Intel Cloud Integrity Technology 3.0

**Quick Start Blueprint**

# Background

Cloud Integrity Technology software is packaged as a bundle of Linux self-extracting installers for attestation service, trust director, key broker, key broker proxy, and trust agent. Each installer can be fully automated through the use of an “environment file” in which the administrator defines variables corresponding to required or optional settings.

In addition, the openstack controller must be configured to use the attestation service and patched to show trust information.

Understanding where and how to deploy the components, and which settings are required or optional for each one, can be overwhelming for a first-time user.

The deployment wizard (the Wizard) is needed to simplify the Cloud Integrity Technology deployment experience for both testing and operations.

# Data Model

The quickstart data model is in the Java package com.intel.mtwilson.deployment.

A DeploymentRequest describes the features, environment, layout, credentials, and settings specified by the user.

The server processes the features, environment, and layout specified in a DeploymentRequest to generate a set of deployment targets. These targets, together with the credentials and settings specified by the user, comprise the Deployment descriptor.

A target is a host and set of features to be installed on that host.

The environment is one of three possible constants: PRIVATE, PROVIDER, or SUBSCRIBER.

The layout is determined by the user interface or client and provided to the server as a set of targets. In the UI, we currently support two layout options: ALL IN ONE and ONE-PER-HOST. The user can edit the ONE-PER-HOST to consolidate some of the services by specifying the same host name/IP for them.

# Scheduling

There are six software packages, two built-in layouts and a custom layout option, as well as optional tasks that must be executed depending on other selections. Ensuring that installation steps happen in an effective order is an important capability for the automated deployment.

The Task interface includes a dependencies field so that tasks may declare dependencies on other tasks. This allows a task scheduler to ensure that the dependencies are executed before the dependent tasks.

The DeploymentTaskFactory is responsible for generating the entire set of tasks that will be needed for a deployment. It considers the user’s input for selected features and software packages, and it knows which tasks are required for which software packages. When it generates the tasks, it sets up dependencies between tasks to ensure that tasks that need the least input (or no input) are run first, and tasks that depend on them are run later. This especially applies to integration tasks, for example creating a user for Trust Director in the Attestation Service after Attestation Service installation and then providing that credential to the Trust Director installer.

The software packages descriptor (software-packages.json) defines the available software packages as well as their dependencies on each other. This is used by the DeploymentTaskFactory when it creates synchronization tasks that ensure this software package dependency order.

# Monitoring Installer Progress

After an installer is copied to the target host and is executed, the user needs to see some progress indication because some installers may take several minutes to complete.

The quickstart wizard deploys a second script with each installer to monitor the installer’s progress. It does this by comparing the installer’s output to a “marker file” that has unique lines to expect in the installer’s output and measuring progress according to the location of the matched line in the marker file. The monitor script matches the lines in a “forward-only” fashion, so when the installer starts the monitor looks for any marker lines that match its output, but as matches are found then it only looks for subsequent lines to match subsequent output. Progress percent is calculated as the line number of the last line matched divided by the total number of lines in the marker file.

## Preparing the marker file

This is an easy procedure and only needs to be repeated when the installer output changes significantly, for example if a lengthy new operation is added or if the text of existing messages is changed.

First, run the installer in non-interactive mode (environment variables to answer all prompts) and redirect stdout to a file.

Second, edit the output file to identify & retain unique lines that can mark the installer’s progress. Delete all other lines.

Guidelines for selecting marker lines:

* Preferably complete lines but any unique part of a line would work
* There should not be another line or part of a line in the output that would match; if that happens only the first occurrence will count
* There must not be any variable values that would cause a line not to match the corresponding output from a subsequent installation
* There must not be any characters that would interfere with “grep” on the command line so no pipes, quotes, brackets, etc.
* The last line in the marker file should be after the last operation executed by the installer because when it is found the progress will be marked at 100%

## Alternatives

Design alternatives considered are shown below. The selected design is highlighted in green.

Table

|  |  |
| --- | --- |
| **Alternative** | **Pros(+) Cons(-)** |
| Show an “indefinite progress indicator” such as a spinning wheel | + easy to implement  - doesn’t measure progress, so equivalent to showing “Please wait” and only slightly better than not showing anything at all |
| Modify all installers to write current progress to a specific file location by including some progress update statements throughout the setup.sh script; wizard would download the progress report periodically to display | + straightforward implementation for wizard  + easy but cumbersome implementation for each installer  - need to edit and retest all the installers  - changing how finely the progress is reported means editing and rebuilding an installer |
| Rewrite all installers so that every action is a separate function and add a reusable controller script to call each function in turn and update the progress | + improved readability of each installer’s setup.sh script  + easier to identify reusable functions and consolidate across installers  + progress monitoring function with controller is reusable  + easier to implement features such as allowing user to choose if to stop or continue on errors within each installer  - a lot of effort to rewrite and retest each installer |
| **Write an external control script that would run an installer and monitor its progress; the script would write progress to a file and wizard would download that progress report periodically to display** | **+ no need to edit existing installers**  **+ reusable progress monitoring function** |

# Operation

The wizard is a web service based on Mt Wilson Core. It adds the following features:

cit-quickstart-wizard

The quickstart application includes the self-extracting installers for the following services:

* Attestation Service
* Trust Director
* Key Broker
* Key Broker Proxy
* OpenStack Extensions
* Trust Agent

The wizard does NOT install OpenStack, it only installs the Cloud Integrity Technology OpenStack Extensions on an existing OpenStack controller.

