



# Environment Automation Strategy Whitepaper

## Patterns for Environment Consistency

March 2008

### Confidential Information & Copyright

This document contains information that is proprietary to Odecee Pty Ltd. All information within this document relating to the business of Odecee and is the property of Odecee Pty Ltd and deemed to be confidential information of Odecee Pty Ltd. Unauthorised disclosure of this information will be considered a breach of confidentiality resulting in material damage to Odecee Pty Ltd.

Overview .....	3
Environment Management Strategies .....	4
Manual Configuration .....	4
Automated Configuration .....	4
SDLC Overview .....	5
Deployment Requirements of the SDLC .....	5
Automated Deployment and Configuration Strategy .....	7
Scope of Scripting .....	7
Turn Key Deployments .....	7
Environment Agnostic Design .....	7
Version Control .....	8
Consistency in Deployments and Environment Configuration .....	8
Automated Deployment Processes .....	9
Nightly Deploys .....	9
Labelled Deploys .....	9
Description Of Process .....	9
Return On Investment .....	11
Further Considerations .....	12
Continuous Integration and the Build Loop .....	12
Monitoring .....	12
Environment State Enquiry .....	12

## Overview

The Software Development Life Cycle presents a number of challenges to all aspects of systems development, one of which is environment management. This whitepaper presents a strategy to address concerns relating to environment consistency and availability for this key activity in the SDLC. The Pattern that is proposed in this whitepaper addresses a number of concerns that will increase efficiency in environment related changes to provide a high level of agility and promote consistency in all managed environments for a program of work.

Environment management has many facets. Primary, it involves the construction of the physical hardware, which represents all nodes in the (inter) connected system. Each node in the system can be seen to provide a service to the holistic system and, is therefore interconnected with other nodes in the system. Given that most nodes in the system have dependencies to other nodes, the entire environment can be represented as a system of interconnected, dependent nodes. This fact increases the fragility of a system and enforces a greater need on environment consistency.

A number of strategies exist that attempt to solve the consistency concern. Each one of these strategies will impose a risk onto a program and must be mitigated in the form of an environment management strategy.

This whitepaper aims to present a proven strategy to solve the problem of environment consistency management. The presentation will not detail the technologies that implement the strategy as factors such as operating system selection and technology stack will affect the selection of tools used to implement the strategy.

## Environment Management Strategies

Environment management has many facets, the most important one being the attainment of configuration consistency. The following sections define two strategies that are currently used by most programs.

### Manual Configuration

The definition of manual configuration is when a technician manually performs every configuration change made to an environment. Typically, the technician will issue commands to the operating system and/or software components of the operating system manually; either by the use of a Graphical User Interface or Command Line. There is little or no automated means of which the configuration changes are executed. A technician will typically follow a set of instructions that will guide the user through a number of operations that aims to apply the required configuration settings.

### Automated Configuration

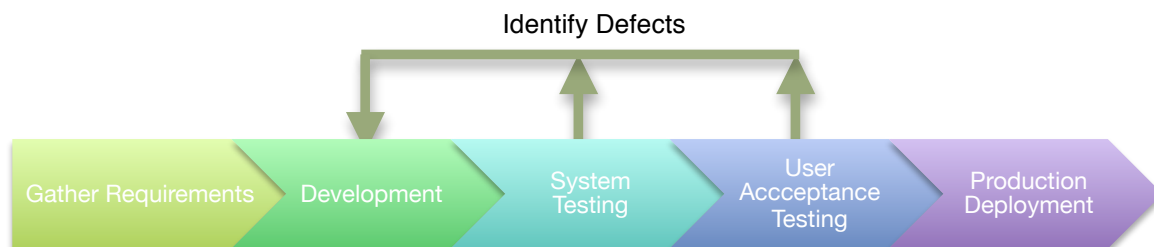
The definition of Automated Configuration is when software applies the configuration system to a target node. This is typically a script that is executed by a technician, which applies all the configuration changes to the node that would otherwise be a manual task.

The automation process typically required a development activity and maintenance strategy for any artefacts generated by the development activity.

