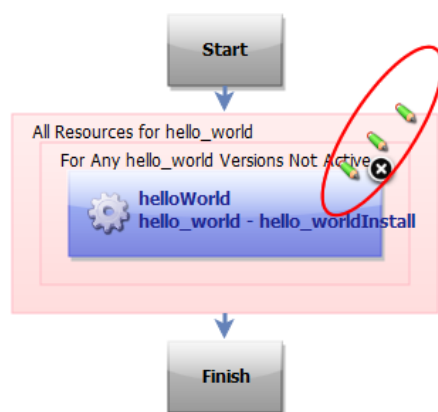


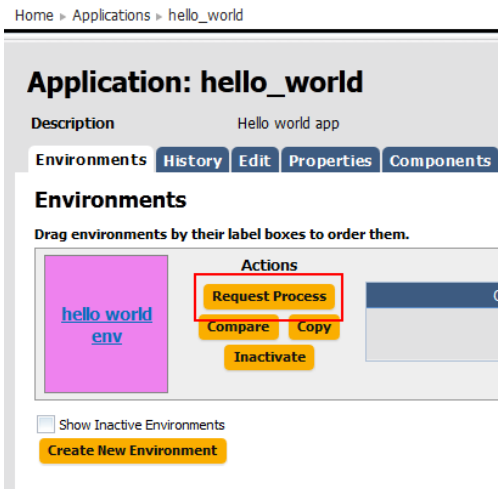
Figure 14: Nested parameters

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

Running the Application

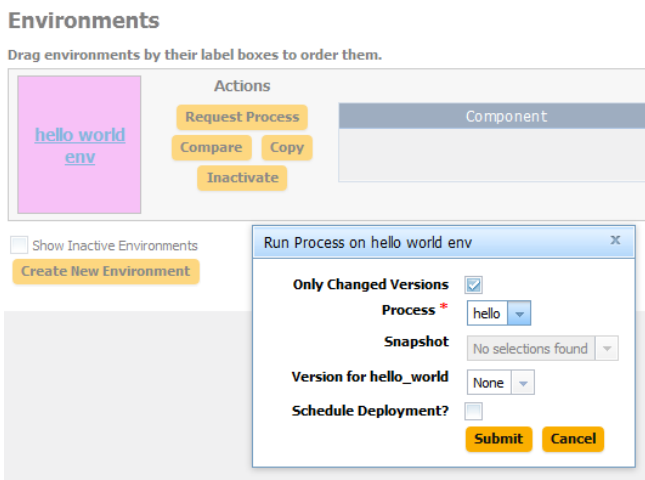
Now that the component, environment, and application are complete, you are ready to perform the deployment by running the application.

- 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 52 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 hello_world	1/9/12 5:31:45 PM	0:00:04	Success	
S	top-agent	1/9/12 5:31:45 PM	0:00:03	Success	
	helloWorld (1..4)	1/9/12 5:31:45 PM	0:00:03	Success	Details

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](#)
 - [Schedule Deployments](#)
 - [Work Items](#)
-

Components

Components map to the existing tiers of an Application. They contain the "content" that is to be deployed, which can be a single file, images, a database, etc.: the contents of a Component are called artifacts.



Note: Components can represent configurations, not just code or infrastructure. For example, a Component may hold the application-specific configuration for WebSphere.

Components are used to import the artifacts you want to deploy. For example, an Application may consist of a WEB component containing the static content served by the HTTP server for your application; a MID component for your EAR file deployed to your J2EE container; and a DB component that contains database changes. In this case, UrbanDeploy treats the contents of each Component as grouping of artifacts.

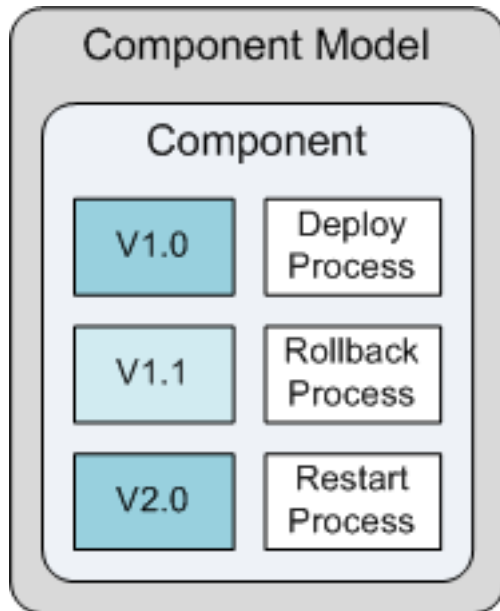


Figure 15: New Promotion Page

Processes. Processes are composed of a series of automated Steps that are run when deploying a Component. Authoring of Processes is performed using a visual drag-n-drop editor, using standard Steps that implement functionality. The Steps within the Process are designed to replace what is typically performed manually or via a series of scripts.



Note: Deploying a J2EE EAR file to WebSphere typically consists of the following automated steps: (a.) Transfer the EAR file to the target machine; (b.) stop the WebSphere server instance; (c.) invoke wsAdmin with the location of the EAR file and appropriate deployment properties; (d.) start the WebSphere instance and verify that the deployment succeeded by hitting a specified URL. This is a plain-vanilla, out-of-box deployment Process.

In addition to running deployments, Processes can also be used to rollback an Application. The system keeps track of the history of each Versions it has deployed, so when you create a uninstall Process you typically reverse the order of a deployment.



Note: Many UrbanDeploy users have found that setting up only one component at a time is the quickest way to success: it makes troubleshooting problems much easier when running the first deployment. For example, configuring a component that only contains static content can be used. Once the first component has been successfully configured and deployed, the same general workflow can be used for other, more complex components.

Creating Components

Components map to the existing tiers of an Application. They contain the "content" that is to be deployed, which can be a single file, images, a database, etc.: the contents of a Component are called artifacts.



Note: Components can represent configurations, not just code or infrastructure. For example, a Component may hold the application-specific configuration for WebSphere.

Components are used to import the artifacts you want to deploy. For example, an Application may consist of a WEB component containing the static content served by the HTTP server for your application; a MID component for your EAR file deployed to your J2EE container; and a DB component that contains database changes. In this case, UrbanDeploy treats the contents of each Component as grouping of artifacts.

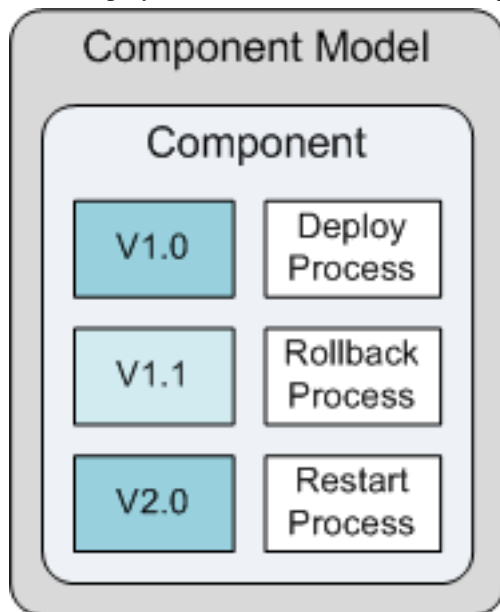


Figure 16: New Promotion Page

Components are the point of contact between uDeploy and your build artifacts: the component is responsible for pulling in the artifacts that make up an Application. To keep track of changes in the artifacts, Components are assigned a Version, based on your current versioning scheme. Components also have a second role: they are responsible for deploying the artifacts that have been imported into UrbanDeploy. This is done via the Process.

Versions

Components change over time: as development continues on the Application, new builds are created and made ready for deployment. When you import the Components into UrbanDeploy, a Version is created. Versions are unique and never change. For example, Version 1.0 will include the exact same artifacts as the Component moves through the production pipeline. Every time a new build is imported into UrbanDeploy, a new Version is created. It is possible for the Component files to come from sources other than a build: they can come from some other system or from a user manually uploading files into the repository. UrbanDeploy ships with its own Artifact Repository as well as with integrations to all leading open-source and commercial repositories.

Versions come in two flavors: Full and Incremental. A Full Version contains all files for a Component, while Incremental Components contain only the files that have been modified since the previous Version was created.

Processes

Processes are composed of a series of automated Steps that are run when deploying a Component. Authoring of Processes is performed using a visual drag-n-drop editor, using standard Steps that implement functionality. The Steps within the Process are designed to replace what is typically performed manually or via a series of scripts.



Note: Deploying a J2EE EAR file to WebSphere typically consists of the following automated steps: (a.) Transfer the EAR file to the target machine; (b.) stop the WebSphere server instance; (c.) invoke wsAdmin with the location of the EAR file and appropriate deployment properties; (d.) start the WebSphere instance and verify that the deployment succeeded by hitting a specified URL. This is a plain-vanilla, out-of-box deployment Process.

In addition to running deployments, Processes can also be used to rollback an Application. The system keeps track of the history of each Versions it has deployed, so when you create a uninstall Process you typically reverse the order of a deployment.

Component Best Practice

1. Create a Version.

Before you can run a deployment, UrbanDeploy will need to know where the artifacts (or different tiers) are stored on your network. Currently, UrbanDeploy includes integrations that allow you to fetch the artifacts from a number of different sources. When determining the Component source, you also assign a version to the component. This is accomplished in one of two ways: Either by using the "import version automatically" option or by manually creating versions. UrbanDeploy can use existing version schemes; for example, the numbers assigned by your build server or artifact management server. See Create Component Version for more.

2. Design Deployment.

The Process is designed by assembling basic units of automation, called Steps. These steps will typically replace most of your deployment scripts and/or manual processes. When designing the Process, you drag-and-drop the steps in the order that they are to be executed by UrbanDeploy. See Design Component Deployment Process.

Creating a Version

Component creation is similar for all component types. For most web application, a separate component will be created for each tier. For example, a typical 3-tier web application will have three components: database (DB), middleware code (MID), and a web component (WEB). In addition, a middleware configuration component can also be configured.



Note: When configuring a new component, keep in mind that a single component can be used by multiple projects. For example, if you have two applications that run on the same version of WebSphere, you need only fetch the WebSphere component once. Later on, when you are setting up your applications in UrbanDeploy, you are able to select the exact same version of WebSphere for each application. If a component is to be shared, the name you give should reflect this.

Component configuration differs slightly, based on the source UrbanDeploy will use as the artifact source. Before you can create a deployment process, UrbanDeploy needs to know where the artifacts (or different tiers of your application) are stored within your network. Currently, UrbanDeploy enables you to pick up the artifacts a file share; Maven, Subversion; or TeamCity.

To create a version:

1. Name the Component and give a description. Then name given here will be used when assembling the application. If the component is to be used by more than one application, the name should not be project specific. For example, the name of the component can correspond to a shared tier (WEB, etc.) that is used by different applications. For components that are application specific, a name that conveys this information can be helpful (e.g., My Applications DB).
2. Description. The optional description can be used to convey additional information about the component. For example, if the same component is used by more than one application, giving something like "Used in applications A, B, C, etc." can help others easily identify how the component is used. If you are unsure about what applications will use this component, you can leave this field blank; you can always return to the component and edit the description (go to Components > select component > Edit) at any time.
3. Source config type. Select the location where the artifacts are stored. UrbanDeploy will fetch the artifacts from their location and then store them in CodeStation (the embedded artifact repository). UrbanDeploy supports artifact fetching from a file share or local file system; Maven; Subversion; or Team City. See [Plug-in Integration](#) on page 106 for information about configuration types (plug-ins).

Resources

To run a deployment UrbanDeploy 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 UrbanDeploy, you will need to take the Environmental differences into consideration, which may require gathering some information in order to fully roll out UrbanDeploy. The Getting Started section includes some general guidelines for setting up and using UrbanDeploy.

The screenshot displays the 'Resources' pane in the UrbanDeploy web application. At the top, there is a navigation bar with the 'urban{deploy}' logo on the left and user information 'Hello admin' and a 'Logout' link on the right. Below the logo, there are several menu items: 'Components', 'Applications', 'Resources' (which is the active tab), 'Deployment Calendar', 'Work Items', and 'Settings'. The main content area is titled 'Resources' and has sub-tabs for 'Resources', 'Groups', and 'Roles'. Below these tabs is a table with the following columns: 'Name', 'Type', 'Description', 'Status', and 'Actions'. The table contains four rows of data:

Name	Type	Description	Status	Actions
Master			Online	Edit Restart
Deploy SIT			Online	Edit Restart
Deploy UAT			Online	Edit Restart
Deploy PROD			Online	Edit Restart

Below the table, there is a pagination control showing '10' items per page, '4 records', and a 'Refresh' button. At the bottom of the pane, there is a yellow button labeled 'Create New Resource'.

Figure 17: 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 type="checkbox"/> All Resource Groups	1 (0)	
PROD	0 (0)	
<input checked="" type="checkbox"/> SIT	1 (1)	
UAT	1 (1)	

Figure 18: 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
<input type="checkbox"/> All Resource Groups	1 (0)	
<input checked="" type="checkbox"/> SIT	1 (1)	
APP	1 (1)	
DB	1 (1)	
WEB	1 (1)	
UAT	1 (1)	

Figure 19: 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.

