Tower DB

Application Overview and Design

Clark Ritchie, [clark@inveneo.org](mailto:clark@inveneo.org)

July 12, 2013

Table of Contents

Author 1

Introduction 1

History 1

Features 2

Design 2

Core Technologies 3

Other Technologies 3

JavaScript Libraries 3

CSS 3

CakePHP Plugins 3

CakePHP Behaviors 3

Note on Github Repository Architecture 3

Limitations 4

OpenNMS 4

# Introduction

Tower Database, or Tower DB, is a web application to aid in the design and deployment of long-range Wi-Fi networks. It is primary users are technical staff at Wireless ISPs (or WISPs).

# History

Inveneo, a US 501c3 non-profit, commenced the Tower DB project in the summer of 2012. The project was given the internal code name of “Poundcake”, a term that is used often and can be found throughout the code in comments and other project documentation.

In its original incarnation, Poundcake was to be a short-term project – several weeks of effort to produce a web application for use on a project to manage a large-scale deployment of computer and network equipment. Regrettably, Inveneo’s involvement in that project stopped before Poundcake was put into production. Nevertheless Inveneo decided to continue development of Poundcake and repurpose it for another project, with related, yet sufficiently different, requirements. At that time Poundcake was renamed Tower DB. Certain design decisions related to this repurposing are noted later in this document.

Throughout 2012 and into 2013, Tower DB continued to evolve as an application to aid in the design and management of large-scale wireless networks. Its primary function has since evolved to be:

1. **Network Overview** – Provide a high-level overview of the network, graphically depicting sites and connections between-sites. Provide an overview of the network’s health via integration with network monitoring tools.
2. **Network Inventory** – Be the definitive source for what is installed where, with a facility to capture assorted types of site meta-data, information that is necessary to provide long-term support of a network.
3. **Network Design, Deployment** – Provide tools to support the design and deployment of large-scale wireless networks.

# Features

Tower DB has many features, too many to fully enumerate in this document. Listed below are a few of Tower DB’s major capabilities.

* Multi-project support
* User authentication and authorization, with three permission levels (configurable by project)
* Graphical depiction of a network on a map
* Configurable site states (configurable by project)
* Link wireless radios by SSID
* Support for Point-Point and Point-Multipoint radio configurations
* Lightweight integration with OpenNMS for network monitoring
* Support for a diverse array of site meta-data
* IP Address planning and management
* Support for variable types of radios, routers, switches, antennas
* Workorder generation
* Device configuration file generation
* Produce site-specific equipment lists for deployments
* KML import/export
* Extensive configuration for items such as radio bands, SNMP versions, tower types, and so forth
* Report generation

# Design

Tower DB was intended to be a short-term project, and CakePHP was selected primarily due to familiarity and ease of use. CakePHP was also highly compatible with an existing LAMP environment. Tower DB’s initial developer was on a short timeframe and given the application’s original set of requirements, which were relatively straightforward, CakePHP seemed appropriate.

## Core Technologies

As of the writing of this document, some of the core technologies that Tower DB uses include:

* PHP 5.4.10
* MySQL 5.5.29
* CakePHP 2.3.5

## Other Technologies

Other dependencies that are bundled with the application include:

### JavaScript Libraries

* jQuery 1.9.1 – base jQuery JavaScript library
* jQuery-ui-1.10.3 - JavaScript library for user interface interactions, effects, widgets, and themes
* jQuery-ui-map-3.0 – Google-developed JavaScript library for rendering Google maps
* jasny-bootstrap – JavaScript library for enhanced Bootstrap-like functionality

### CSS

For UI support, Tower DB uses:

* Twitter Bootstrap 2.3.2 – Provides base CSS
* Bootbox – provides dialogs in a Bootstrap way
* Bootswatch – A free Bootstrap-compatible theme site.