The goal of the Automated Configuration strategy is to deliver a “turn-key” solution to all configurations aspects of the system under management. This strategy will implement the “single execution point” or “double-click” configuration solution.

## SDLC Overview

There are many methods defined for an SDLC. The typical approach can be defined as the well-known Waterfall Model, where the process of development will typically follow a process like:



As illustrated in the above diagram, requirements are defined, developed, tested then finally deployed to a target environment. There is feedback from the System and User Acceptance Testing components of the process to perfect the requirements defined early in the process.

## Deployment Requirements of the SDLC

When the system components are developed, they will be deployed to a target environment. The two main deployment purposes are typically for:

### 1. Testing

System testing to verify the correct implementation of the system. Performance and Volume Testing to ensure the scalability and stability requirements of the system are met. Integration Testing to ensure the system is functional from a component perspective (typically a development concern) and User Acceptance Testing for the business to agree the system is implemented as required.

### 2. Production Deployment

The deployment of the system to begin operational usage and implement the business function.

As illustrated above, during the SDLC phases, the system will require a larger number of deployments into different environments (System Test, Integration Test, User Acceptance Testing environments etc.). In a typical SDLC, this deployment process will be repeated a number of times to ensure the system is well tested and defect free.

Environment Management Strategies section of this whitepaper, its evident that the Manual Configuration strategy has the potential to become problematic. This can become a very verbose process when the deployment process is frequently repeated; especially if the environment design has the following characteristics:

1. Large number of system nodes (highly distributed)

This is typical of distributed systems where the nodes in the system are large (e.g. greater than 10). Each node can require some configuration to complete the deployment.

Technician is required to connect to each node and execute a number of configuration operations on the node.

Complexity of deployment is further increased as the number of configurable software components on each node increases.

2. Complex interconnections between nodes (highly interconnected and dependent on other system nodes)

As the interconnections within the nodes of the system increase, the configuration of the node will also increase as a factor of the number of connected nodes.

3. Complex software stack on nodes (greater configuration complexity)

The greater the number of software components on the node, the greater number of configuration commands are required by the technician to apply configuration changes.

As manual processes are prone to error, the effects of erroneous configurations can have a significant impact on system stability and ultimately affect project delivery timelines. Therefore, an efficient environment configuration strategy is required to reduce deployment times and increase the SDLC agility.

# Automated Deployment and Configuration Strategy

Having defined the requirements for agile environment configuration in the context of an SDLC, we will now define the process of attaining and implementing an automated deployment strategy.

## Scope of Scripting

The view of the scripting strategy is to script every configuration operation of the system as possible. The value of the scripting strategy will diminish as the gaps in the scripting implementation increase. Manual steps in the process require careful consideration as each manual intervention is subject to inconsistent configuration.

## Turn Key Deployments

The solution for environment configuration management must be implemented as a “Turn Key” solution. This implies a high level of automation for all environment and application configuration. This automation is typically implemented using a number of scripting languages; selected from a list of candidate scripting languages available for the target operating system and software stack.

Each of the available scripting languages should be selected based on the merit and skill levels in the team. Due diligence should be used when selecting the appropriate scripting languages as selection of scripting languages can have an impact on (i) script execution times / performance, (ii) script maintainability, (iii) vendor support, (iv) team skill levels, (v) applicability, (vi) other general architectural concerns.

The focus of the Turn Key solution should be to achieve a full deployment (which includes all system level configuration) with the execution of a single command. This way, the environment is configured in a predictable and consistent manner.

## Environment Agnostic Design

As a typical software project will involve 2+ environments, the script implementation must cater for environment differences and be designed and implemented to be environment agnostic. Environment differences are typically defined as having differences in:

1. Host names

Each node in the system has a unique name that can be connected to other nodes that are specific to an environment, e.g. System Testing environment

2. Usernames

Every environment will typically have a security system dedicated and isolated to the environment, e.g. LDAP that could contain user details specific to the environment.

### 3. Connection Strings

Connection strings define end points for system components like RDBMS, LDAP, Content Engines etc. Furthermore, they can typically also include definitions in naming directories like JNDI.