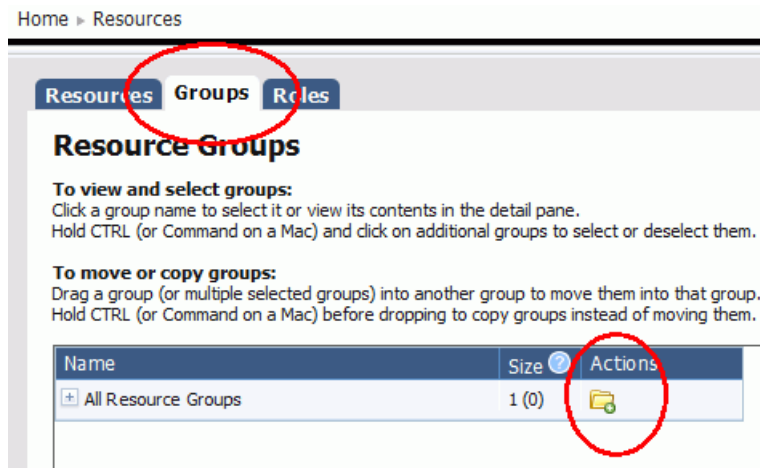


Figure 20: Action Tool

2. 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.

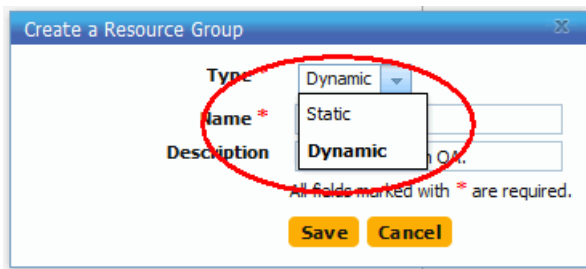


Figure 21: Create a Resource Group Dialog

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

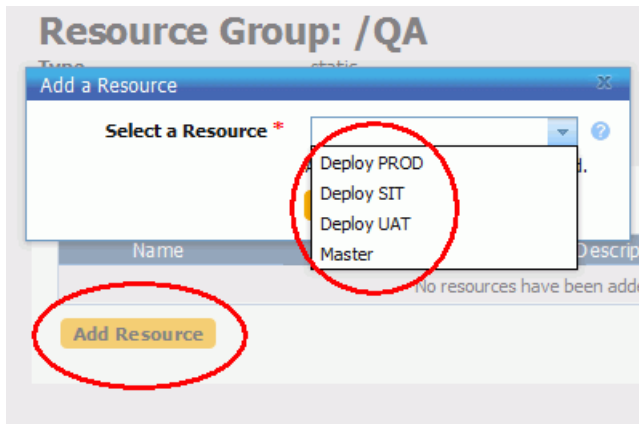


Figure 22: Add a Resource Dialog

4. 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 23: 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 for more.

Applications

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

Deployments

tbd

tbd:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

tbd

Advanced 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):

1. Download Artifacts By Label.
2. Stop Application. Based on the configuration, this step will stop your application prior to deploying it.
3. 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.
4. Deploy Application. Sends the exact Component Version to the target server and installs the artifacts in the appropriate location.
5. Start Application. Once the artifacts have been transferred, UrbanDeploy will automatically restart the application server.
6. Add Inventory Status.

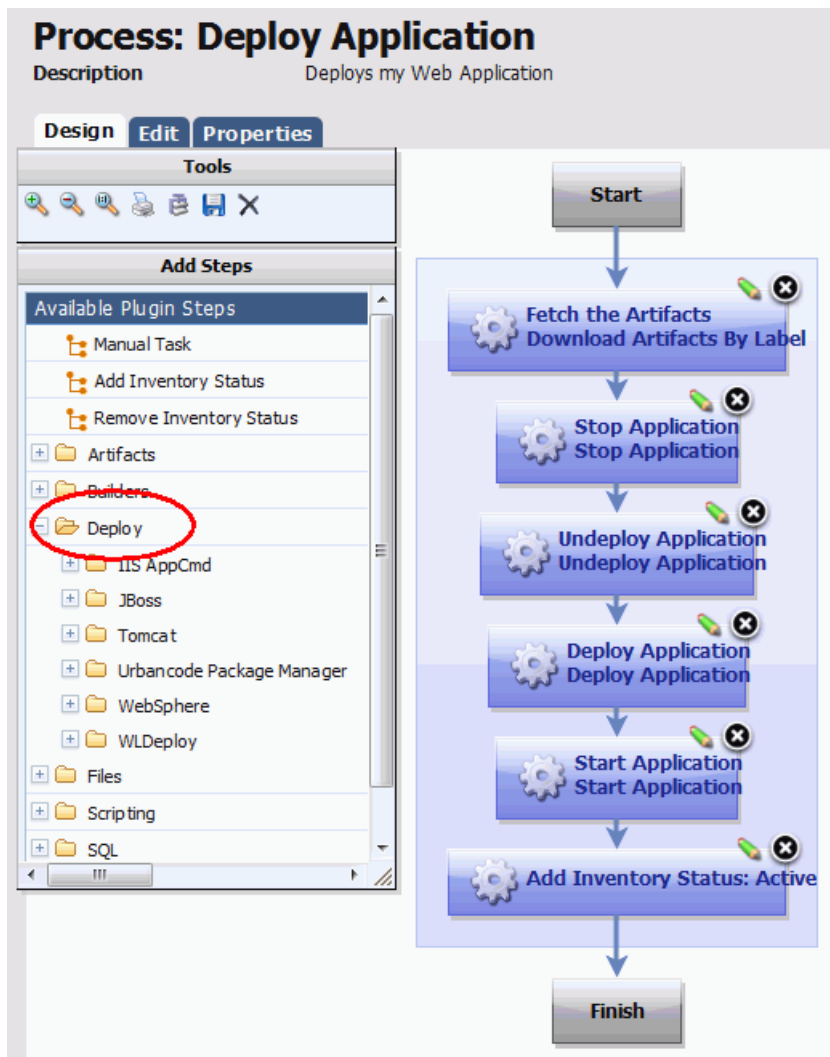


Figure 24: Deploy Application

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.

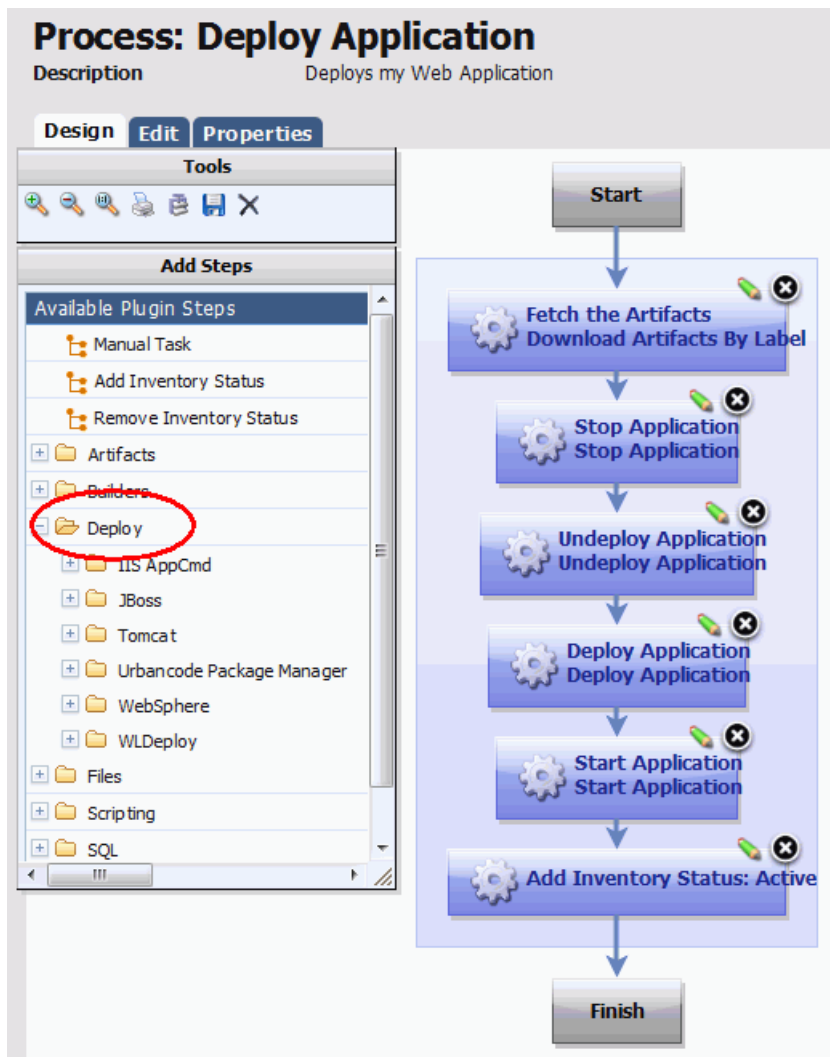
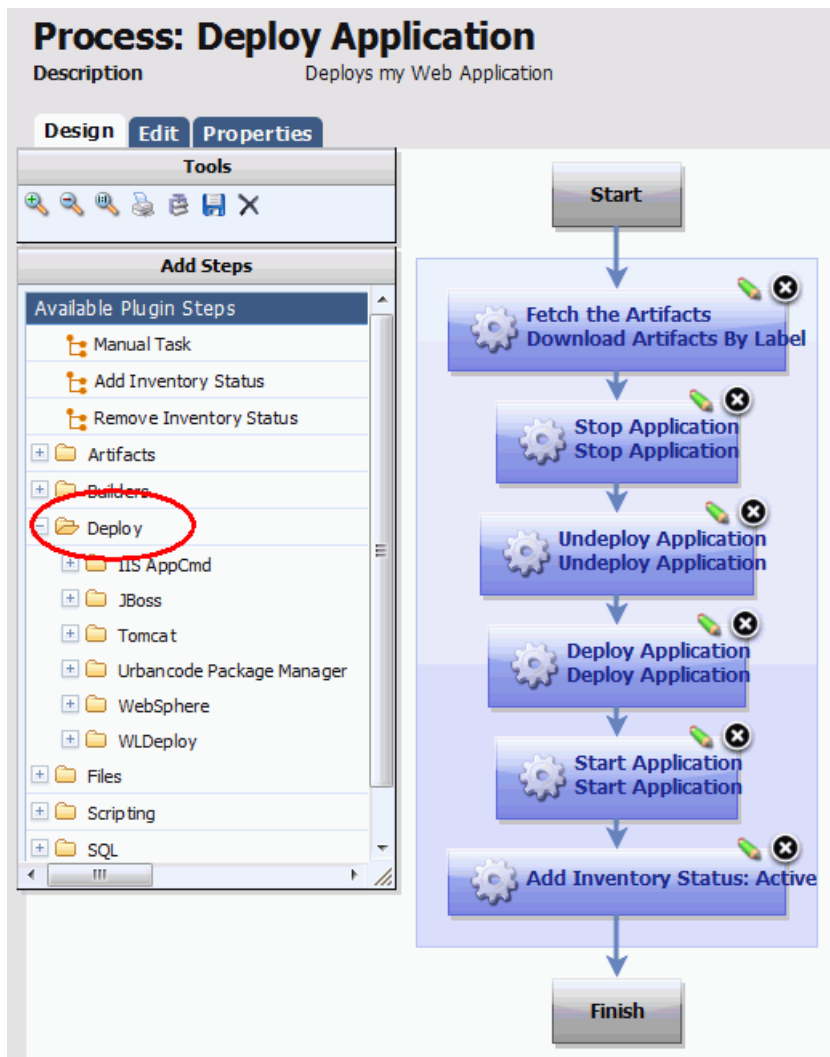


Figure 25: tbd

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 Deploy 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.

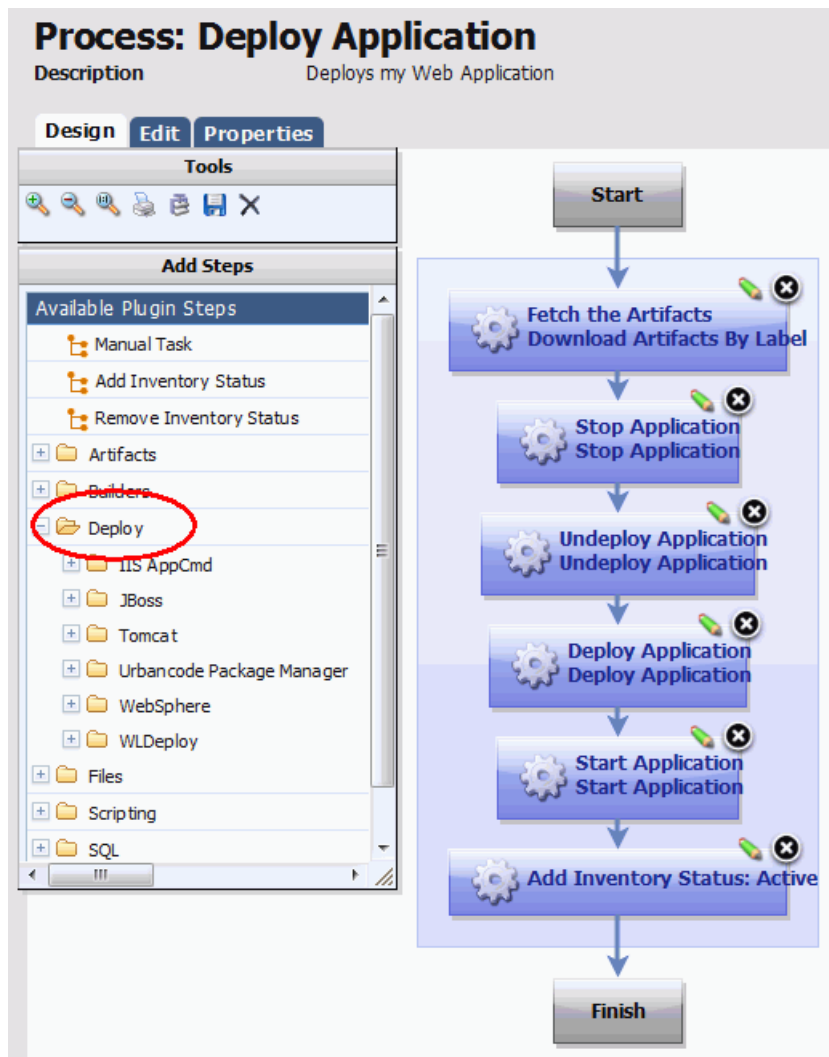


Figure 26: Edit Properties Dialog

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. UrbanDeploy 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 Plugin system has built-in steps that enable UrbanDeploy to automatically stop, undeploy, deploy, and run servers such as Tomcat, JBoss, and WebSphere. For more, see Plugins.