### CakePHP Plugins

Tower DB leverages CakePHP’s plugin architecture. Some of the plugins used include:

* AjaxMultiUpload – a component to handle drag-drop file attachments
* Composer – for dependency management in PHP
* Less - converts.less files into .css without relying on Node.js or client-side parsing
* Uploader - plugin that will validate and upload files through the model layer (may be depreceated)
* Utility – a collection of CakePHP utility components
* ExcelWriterXML – for generation of XML files that can be opened in Microsoft Excel
* DOMPDF – for generation of PDF files

### CakePHP Behaviors

Tower DB leverages other small pieces of open source functionality, including:

* Cryptable – A CakePHP behavior to save sensitive data

## Object Model

A graphical depiction of Tower DB’s object model can be found in the Github repository in the folder /etc/docs.

The primary objects in Tower DB include:

* NetworkSwitch – A NetworkSwitch[[1]](#footnote-1) is meant to represent a common layer 2/3 switch used in Ethernet networks.
* NetworkRouter – A NetworkRouter[[2]](#footnote-2) represents a common router used to connect different Ethernet networks.
* NetworkRadio – A NetworkRadio[[3]](#footnote-3) is meant to represent a Wi-Fi radio. The name NetworkRadio was used for consistency with NetworkSwitches and NetworkRouters.
* Antenna – A NetworkRadio can be associated with one of the compatible antennas.
* Sites – A site is the central item to which network devices are attached. A site can have zero to many NetworkRadios, zero or one NetworkSwitch and zero or one NetworkRouter. See comments in *Limitations* below.
* Projects – Projects contain one or many sites.
* Users – Users belong to projects, and have view or edit permission. Users can have mixed permissions (i.e. view permission on one project, edit permission on another). Users who are administrators have access to the all projects in the system, as well as all system configurations.
* [item]Type – Many objects in Tower DB have a corresponding “type”. The type of an object further defines the characteristics of that object, such as the manufacturer, the number of ports, watts consumed, and so on.
  + For example, there exists an object named SwitchType and an administrator may configure any number of valid kinds of switches in the Tower DB application. Which is to say, an administrator of Tower DB may have defined, at the system-level, several types of switches, say an 8-port DC switch, a 16-port AC switch, and so on. When a NetworkSwitch is created, the user picks what kind of switch it is; the NetworkSwitch object has a corresponding SwitchType.
  + Other examples of the [item]Type model:
    - A NetworkRouter has a RouterType
    - A NetworkRadio has a RadioType
    - An Antenna has an AntennaType
    - And so forth…

## Note on Github Repository Architecture

Note that none of the aforementioned technology dependencies are configured in the Github repository as Git submodules. Doing this is of interest. Re-architecting the Github repository, and modifying the deployment process, has simply not been a priority.

# Limitations

## Many Radios, One Switch, One Router

## OpenNMS

Tower DB’s integration with OpenNMS is extremely primitive, and should be considered proof of concept.

* Provisioning – Provisioning an item (radio, router, switch) from Tower DB into OpenNMS (as an OpenNMS node) for network monitoring is functional but severely lacks in robust error checking. For example, items can be repeatedly provisioned with each provision creating a new node with a new unique Foreign ID.
* Rendering OpenNMS graphs in Tower DB is extremely simplistic due to OpenNMS not exposing graphing capabilities via its ReST API.
* Alarms are functional but only displayed when a user clicks on Alarms.
* Configuring Tower DB to query OpenNMS remains a manual task. Tower DB’s ability to ascertain a node’s status is best done on current releases of OpenNMS due to enhancements in the OpenNMS ReST API.

1. The name NetworkSwitch was used because “switch” is a PHP keyword. [↑](#footnote-ref-1)
2. The name NetworkRouter was chosen because “router” is a CakePHP keyword. [↑](#footnote-ref-2)
3. The name NetworkRadio was chosen simply for consistency with NetworkSwitch and NetworkRadio. [↑](#footnote-ref-3)