## Updating an installer

The wizard stores installers and related files in /opt/cit/packages. Each package is in a separate subdirectory, and the name of the subdirectory is the symbolic package name. This symbolic package name is referenced in the backend code and the UI, so renaming packages must be done in multiple places.

Within each package subdirectory is an installer, a marker file, and an .env file template. Other files may be there also as needed. There may be multiple versions of the installer present but only one may be active at a time. The active installer is designated in /opt/cit/packages/packages.json.

There is just one .env file template per package, but with code changes this could be changed to one per installer if necessary.

There is one marker file per installer, and this file must have the same filename as the installer with a “.mark” suffix.

# Maintenance

## Log Rotation

Log rotation is implemented via the “logrotate” utility. The setup.sh file adds /etc/logrotate.d/cit during installation.

# User Experience

The wizard user interface is based on mtwilson-core-html5. Each wizard feature adds more user interface components.

The mtwilson-deployment-wizard feature adds a tab to the user interface which is the home tab (see blueprint for HTML5).

The wizard guides the user through various screens. Choices in each screen affect what is displayed in subsequent screens.

The wizard presents the configuration options to the user in three screens. First, the user selects the target environment (private data center, cloud service provider, or enterprise subscriber of a cloud service provider), then the features to install, then desired layout of components in the target environment, and finally any required settings and login credentials to the target hosts.

After collecting all required inputs, the wizard presents a preview of work about to be initiated.

After the user confirms the preview is correct, the wizard presents a progress indicator during installation of services.

After deployment is complete, the wizard presents a summary page to the user with URLs, usernames, and passwords to access the installed components.

# Packaging

The wizard is packaged as a self-extracting installer using Ant, Maven, and Makeself.

The application zip file is created using parent mtwilson-core-application-zip from Mt Wilson Core.

The self-extracting installer is created using parent mtwilson-maven-package-makeself from Mt Wilson Core.

There are two Linux self-extracting installer artifacts that are generated by the build process:

* The wizard application only, without components (attestation, trust director, etc)
* The wizard application with components included

The Ant build.xml file automatically generates both artifacts in the “packages” phase.

# Installation

This section describes installation of the CIT deployment wizard itself, not of CIT. The CIT installation is described under “User Experience”.

The system administrator can run the installer with a parameter that indicates all services should be installed on the local host with default options.

Alternatively, the default installation will install the wizard web service from which the administrator can either install all the services on local host, or one or more remote hosts, with customized options. The installer will display in the console the URL for accessing the web service.

# Security

When deploying services to remote hosts, the wizard needs the host address and root password for each one, or an authorized SSH key. Storing these root credentials may create a vulnerability, for now they would not be stored.

# Performance

When deploying installers to multiple remote hosts, the wizard can copy the installers to the remote hosts in parallel (but this isn’t implemented yet). However, the installation must be sequential because, for example, a TLS certificate from Mt Wilson must be generated before it can be copied to Key Broker and Trust Director.

# Upgrading CIT

After a user has installed CIT using the tool, the user may later need to upgrade CIT. This section describes options for accomplishing this.

For all the alternatives below, there would be no facility for tracking upgrade-specific progress markers (could be different messages), would need to develop separate marker files for upgrades and this increases maintenance for every release, or accept less accurate progress tracking (error messages would still work).

Table 2 Wizard deployment alternatives

|  |  |
| --- | --- |
| **Alternative** | **Pros(+) Cons(-)** |
| Install a new CIT deployment tool corresponding to new version | + easy to implement, it’s just a new build of the wizard from the current CIT repository  + new wizard build already includes latest installers of CIT components  + relies on individual CIT installers to upgrade their components properly  + user can reinstall older version anytime by using previously installed older version of the wizard with the older CIT components |
| Update installers in existing wizard | + no development effort  - no facility for upgrading the wizard itself, which may be required to add or update setup tasks related to the new CIT version being installed as an upgrade  + relies on individual CIT installers to upgrade their components properly  - requires more effort from user to identify and copy each installer to its proper location in the wizard |
| Network updates of wizard and installers | - most development effort  - Intel must maintain an Internet site with CIT updates  + least effort for user, wizard downloads updates for CIT installers and for the wizard itself  + relies on individual CIT installers to upgrade their components properly  - one way upgrades (unless we develop a rollback feature by keeping backup older copies of the wizard and component installers when we update) |

After the updated wizard and installers are available using any one of the alternatives above, the next challenge to solve is how to obtain the necessary information for the update.

Table 3 Wizard configuration discovery alternatives

|  |  |
| --- | --- |
| **Alternative** | **Pros(+) Cons(-)** |
| User inputs all configuration | + minimal development  - user needs to keep record of all settings and re-enter them |
| Wizard stores configuration locally | + easy to develop  - requires login & encryption features to protect the data, and this also complicates wizard self-installation  = could skip login & encryption features initially and then at the end of the install ask the user if they want to upgrade later, then ask for password to store configuration  - if user doesn’t predict future upgrade requirement, this would be useless |
| Wizard stores configuration in CIT attestation service | - moderate development effort  + environment, features and layout information stored under a special “wizard” feature of each installed CIT instance  + host root passwords are NOT stored at all  + configuration protected by user having to first enter the IP & root password of any installed CIT instance, then prompting user for root password of each configured host to gain access  + settings already stored in each service and can be detected once root password is provided by user  + could use same design to “clone” CIT servers, for example add another MTW server with same configuration settings on another IP address, etc. |