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, UrbanDeploy 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.

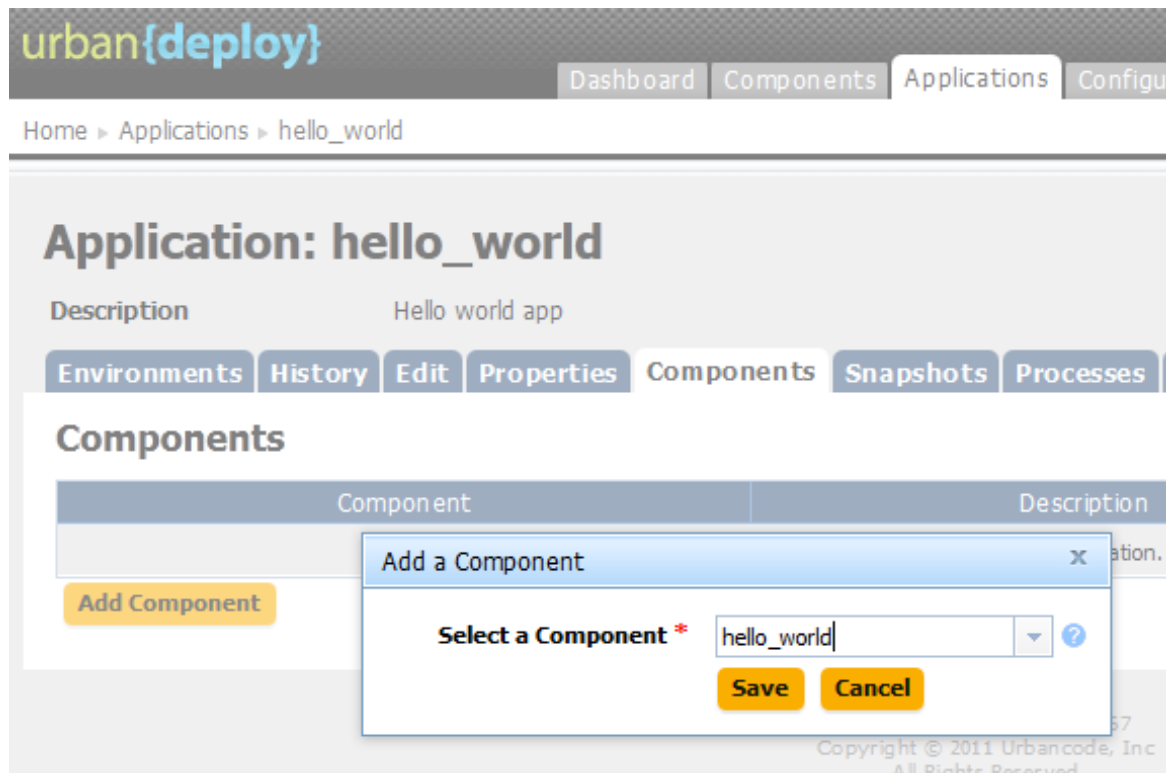


Figure 27: tbd

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.

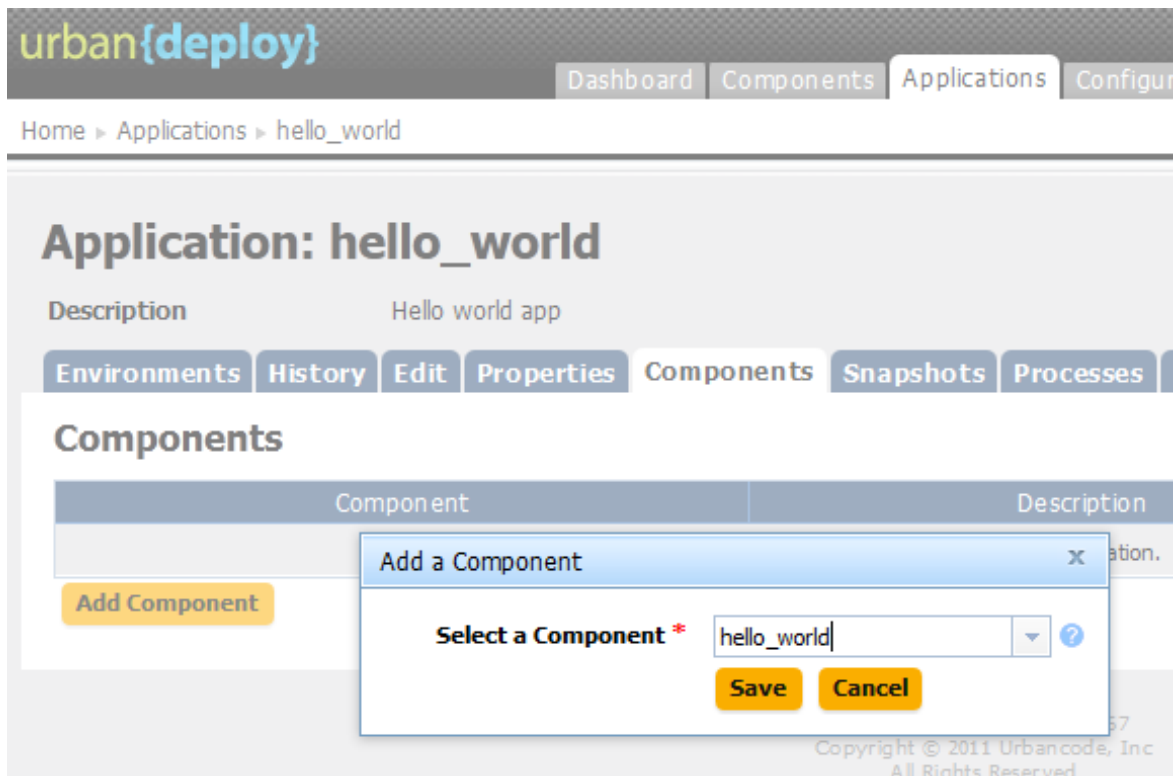


Figure 28: tbd

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. UrbanDeploy will prompt you to give the dates and times for the blackout.

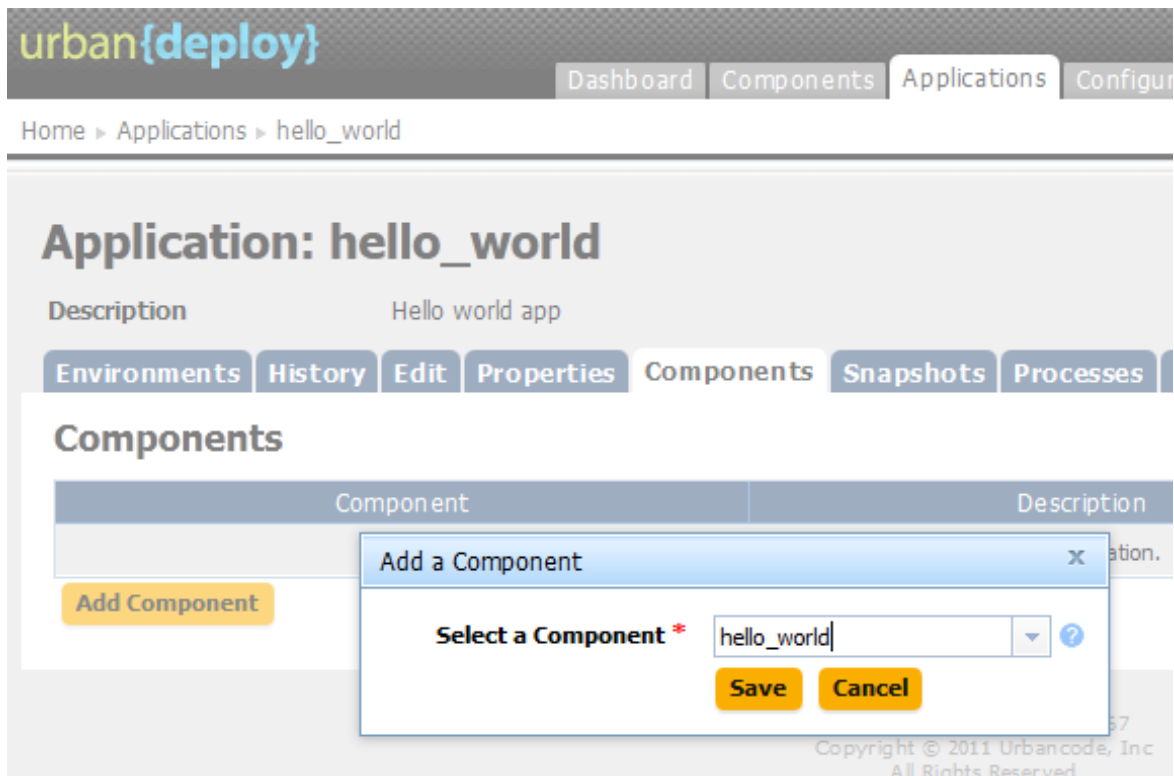


Figure 29: tbd

Work Items

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

Part IV

Administration

Topics:

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

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, DB2, or MySQL.

Installation Steps

1. Review the system requirements. See [System Requirements](#) on page 64.
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 66
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 68.
8. Finally, install at least one agent. See [Agent Installation](#) on page 70.

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, DB2, 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 68. 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 68. 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 68. 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.

Installing DB2

Before installing the uDeploy server, install a DB2 database. When you install uDeploy, you will need the DB2 connection information, and a user account with table creation privileges.

1. Obtain the DB2 JDBC driver from your vendor.
2. Copy the JDBC jar file to `uDeploy_installer_directory\lib\ext`.
3. Begin server installation, see [Server Installation](#) on page 68. When you are prompted for the database type, enter `db2`.
4. Provide the JDBC driver class uDeploy will use to connect to the database.

The default value is `com.ibm.db2.jcc.DB2Driver`.