The effectiveness of the deployment and configuration scripts will require a parameterised implementation that factors variables into environment configuration. An automated scripting solution is ineffective without a solution to this design consideration.

## Version Control

The implementation of the environment automation system must be version controlled to facilitate identification and labelling of script implementation version. Therefore, all scripts that are developed must be version controlled in a Version Control System like Subversion or CVS.

All changes to the scripts can be tracked and labelled along with the developed software version. This way, the environment configuration/deployment scripts are tightly coupled to a version of the application to ensure the application is consistently deployed irrespective of application version. For example, if the software is labelled (using a labelling process) then the deployment scripts that are developed to deploy the particular version of software will be guaranteed to deploy the selected version of software.

## Consistency in Deployments and Environment Configuration

The implementation must be designed so that it will leave environments in a consistent state even after an error during script execution. This can (but does not require) the capabilities of the scripts to support transaction semantics to provide ACID (Atomic, Consistent, Isolation, Durable) principles to the automated configuration/deployment processes.

Atomicity is required to group dependent configuration operations to such that, if a single operation fails, then all related operations must be rolled back. This way, configurations that depend on other configurations will always be rolled back so leave the system in a consistent state post failure.

Consistency of environment configuration changes can only be made possible by scripts that demarcate grouped operations and further recovery of configuration operations post failure. This is typically implemented as a rollback operation where all configuration changes bounded by a transaction are “un-configured” in the event of a failure.

Given that deployment scripts are not required to run in parallel, the isolation principle does not hold and can be ignored.

However, durability is required from the scripting design, which requires the state of the changes to be persisted post committal. I.e. once a system operation is executed by the deployment scripts,

they must be made remembered by the system after the script completes its execution.

## Automated Deployment Processes

Enablement of automated deployments should be inline with the SDLC processes. Simply put, the deployments of the software versions should be scheduled to increase the effectiveness of the solution. The strategy needs to be viable to increase system availability and promote testability of the system. For example, the execution times of the deployment & configuration scripts must run within the allocated time periods and must be predictable, e.g. overnight deploys or bi-daily deploys.

### Nightly Deploys

As the development teams iteratively build the application components, the application components should be assembled and deployed to a target environment as frequently as possible. The frequency of the deployment of the application components depends on a number of factors, some of which were briefly disused in this whitepaper. Firstly, the speed at which the deployment and environment configuration scripts execute will be the major factor in the frequency of the deployments. Therefore, deployment scripts should be designed and built with speed of execution in mind. Some parallelism can also be factored into the design where possible to decrease the execution times.

The target for environment deployments should be aimed at nightly execution. For example, the build scheduler should execute the deployment scripts at the end of business hours and should complete by the beginning of business hours the next day. This way, the latest build from the development repository will be available daily for QA purposes.

### Labelled Deploys

A labelled deploy is enabled by the use of a Version Control System, such as ClearCase or CVS. The aim of a Labelled Deploy is to apply a label to the “head” of the source branch that defines a version of the source code. This way, a version of the scripts can be retrieved and built/executed, promoting consistency and reliability.

### Description Of Process

As the development team add source code artefacts into the version control system, the artefacts in the system are (potentially) available for deployment purposes. Quality considerations need to be factored into the “check-in” procedure to ensure the source code is integrated into the code repository. At an allocated time (end-of-day), the code artefacts are built into deployable forms and then installed into a target environment. The procedure in which these build artefacts are installed into the environment are delivered by deploy scripts. Therefore, the deployment scripts are executed to move the built artefacts into the target environment.

Furthermore, all configurations that are required to enable a functional deployment environment are

also required to facilitate a correct and operational platform. Simply put, the environment configuration scripts must also execute the required operations to configure the environment and enable the correct operation of the application.

CruiseControl forms the backbone of the automation solution. CruiseControl is an out-of-the-box build scheduler that can be extended to run scripts that implement both environment automation and configuration operations. Functionally, CruiseControl uses Ant (an open-source XML scripting language) which is perfect for automating command execution. Ant scripts are the entry point into all environment configuration and deployment scripts. CruiseControl is configured to point to a set of scripts and scheduled to execute at a given frequency.