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

Typically, it is similar to:

```
jdbc:db2://localhost:48665/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 66.

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`, `db2`, 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 66.

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`, `db2`, 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 66.

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 66.

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 66

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 UrbanDeploy, 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 UrbanDeploy 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 UrbanDeploy so all they see are the assigned Work Items.

UrbanDeploy 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 UrbanDeploy, it is recommended to use the LDAP integration.

 **Note:** If you are evaluating UrbanDeploy, 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 UrbanDeploy, 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).

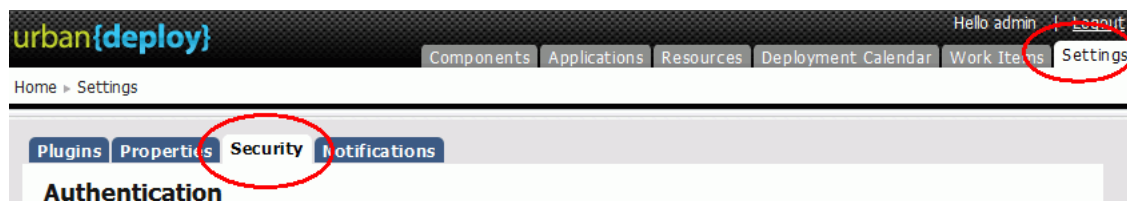


Figure 30: Security Pane

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 UrbanDeploy. There are two basic Authorization Realms in UrbanDeploy: 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 users 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, UrbanDeploy will poll all the configured Authentication Realms for matching credentials.
- **Schemas.** The Security Schemas are visual representations of the different parts of UrbanDeploy 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 UrbanDeploy, 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 UrbanDeploy. 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 31: Security Pane

Authentication

The LDAP integration enables you to map UrbanDeploy Groups and Roles to your existing infrastructure. Once the integration is configured, when a user logs into UrbanDeploy 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 UrbanDeploy to determine a User's identity as defined by LDAP.

Configuring LDAP Integration

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

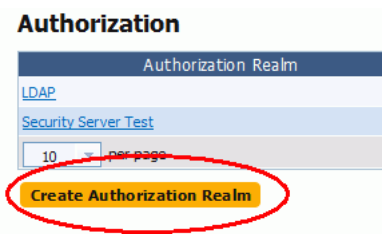


Figure 32: Create Authorization Realm

- Now, provide UrbanDeploy 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) UrbanDeploy will import your LDAP information.

 A screenshot of the "Authentication Realm: LDAP" configuration dialog. The dialog has a "Description" section and an "Edit" button. Below, there are several fields:

- Name: LDAP
- Description: (empty)
- Authorization Realm: LDAP
- Type: LDAP
- Context Factory: com.sun.jndi.ldap.LdapCtxFactor
- LDAP URL: ldap://ldap.yourcompany.com/lda
- User DN Pattern: (empty)
- User Search Base: ou=employees,dc=yourcompany
- User Search Filter: uid={0}
- Search User Subtree:
- Search Connection DN: (empty)
- Search Connection Password: (empty)
- Name Attribute: givenName
- Email Attribute: mail

 At the bottom, there are "Save" and "Cancel" buttons and a note: "All fields marked with * are required."

Figure 33: Authorization Realm Dialog

Name and description. The name you give here will be used when configuring the Authentication Realm. In 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). This 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}). This 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 UrbanDeploy search the Group Subtree for the Users. Leave blank to not search the Subtree.

- Next, you need to configure an Authentication Realm. The Authentication Realm is used to determine a user's 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, UrbanDeploy will poll all the configured Authentication Realms for matching credentials.



Figure 34: Create Authentication Realm

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

 The image shows a screenshot of the 'Authentication Realm: LDAP' configuration dialog. The dialog has a title bar with 'Authentication Realm: LDAP' and a 'Description' section. Below the title bar are 'Edit' and 'Users' buttons. The main area contains several configuration fields:

- Name ***: LDAP
- Description**: (empty)
- Authorization Realm ***: LDAP (dropdown)
- Type ***: LDAP (dropdown)
- Context Factory ***: com.sun.jndi.ldap.LdapCtxFactory (with a help icon)
- LDAP URL ***: ldap://ldap.yourcompany.com:389 (with a help icon)
- User DN Pattern**: (empty) (with a help icon)
- User Search Base**: ou=employees,dc=yourcompany, (with a help icon)
- User Search Filter**: uid={0} (with a help icon)
- Search User Subtree**: (with a help icon)
- Search Connection DN**: (empty) (with a help icon)
- Search Connection Password**: (empty) (with a help icon)
- Name Attribute**: givenName (with a help icon)
- Email Attribute**: mail (with a help icon)

 At the bottom, there is a note: 'All fields marked with * are required.' and two buttons: 'Save' and 'Cancel'.

Figure 35: Authentication Realm Dialog

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 UrbanDeploy 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 UrbanDeploy 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 UrbanDeploy using their LDAP credentials, they will be listed on the Authentication Realm User tab. Since UrbanDeploy 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 UrbanDeploy, 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 UrbanDeploy using LDAP credentials, UrbanDeploy 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 UrbanDeploy, they were automatically added to the LDAP Default group. If a user logs on to UrbanDeploy and they are part of a mapping that is not currently associated with a Group, UrbanDeploy will create a new Group based on the information fetched from LDAP. Conversely, if a user logs onto UrbanDeploy and their LDAP credentials map to an existing Group, they will be automatically added to that Group.

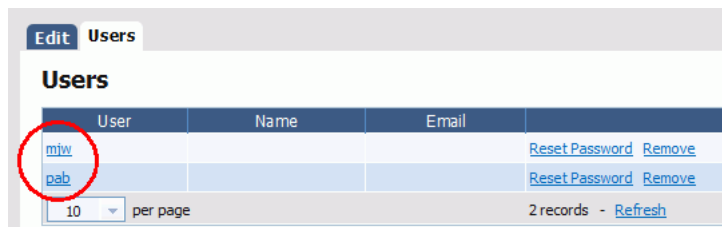


Figure 36: Edit Users

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.

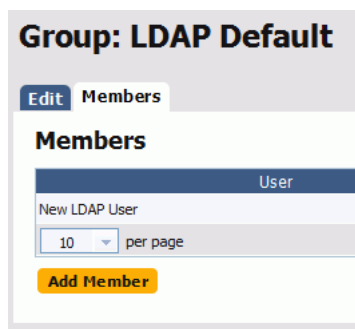



Figure 37: Group Dialog

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.

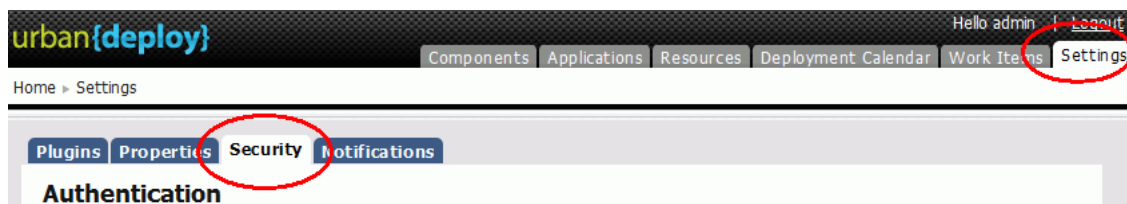


Figure 38: tbd

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, UrbanDeploy will poll all the configured Authentication Realms for matching credentials.

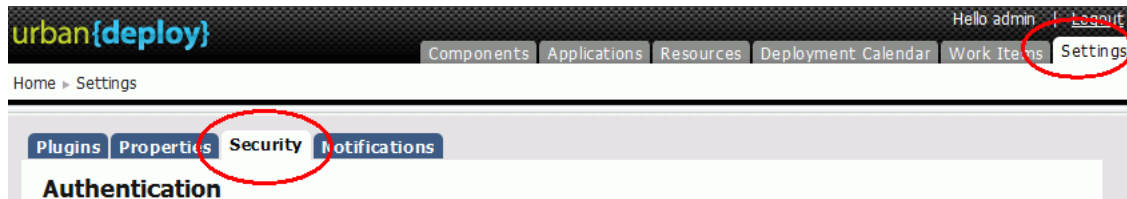


Figure 39: tbd

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

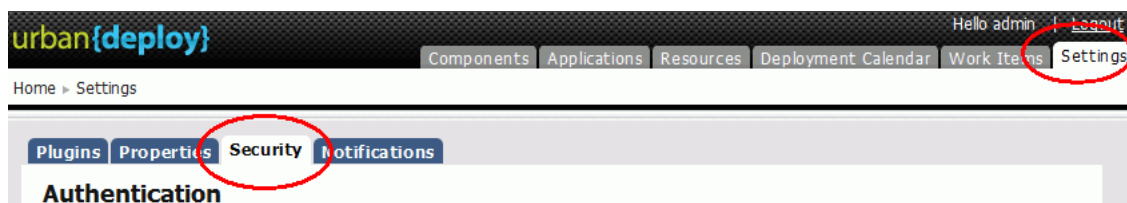


Figure 40: tbd

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 UrbanDeploy. Once the Group container is created, select it from the list and then manually enter the new User.

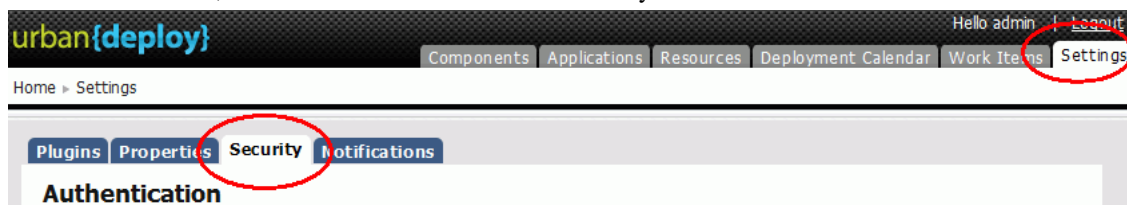


Figure 41: tbd

4. Next, create a new Role. The purpose of the Role is to assign permission that allows Users with that Role to use UrbanDeploy. For example, if you are setting up a new user that must access every page in UrbanDeploy, 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.

Figure 42: 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 UrbanDeploy, you will not need to manually add them to the resources they need access to.

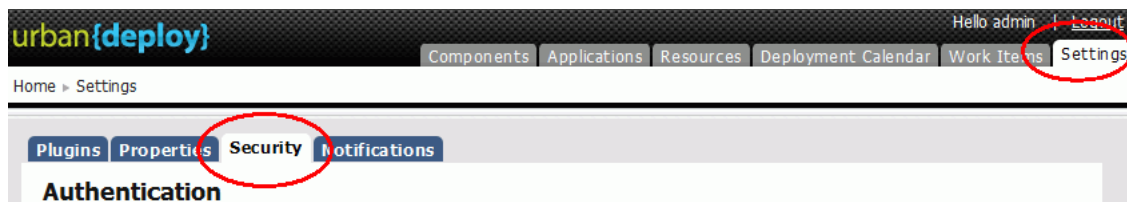


Figure 43: tbd

Default Permissions

The LDAP integration enables you to map UrbanDeploy Groups and Roles to your existing infrastructure. Once the integration is configured, when a user logs into UrbanDeploy 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 UrbanDeploy to determine a User's identity as defined by LDAP.

Configuring LDAP Integration

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

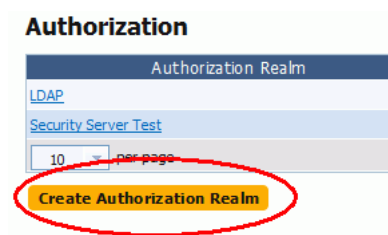


Figure 44: Create Authorization Realm

- Now, provide UrbanDeploy 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) UrbanDeploy 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 45: Authorization Realm Dialog

Name and description. The name you give here will be used when configuring the Authentication Realm. The description is optional.

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). This 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}). This 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 UrbanDeploy search the Group Subtree for the Users. Leave blank to not search the Subtree.

- Next, you need to configure an Authentication Realm. The Authentication Realm is used to determine a user's 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 User attempts to log in, UrbanDeploy will poll all the configured Authentication Realms for matching credentials.



Figure 46: Create Authentication Realm

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

 A screenshot of the 'Authentication Realm: LDAP' configuration dialog. The dialog has a title bar with 'Edit' and 'Users' buttons. Below the title bar, there is a 'Description' section. The main area contains several fields:

- Name * (text input: LDAP)
- Description (text input)
- Authorization Realm * (dropdown: LDAP)
- Type * (dropdown: LDAP)
- Context Factory * (text input: com.sun.jndi.ldap.LdapCtxFactory)
- LDAP URL * (text input: ldap://ldap.yourcompany.com/lda)
- User DN Pattern (text input)
- User Search Base (text input: ou=employees,dc=yourcompany)
- User Search Filter (text input: uid={0})
- Search User Subtree (checkbox: checked)
- Search Connection DN (text input)
- Search Connection Password (text input)
- Name Attribute (text input: givenName)
- Email Attribute (text input: mail)

 At the bottom, there is a note: 'All fields marked with * are required.' and two buttons: 'Save' and 'Cancel'.

Figure 47: Authentication Realm Dialog

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 UrbanDeploy 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 UrbanDeploy 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 UrbanDeploy using their LDAP credentials, they will be listed on the Authentication Realm User tab. Since UrbanDeploy 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 UrbanDeploy, 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 UrbanDeploy using LDAP credentials, UrbanDeploy 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 UrbanDeploy, they were automatically added to the LDAP Default group. If a user logs on to UrbanDeploy and they are part of a mapping that is not currently associated with a Group, UrbanDeploy will create a new Group based on the information fetched from LDAP. Conversely, if a user logs onto UrbanDeploy and their LDAP credentials map to an existing Group, they will be automatically added to that Group.

User	Name	Email	
miw			Reset Password Remove
pab			Reset Password Remove

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

Figure 48: 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.

Group: LDAP Default

[Edit](#) [Members](#)

Members

User
New LDAP User

10 per page

[Add Member](#)

Figure 49: Group Dialog

Role Configuration

The LDAP integration enables you to map UrbanDeploy Groups and Roles to your existing infrastructure. Once the integration is configured, when a user logs into UrbanDeploy 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 UrbanDeploy to determine a User's identity as defined by LDAP.

Configuring LDAP Integration

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

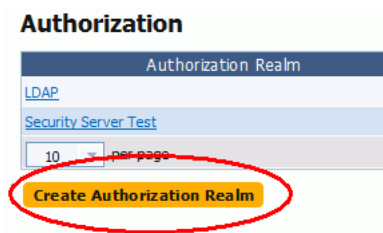


Figure 50: Create Authorization Realm

2. Now, provide UrbanDeploy 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) UrbanDeploy will import your LDAP information.

Figure 51: Authorization Realm Dialog

Name and description. The name you give here will be used when configuring the Authentication Realm. 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). This 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}). This 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 UrbanDeploy 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 user's 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 User attempts to log in, UrbanDeploy will poll all the configured Authentication Realms for matching credentials.



Figure 52: Create Authentication Realm

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

 A screenshot of the 'Authentication Realm: LDAP' configuration dialog. The title is 'Authentication Realm: LDAP' and the subtitle is 'Description'. There are 'Edit' and 'Users' tabs. The form contains the following fields:

- Name *: LDAP
- Description
- Authorization Realm *: LDAP
- Type *: LDAP
- Context Factory *: com.sun.jndi.ldap.LdapCtxFactor
- 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

 At the bottom, there is a note: 'All fields marked with * are required.' and two buttons: 'Save' and 'Cancel'.

Figure 53: Authentication Realm Dialog

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

Authorization Realm. Select the Realm 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 UrbanDeploy 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 UrbanDeploy 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 UrbanDeploy using their LDAP credentials, they will be listed on the Authentication Realm User tab. Since UrbanDeploy 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 UrbanDeploy, 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 UrbanDeploy using LDAP credentials, UrbanDeploy 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 UrbanDeploy, they were automatically added to the LDAP Default group. If a user logs on to UrbanDeploy and they are part of a mapping that is not currently associated with a Group, UrbanDeploy will create a new Group based on the information fetched from LDAP. Conversely, if a user logs onto UrbanDeploy and their LDAP credentials map to an existing Group, they will be automatically added to that Group.



User	Name	Email	
miw			Reset Password Remove
pab			Reset Password Remove

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

Figure 54: 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.

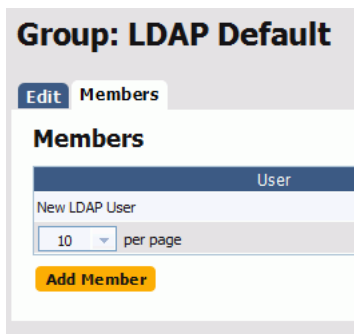


Figure 55: Group Dialog

User Interface Security

The LDAP integration enables you to map UrbanDeploy Groups and Roles to your existing infrastructure. Once the integration is configured, when a user logs into UrbanDeploy 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 UrbanDeploy to determine a User's identity as defined by LDAP.

Configuring LDAP Integration

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



Figure 56: Create Authorization Realm

2. Now, provide UrbanDeploy 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) UrbanDeploy will import your LDAP information.

Figure 57: Authorization Realm Dialog

Name and description. The name you give here will be used when configuring the Authentication Realm. The description is optional.

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). This 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}). This 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 UrbanDeploy search the Group Subtree for the Users. Leave blank to not search the Subtree.

- Next, you need to configure an Authentication Realm. The Authentication Realm is used to determine a user's 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 user attempts to log in, UrbanDeploy will poll all the configured Authentication Realms for matching credentials.

Figure 58: Create Authentication Realm

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

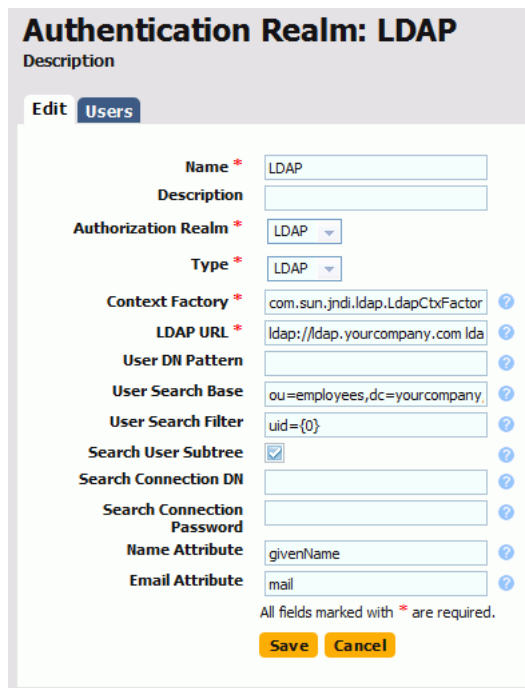


Figure 59: Authentication Realm Dialog