For clarity purposes, the procedure is detailed as follows:

Preface:

- Environment Configuration Strategy is known to all and architected to automate every configuration aspect as possible.
- The application artefacts are built using build scripts.
- Deployment scripts are implemented to migrate the application artefacts into the target environment and furthermore configured and installed into the appropriate containers.
- CruiseControl is installed and appropriate build and configuration tasks scheduled.

High level automation Process:

1. CruiseControl will extract the environment configuration scripts from the version control repository.
2. Environment configuration scripts are executed to configure the environment and apply the configuration to the environments.
3. Extract source code artefacts from version control and built using build scripts to create application artefacts.
4. Application artefacts are labelled for consistency and ease of identification.
5. Deployment scripts are executed to copy the artefacts into the appropriate locations and the container specific configuration scripts are executed to install the application components.
6. Integration testing scripts are executed to ensure the system is stable and all application components and environment configurations have integrated correctly.

All the above will ensure that the application components are deployed, and furthermore, verify the correct operation of the application in the target environment

## Return On Investment

An automated environment configuration and deployment processes is a costly exercise to perfect. The scripting and ongoing maintenance requirements can be costly to the business. However, the expenses are upfront and the cost ramifications of a clean, dependable, consistent and efficient are redeemed later in the SDLC. The processes are proven to take the pressure last minute configuration changes that can potentially lead to system instability and outages. Therefore, upfront costs are redeemed later in the SDLC as the configuration and deployment of the application require less human input.

The “turn-key” solutions are proven to be simpler to administer and execute. They provide quality environment configuration output throughout the SDLC by ensuring the environments remain consistent and free from unknown and rouge changes. Administering the process is critical to ensure the environment configurations maintain consistent and known.

## Further Considerations

This section of the whitepaper will look a little further into some of the considerations that enable the “turn-key” environment configuration and deployment automation. They extend the patterns of this whitepaper by defining some further processes and patterns that can add great benefit to the automation approach.

### Continuous Integration and the Build Loop

Continuous Integration is a pattern and processes that most developers are familiar with. The definition of Continuous Integration relates to the process of integrating code into a common source code repository. Continuous Integration is a quality enabler which forces the frequent execution of unit and integration tests over a code base to view the regressive impacts and sociability of source code into a branch of code. As developers check their source code into the code repository (SVN, ClearCase, CVS, PVS etc.), the code artefacts are build and tested to view the successful integration of the developers code. Changes to the code base that are erroneous or have a regressive impact are detected before the code artefacts are delivered into a testing environment. This way, the development team can ensure they have a consistent view of code quality to mitigate the detection of defects late in the SDLC.

The Build Loop is a function of the frequency and coverage provided by the unit and integration test suite. The trigger for the build is typically a “check-in” into the source code repository or manual execution of the build loop task. Apart from building the source code, other functions can also be incorporated and executed as part of the build; like automated application testing scripts, e.g. Selenium.

### Monitoring

Monitoring is a key aspect of the Continuous Integration and Build Loop. A monitor which contains the status of the last build and current build for all build branches is mandatory to view the current status of the build. Red and green indicators are typically used to indicate the build loop health; red being a failed build and green being a successful build. This single point of monitoring facilitates the broadcasting of the status, and therefore health, of the current build (or deployment).

CruiseControl comes out-of-the-box with a HTML monitoring agent that can be used to present the build status to the entire team. Other options available are Widgets which can be installed on many computers that can present build loop status in the same red and green colors. Cruiselt is a widget which also has build control built into the UI to allow build to be executed outside of the scheduled build times to allow for greater control of the build cycles.

### Environment State Enquiry

Environment State Enquiry is a function, typically implemented by scripts, which check the status of selected software services in an environment. The design of these scripts should be to perform all the relevant query operations for a software service to ascertain the status of the service started, stoped,

not responding etc. This way, from a central point (e.g. script) an environment state enquiry could be kicked off to view the status of the environment. This enables the quick troubleshooting of environment issues by providing targeted status views of software services.