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 UrbanDeploy 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 UrbanDeploy 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 UrbanDeploy using their LDAP credentials, they will be listed on the Authentication Realm User tab. Since UrbanDeploy 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 UrbanDeploy, 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 UrbanDeploy using LDAP credentials, UrbanDeploy 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 UrbanDeploy, they were automatically added to the LDAP Default group. If a user logs on to UrbanDeploy and they are part of a mapping that is not currently associated with a Group, UrbanDeploy will create a new Group based on the information fetched from LDAP. Conversely, if a user logs onto UrbanDeploy and their LDAP credentials map to an existing Group, they will be automatically added to that Group.

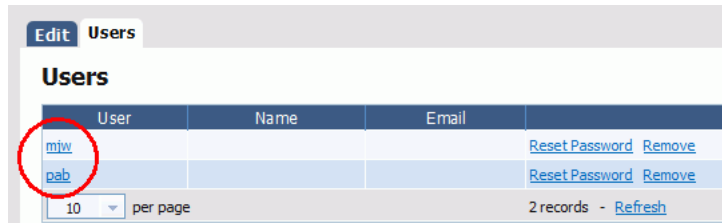


Figure 60: Edit Users

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.

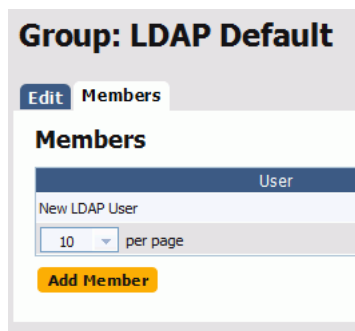


Figure 61: Group Dialog

System Security

The LDAP integration enables you to map UrbanDeploy Groups and Roles to your existing infrastructure. Once the integration is configured, when a user logs into UrbanDeploy 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 UrbanDeploy to determine a User's identity as defined by LDAP.

Configuring LDAP Integration

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

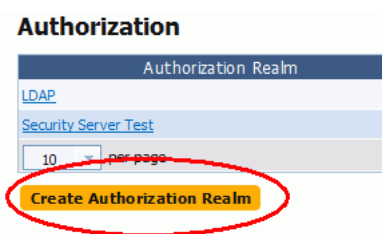


Figure 62: Create Authorization Realm

- Now, provide UrbanDeploy 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) UrbanDeploy will import your LDAP information.

 A screenshot of the "Authentication Realm: LDAP" configuration dialog. The dialog has a "Description" section and an "Edit" button. Below that, there are several fields:

- Name: LDAP
- Description: (empty)
- Authorization Realm: LDAP
- Type: LDAP
- Context Factory: com.sun.jndi.ldap.LdapCtxFactor
- LDAP URL: ldap://ldap.yourcompany.com/lda
- User DN Pattern: (empty)
- User Search Base: ou=employees,dc=yourcompany
- User Search Filter: uid={0}
- Search User Subtree:
- Search Connection DN: (empty)
- Search Connection Password: (empty)
- Name Attribute: givenName
- Email Attribute: mail

 At the bottom, there is a note: "All fields marked with * are required." and two buttons: "Save" and "Cancel".

Figure 63: Authorization Realm Dialog

Name and description. The name you give here will be used when configuring the Authentication Realm. In 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). This 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}). This 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 UrbanDeploy search the Group Subtree for the Users. Leave blank to not search the Subtree.

- Next, you need to configure an Authentication Realm. The Authentication Realm is used to determine a user's 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, UrbanDeploy will poll all the configured Authentication Realms for matching credentials.

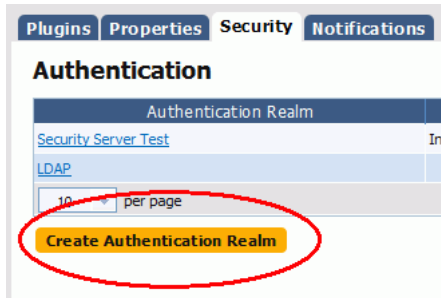


Figure 64: Create Authentication Realm

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

 A screenshot of the 'Authentication Realm: LDAP' configuration dialog. The dialog has a title bar with 'Authentication Realm: LDAP' and a 'Description' section. Below that are 'Edit' and 'Users' buttons. The main area contains several fields:

- Name * (text input: LDAP)
- Description (text input)
- Authorization Realm * (dropdown: LDAP)
- Type * (dropdown: LDAP)
- Context Factory * (text input: com.sun.jndi.ldap.LdapCtxFactory)
- LDAP URL * (text input: ldap://ldap.yourcompany.com/lda)
- User DN Pattern (text input)
- User Search Base (text input: ou=employees,dc=yourcompany)
- User Search Filter (text input: uid={0})
- Search User Subtree (checkbox: checked)
- Search Connection DN (text input)
- Search Connection Password (text input)
- Name Attribute (text input: givenName)
- Email Attribute (text input: mail)

 At the bottom, there is a note: 'All fields marked with * are required.' and two buttons: 'Save' and 'Cancel'.

Figure 65: Authentication Realm Dialog

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 UrbanDeploy 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 UrbanDeploy 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 UrbanDeploy using their LDAP credentials, they will be listed on the Authentication Realm User tab. Since UrbanDeploy 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 UrbanDeploy, 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 UrbanDeploy using LDAP credentials, UrbanDeploy 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 UrbanDeploy, they were automatically added to the LDAP Default group. If a user logs on to UrbanDeploy and they are part of a mapping that is not currently associated with a Group, UrbanDeploy will create a new Group based on the information fetched from LDAP. Conversely, if a user logs onto UrbanDeploy and their LDAP credentials map to an existing Group, they will be automatically added to that Group.

User	Name	Email	
miw			Reset Password Remove
pab			Reset Password Remove

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

Figure 66: 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.

Group: LDAP Default

[Edit](#) **Members**

Members

User
New LDAP User

10 per page

[Add Member](#)

Figure 67: Group Dialog

Settings

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, UrbanDeploy 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.

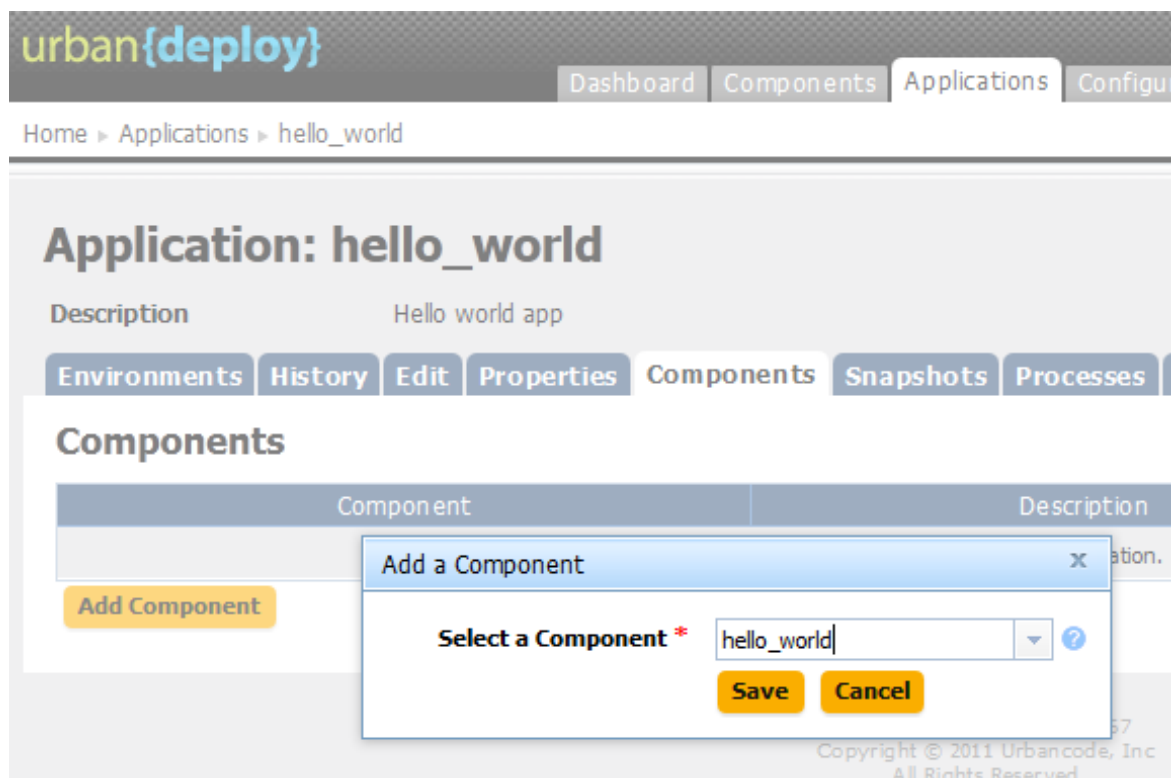


Figure 68: tbd

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.

The screenshot displays the UrbanDeploy web interface. At the top, the logo 'urban{deploy}' is visible, followed by navigation tabs: 'Dashboard', 'Components', 'Applications', and 'Configur'. Below the logo, the breadcrumb path is 'Home > Applications > hello_world'. The main heading is 'Application: hello_world', with a sub-heading 'Description Hello world app'. A row of tabs includes 'Environments', 'History', 'Edit', 'Properties', 'Components', 'Snapshots', and 'Processes'. The 'Components' tab is active, showing a table with columns 'Component' and 'Description'. A yellow 'Add Component' button is present. A modal dialog titled 'Add a Component' is open, featuring a 'Select a Component *' label, a dropdown menu with 'hello_world' selected, and 'Save' and 'Cancel' buttons. A copyright notice at the bottom right reads 'Copyright © 2011 Urbancode, Inc. All Rights Reserved'.

Figure 69: tbd

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. UrbanDeploy will prompt you to give the dates and times for the blackout.

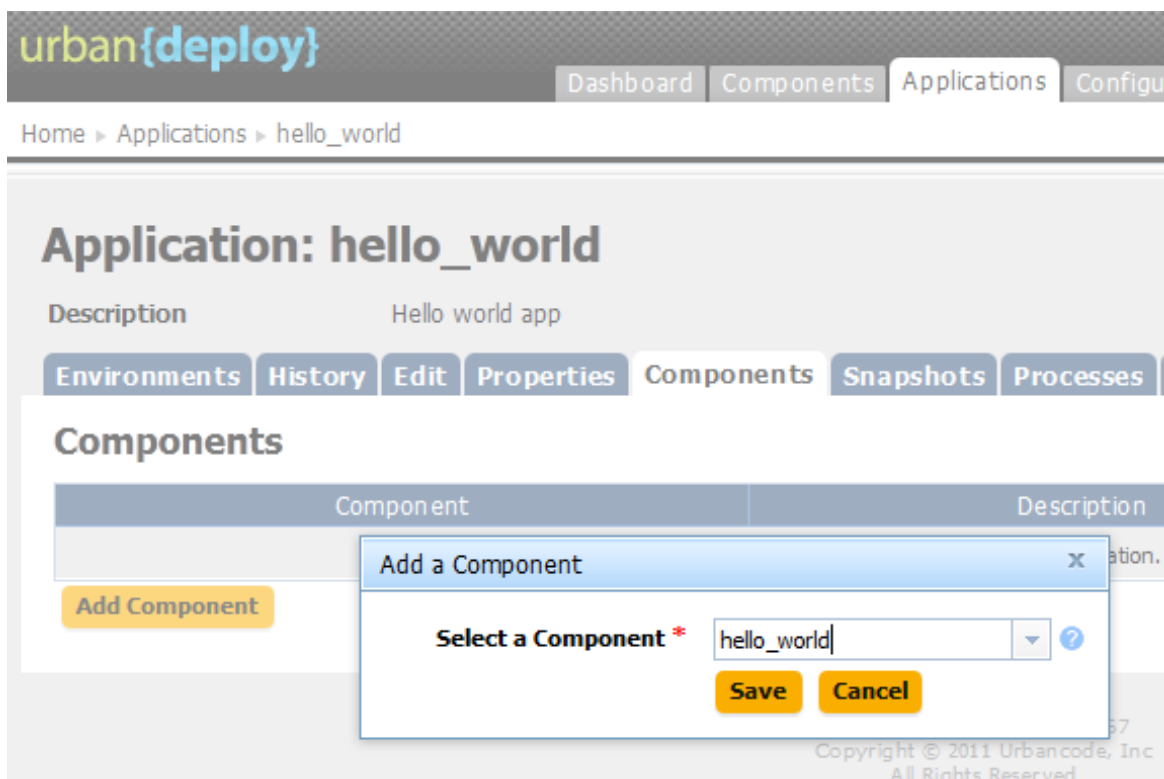


Figure 70: tbd

Licenses

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, UrbanDeploy 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.

The screenshot shows the UrbanDeploy web interface. At the top, there is a navigation bar with the logo 'urban{deploy}' and tabs for 'Dashboard', 'Components', 'Applications', and 'Configure'. Below the navigation bar, a breadcrumb trail reads 'Home > Applications > hello_world'. The main content area is titled 'Application: hello_world' and includes a 'Description' field with the value 'Hello world app'. A secondary navigation bar contains tabs for 'Environments', 'History', 'Edit', 'Properties', 'Components', 'Snapshots', and 'Processes'. The 'Components' section is active, displaying a table with columns 'Component' and 'Description'. An 'Add Component' button is visible. A modal dialog box titled 'Add a Component' is open, featuring a 'Select a Component *' dropdown menu with 'hello_world' selected, and 'Save' and 'Cancel' buttons. A copyright notice at the bottom right reads 'Copyright © 2011 Urbancode, Inc. All Rights Reserved'.

Figure 71: tbd

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.

This screenshot is identical to Figure 71, showing the UrbanDeploy web interface for the 'hello_world' application. It displays the 'Add a Component' dialog box over the 'Components' section. The interface elements, including the navigation bar, breadcrumb trail, application title, description, secondary navigation bar, and the 'Add a Component' dialog box, are all the same as in the previous figure. The copyright notice at the bottom right also remains the same: 'Copyright © 2011 Urbancode, Inc. All Rights Reserved'.

Figure 72: tbd

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. UrbanDeploy will prompt you to give the dates and times for the blackout.

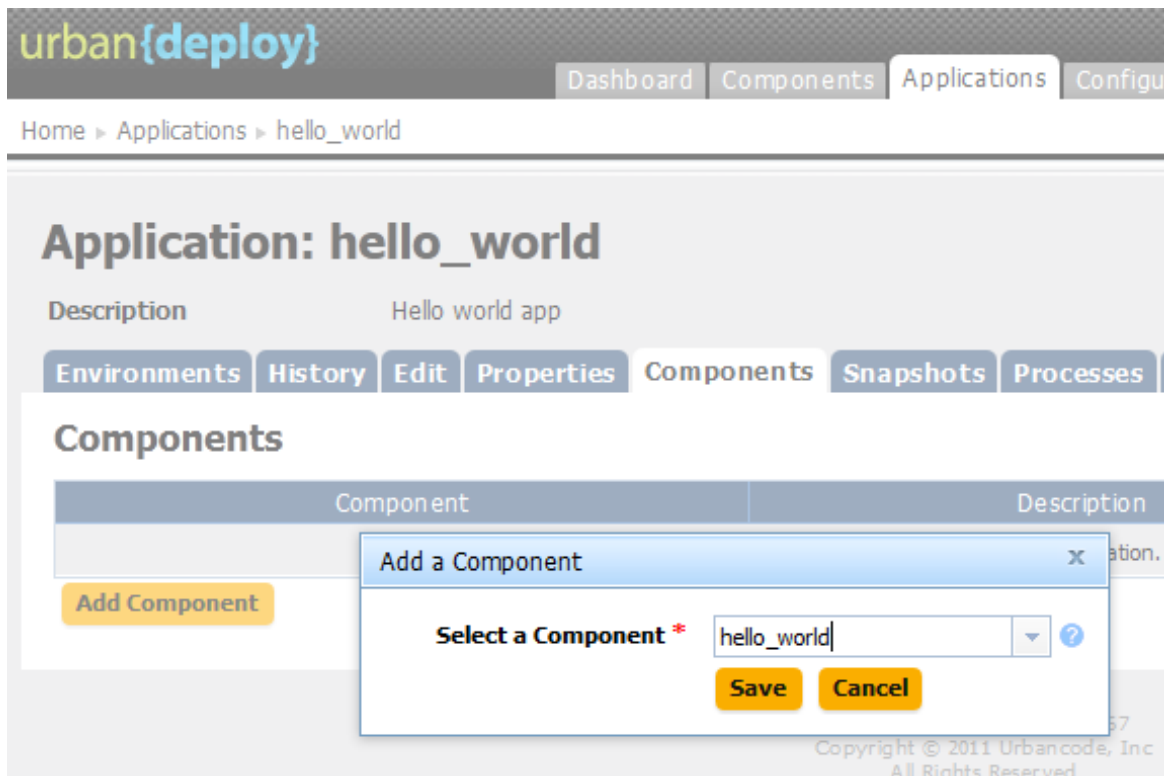


Figure 73: tbd

Network Settings

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, UrbanDeploy 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.

The screenshot shows the UrbanDeploy web interface. At the top, there is a navigation bar with the logo 'urban{deploy}' and tabs for 'Dashboard', 'Components', 'Applications', and 'Configure'. Below the navigation bar, a breadcrumb trail reads 'Home > Applications > hello_world'. The main content area is titled 'Application: hello_world' and includes a 'Description' field with the value 'Hello world app'. A secondary navigation bar contains tabs for 'Environments', 'History', 'Edit', 'Properties', 'Components', 'Snapshots', and 'Processes'. The 'Components' section is active, displaying a table with columns 'Component' and 'Description'. An 'Add Component' button is visible. A modal dialog box titled 'Add a Component' is open, featuring a dropdown menu labeled 'Select a Component *' with 'hello_world' selected. The dialog also includes 'Save' and 'Cancel' buttons. A copyright notice at the bottom right reads 'Copyright © 2011 Urbancode, Inc. All Rights Reserved'.

Figure 74: tbd

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.

This screenshot is identical to Figure 74, showing the UrbanDeploy web interface for the 'hello_world' application. It displays the 'Add a Component' dialog box over the 'Components' section. The interface elements, including the navigation bar, breadcrumb trail, application title, description, secondary navigation bar, and the 'Add a Component' dialog, are all the same as in the previous figure. The copyright notice at the bottom right remains 'Copyright © 2011 Urbancode, Inc. All Rights Reserved'.

Figure 75: tbd

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. UrbanDeploy will prompt you to give the dates and times for the blackout.

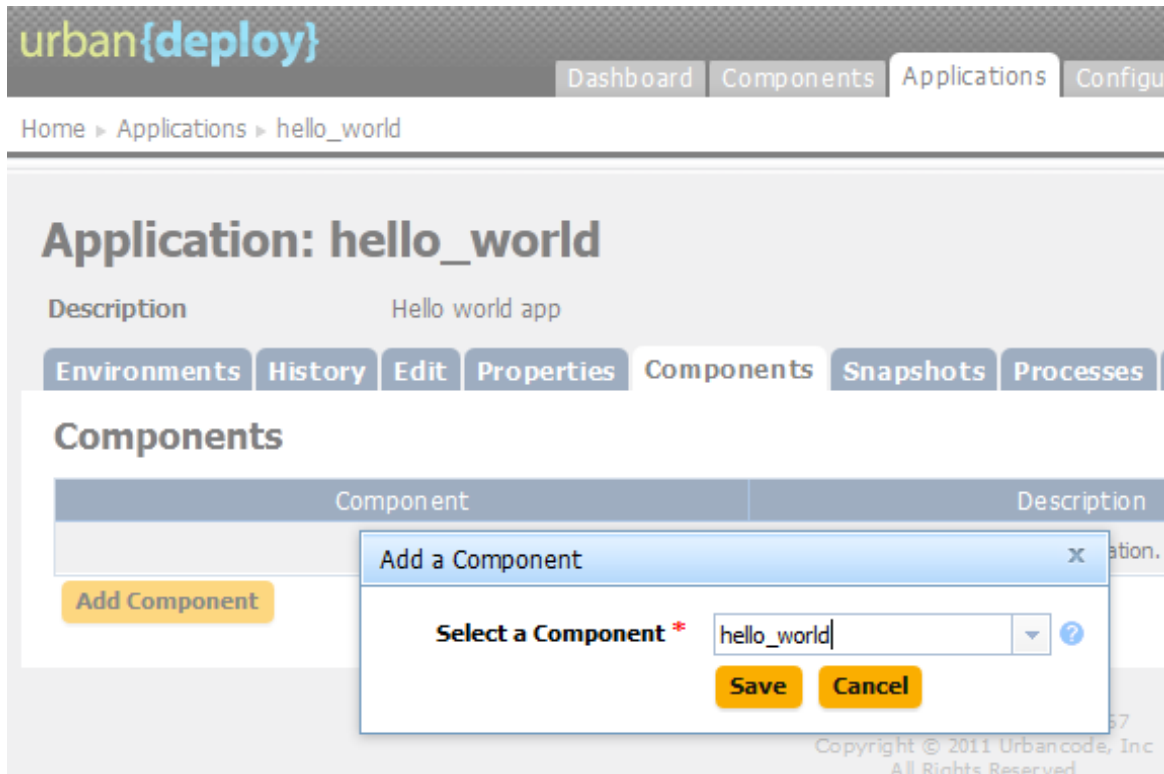


Figure 76: tbd

Notification Schemes

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, UrbanDeploy 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.

The screenshot shows the UrbanDeploy web interface. At the top, there is a navigation bar with the logo 'urban{deploy}' and tabs for 'Dashboard', 'Components', 'Applications', and 'Configure'. Below the navigation bar, a breadcrumb trail reads 'Home > Applications > hello_world'. The main content area is titled 'Application: hello_world' and includes a 'Description' field with the value 'Hello world app'. A secondary navigation bar contains tabs for 'Environments', 'History', 'Edit', 'Properties', 'Components', 'Snapshots', and 'Processes'. The 'Components' section is active, displaying a table with columns 'Component' and 'Description'. An 'Add Component' button is visible. A modal dialog box titled 'Add a Component' is open, featuring a dropdown menu labeled 'Select a Component *' with 'hello_world' selected. The dialog also includes 'Save' and 'Cancel' buttons. A copyright notice at the bottom right reads 'Copyright © 2011 Urbancode, Inc. All Rights Reserved'.

Figure 77: tbd

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.

This screenshot is identical to Figure 77, showing the UrbanDeploy web interface for the 'hello_world' application. It displays the 'Add a Component' dialog box over the 'Components' section. The dialog box has a title bar with a close button, a dropdown menu for 'Select a Component *' with 'hello_world' selected, and 'Save' and 'Cancel' buttons. The background shows the application's navigation and description. A copyright notice at the bottom right reads 'Copyright © 2011 Urbancode, Inc. All Rights Reserved'.

Figure 78: tbd

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. UrbanDeploy will prompt you to give the dates and times for the blackout.

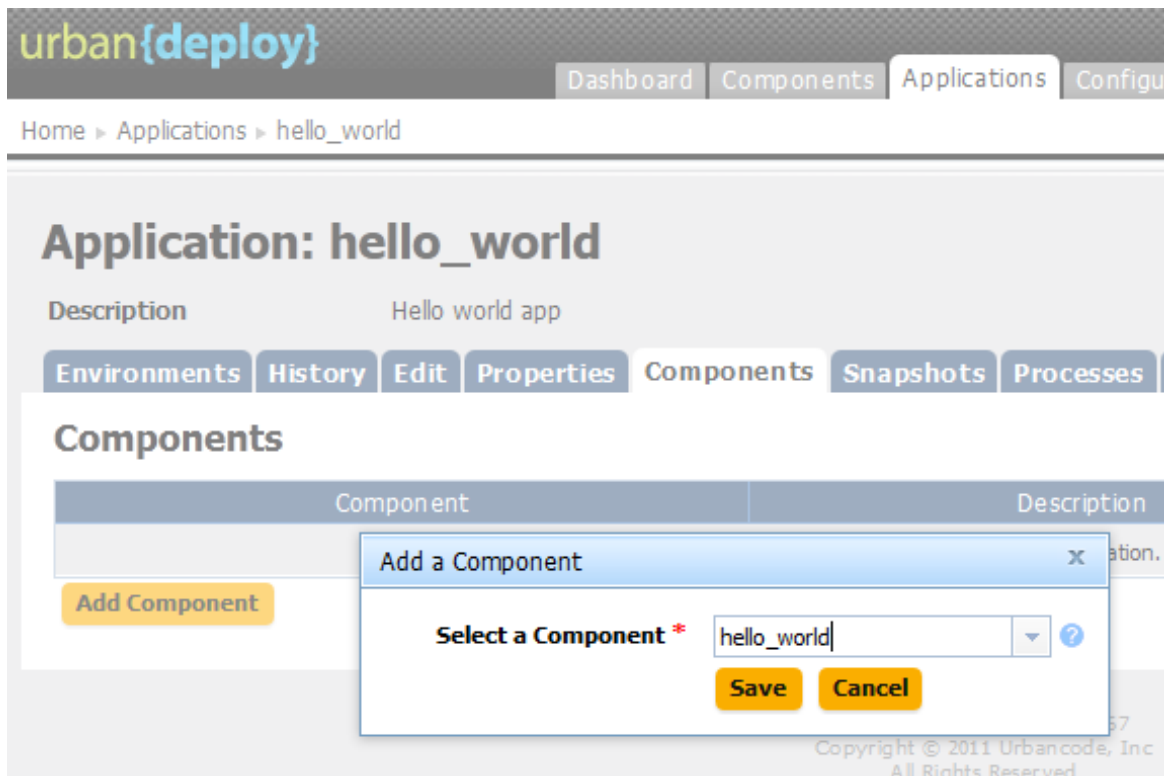


Figure 79: tbd

Properties

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, UrbanDeploy 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.

The screenshot shows the UrbanDeploy web interface. At the top, there is a navigation bar with the logo 'urban{deploy}' and tabs for 'Dashboard', 'Components', 'Applications', and 'Configure'. Below the navigation bar, a breadcrumb trail reads 'Home > Applications > hello_world'. The main content area is titled 'Application: hello_world' and includes a 'Description' field with the value 'Hello world app'. Below this, there is a row of tabs: 'Environments', 'History', 'Edit', 'Properties', 'Components', 'Snapshots', and 'Processes'. The 'Components' tab is active, and a table with columns 'Component' and 'Description' is visible. Overlaid on this table is a modal dialog box titled 'Add a Component'. The dialog contains a label 'Select a Component *' followed by a dropdown menu with 'hello_world' selected. At the bottom of the dialog are 'Save' and 'Cancel' buttons. A copyright notice at the bottom right of the page reads 'Copyright © 2011 Urbancode, Inc. All Rights Reserved'.

Figure 80: tbd

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.

This screenshot is identical to Figure 80, showing the UrbanDeploy web interface for the 'hello_world' application. It displays the 'Components' section with an 'Add a Component' dialog box overlaid. The dialog box prompts the user to 'Select a Component *' and shows 'hello_world' in the dropdown menu. 'Save' and 'Cancel' buttons are present at the bottom of the dialog. The background interface includes the navigation bar, breadcrumb trail, and application details. A copyright notice at the bottom right reads 'Copyright © 2011 Urbancode, Inc. All Rights Reserved'.

Figure 81: tbd

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. UrbanDeploy will prompt you to give the dates and times for the blackout.

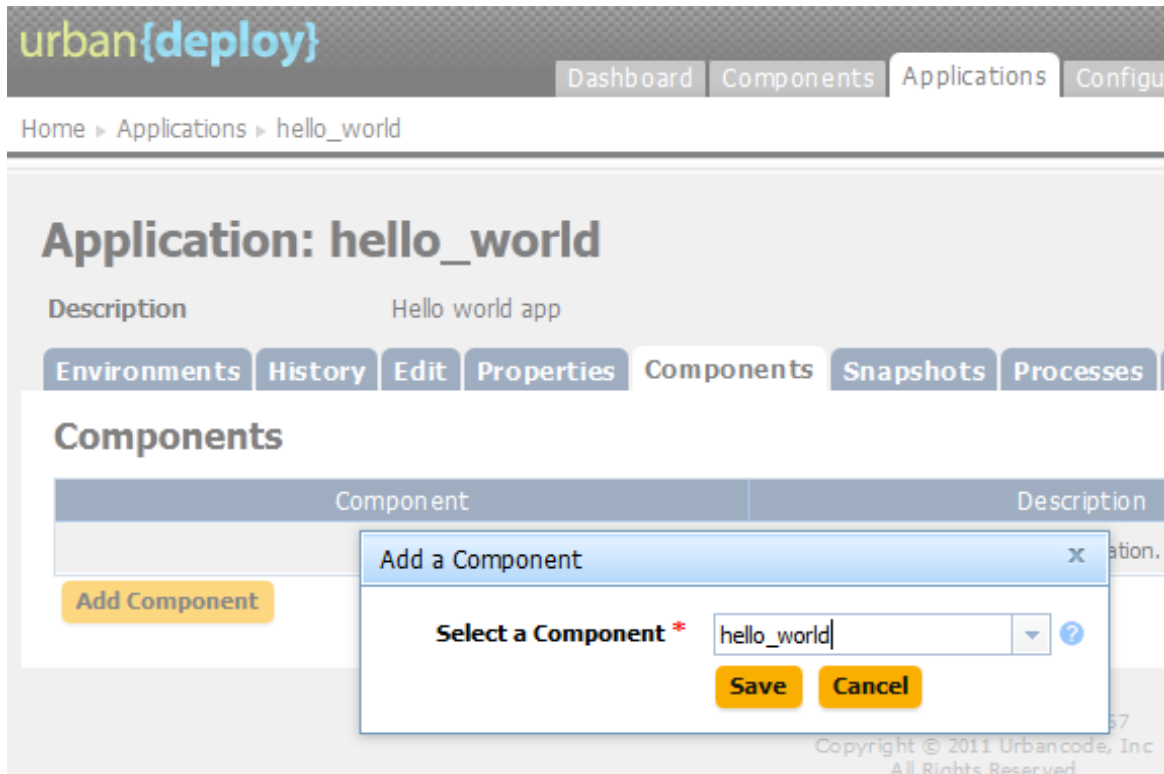


Figure 82: tbd

System Settings

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, UrbanDeploy 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.

The screenshot shows the UrbanDeploy web interface. At the top, there is a navigation bar with the logo 'urban{deploy}' and tabs for 'Dashboard', 'Components', 'Applications', and 'Configure'. Below the navigation bar, a breadcrumb trail reads 'Home > Applications > hello_world'. The main content area is titled 'Application: hello_world' and includes a 'Description' field with the value 'Hello world app'. A secondary navigation bar contains tabs for 'Environments', 'History', 'Edit', 'Properties', 'Components', 'Snapshots', and 'Processes'. The 'Components' section is active, displaying a table with columns 'Component' and 'Description'. An 'Add Component' button is visible. A modal dialog box titled 'Add a Component' is open, featuring a dropdown menu labeled 'Select a Component *' with 'hello_world' selected. The dialog also includes 'Save' and 'Cancel' buttons. A copyright notice at the bottom right reads 'Copyright © 2011 Urbancode, Inc. All Rights Reserved'.

Figure 83: tbd

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.

This screenshot is identical to Figure 83, showing the UrbanDeploy web interface for the 'hello_world' application. It displays the 'Add a Component' dialog box over the 'Components' section. The dialog box has a title bar with a close button, a dropdown menu for 'Select a Component *' with 'hello_world' selected, and 'Save' and 'Cancel' buttons. The background shows the application's navigation and description. A copyright notice at the bottom right reads 'Copyright © 2011 Urbancode, Inc. All Rights Reserved'.

Figure 84: tbd

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. UrbanDeploy will prompt you to give the dates and times for the blackout.

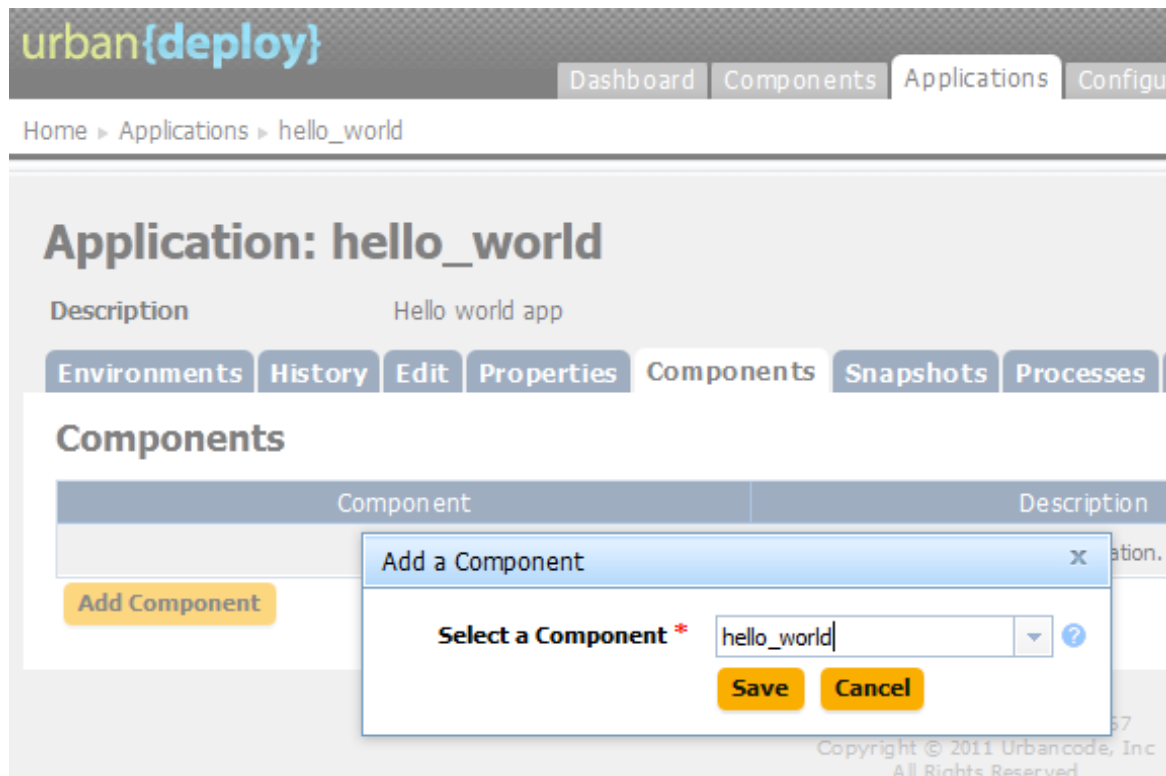


Figure 85: tbd

Part

V


Reference

Topics:

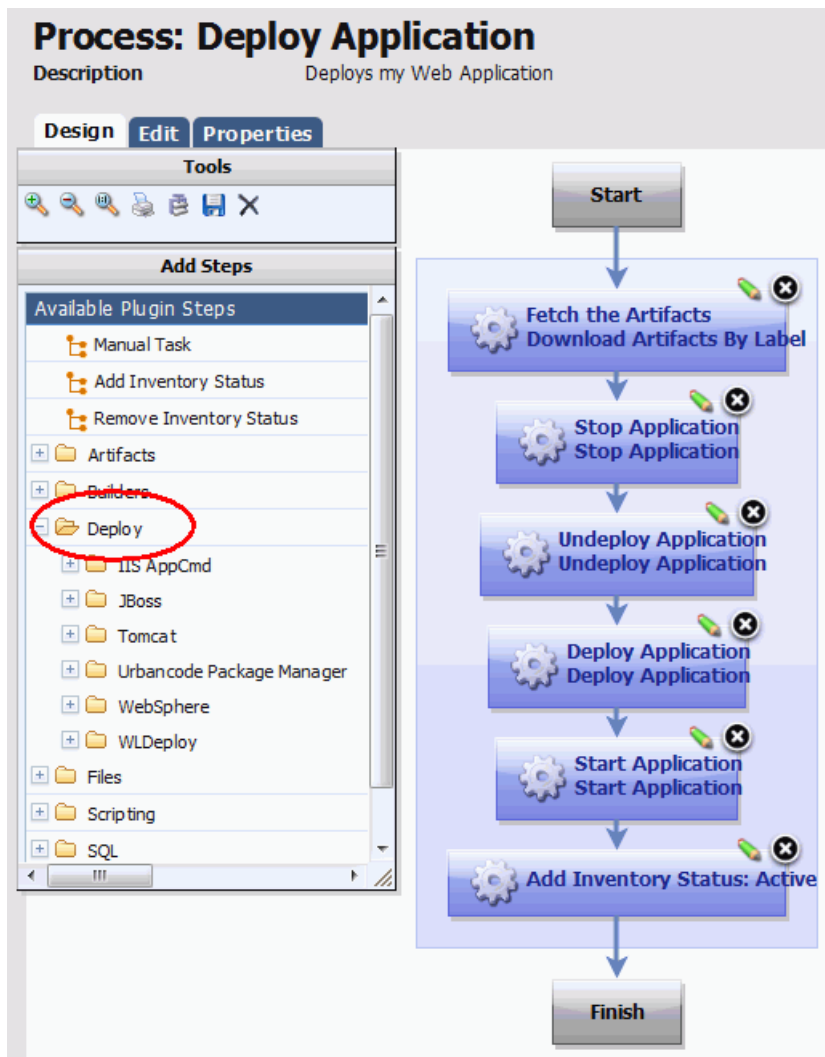
- [Plug-in Integration](#)
- [Source Configuration Reference](#)
- [Notifacations](#)
- [Configuration](#)
- [Inventory](#)
- [CLI Reference](#)

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.

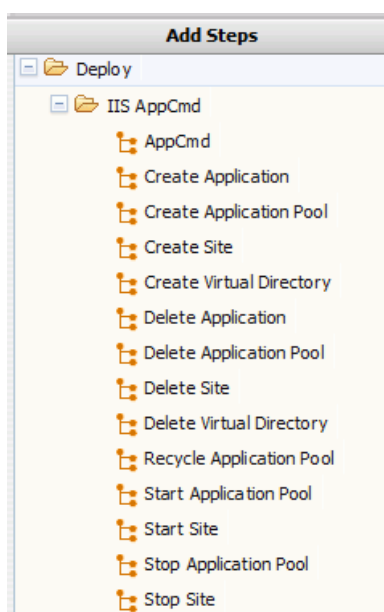
Groovy Plug-in

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

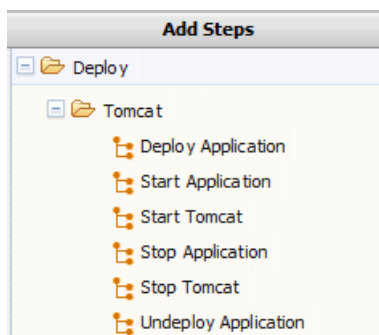
The Oracle SQL-Plus integration consists of a single step that you can include in any deployment process or other process. The most common use case opening and running a SQL statement when updating a database. If the step is used within a larger process, ensure that you set the order correctly. For example,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

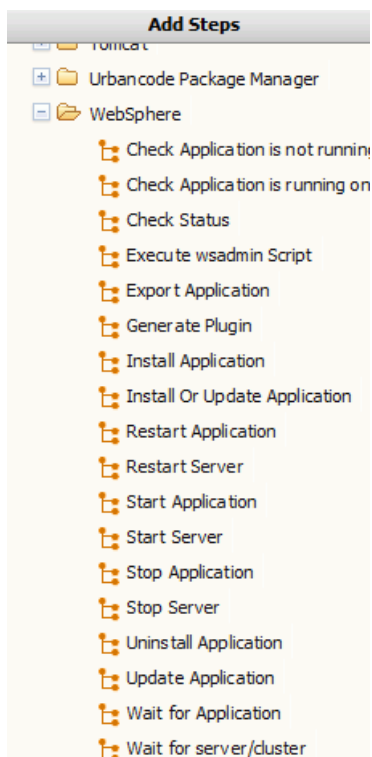


When running the process, ensure that the Resource running the step has access to the Tomcat full client jar, that uDeploy has a user and password to log to connect with, and that the full path to the Tomcat full client jar is available.

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/server
3. Update/uninstall the application
4. Start the application/server

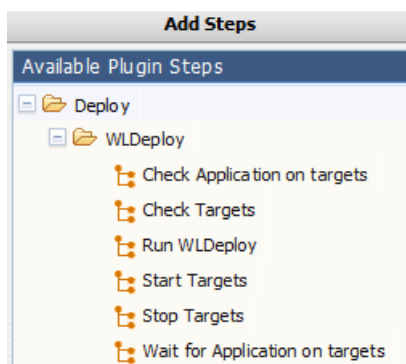


When setting up your deployment, you select one of the pre-defined steps and add it to your process. Step configuration is straightforward: you generally need to give connection information as well as the location to any executables.

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.

Advanced Automation 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

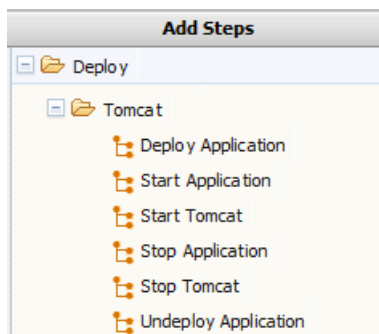
These advanced automation steps will retrieve properties and environments from uDeploy.

Plug-in Automation

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.

Source Configuration Reference

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

AntHillPro

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

PVCS Version Manager

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

Perforce

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

Luntnbuild

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

Maven

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

Jenkins

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

File System

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

Team Forge

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

Team City

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

Subversion

Typically, you will need to perform the following, in order:

1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

Team Foundation Server (TFS)

Typically, you will need to perform the following, in order:


1. Gather Information
2. Configure Resources
3. Configure Components
4. Configure Applications and Snapshots

Gather Information

Notifacations

UrbanDeploy can send notifications to users based on a number of events that occur. Most commonly, UrbanDeploy 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 UrbanDeploy with your existing infrastructure. If you have not already done so, set up UrbanDeploy with LDAP prior to configuring Notifications: UrbanDeploy 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.

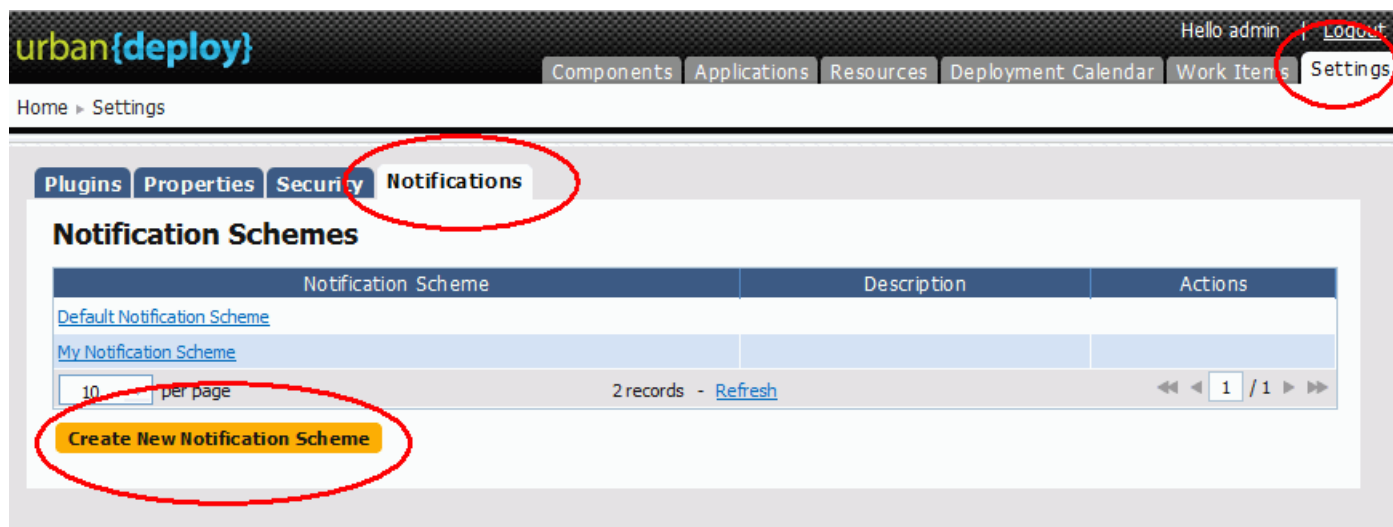


Figure 86: Notification Schemes

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.

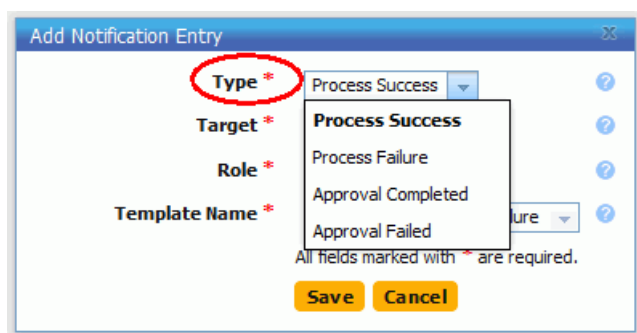


Figure 87: Notification Type

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 88: 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 89: 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 90: 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.

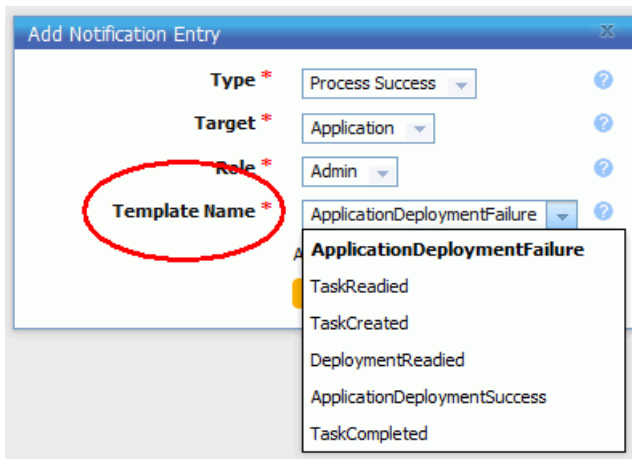


Figure 91: tbd

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)' 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 divided into two sections. On the left is a tree view showing the hierarchy of applications and components. The tree is expanded to show the 'stressTestNov1' application. The tree structure is as follows:

- 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)

On the right is the 'Application: stressTestNov1' properties section. It features an 'Add Property' button and a table with the following structure:

Name	Value	Description	Actions
No properties found. - Refresh			

Below the table is another 'Add Property' button.

Figure 92: 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 current page is 'Home > Configuration'. On the left, a tree view under 'Application / Component / Environment' shows a hierarchy for 'JPetStore', with 'JPetStore' selected and circled in red. The right panel, titled 'Application: JPetStore', displays the 'Properties' section. It features an 'Add Property' button, a table with columns 'Name', 'Value', 'Description', and 'Actions', and a pagination control showing '10 per page' and '1 record - Refresh'.

Figure 93: 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.

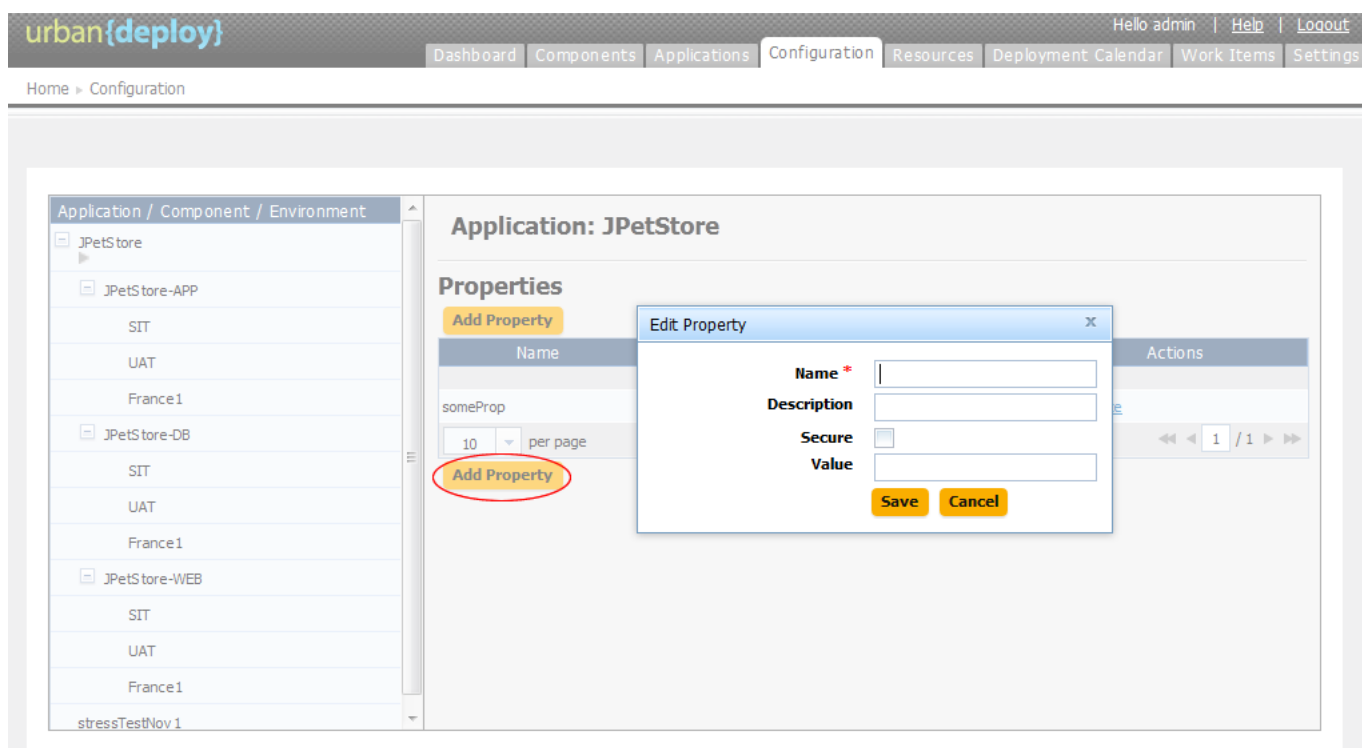


Figure 94: Edit Property pop-up

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 UrbanDeploy'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.

Application / Component / Environment

- [-] JPetStore
 - [-] JPetStore-APP
 - SIT
 - UAT
 - France1
- [-] JPetStore-DB
 - SIT
 - UAT
 - France1
- [-] JPetStore-WEB
 - SIT
 - UAT
 - France1
- stressTestNov 1

Component: JPetStore

Properties

[Add Property](#)

Name	Value

[Add Property](#)

Environment Property

Define properties here to be given values

[Add Property](#)

Name	Label	Required

[Add Property](#)

Configuration Template

Name

[Create New Configuration Template](#)

Figure 95: 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 displays the 'Configuration of JPetsStore' interface. On the left, a tree view titled 'Application / Component / Environment' shows a hierarchy: JPetStore (expanded) -> JPetStore-APP (expanded) -> SIT (circled in red) -> UAT -> France1. Below this, JPetStore-DB and JPetStore-WEB are also expanded, each showing SIT, UAT, and France1 environments. At the bottom of the tree is 'stressTestNov 1'. On the right, the 'Resources and Groups' section shows a resource path: /SIT/APP -> urbandeploy-agent. Below this are two yellow buttons labeled 'Add a Resource'. The 'Environment Properties' section below shows the text 'No environment properties have been defined'.

Figure 96: 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

Home > Resources > SIT Environment

Resource: SIT Environment

Description Agent installed in SIT.

Main Edit Roles Properties Custom Properties Security

Current Inventory

Component	Version	Date
My Application (WEB)	1.0	8/18/11 10:43 AM
My Application (APP)	1.0	8/18/11 10:43 AM
My Application (DB)	1.0	8/18/11 10:36 AM

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

Figure 97: 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}

Components

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	
SIT Environment	1.0	8/18/11 10:43

10 per page 1 record - !

Figure 98: 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} Components Applications Resou

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
<input type="checkbox"/> My Application (APP)	1.0	
SIT Environment		8/18/11 10:43 AM

Figure 99: 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.

```

createComponent

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

```

group    -path, --path
          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...]
```

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
    Read JSON input from a file with the given filename. See command
for
    requirements.
```

Options

```
-user, --user
    Required. Name of the user
```

