|  |
| --- |
| McAfee |
| VmiDC High-Level Design Spec |
| Draft |

|  |
| --- |
| VmiDC Team  10/3/2013 |

Contents

[1. Architecture 2](#_Toc368575431)

[2. Components 3](#_Toc368575432)

[2.1 User Interface 3](#_Toc368575433)

[2.1.1 Vaadin Framework 3](#_Toc368575434)

[2.1.2 Session Management 5](#_Toc368575435)

[2.1.3 User flow, Views, Widgets and Components 6](#_Toc368575436)

[2.1.4 Screen shots of implementation 7](#_Toc368575437)

[2.1.5 UI controller layer 8](#_Toc368575438)

[2.2 Service Layer 9](#_Toc368575439)

[2.3 Persistence Layer 9](#_Toc368575440)

[2.4 Internal vmiDC Callback APIs 9](#_Toc368575441)

[2.5 Business Logics and Integration 9](#_Toc368575442)

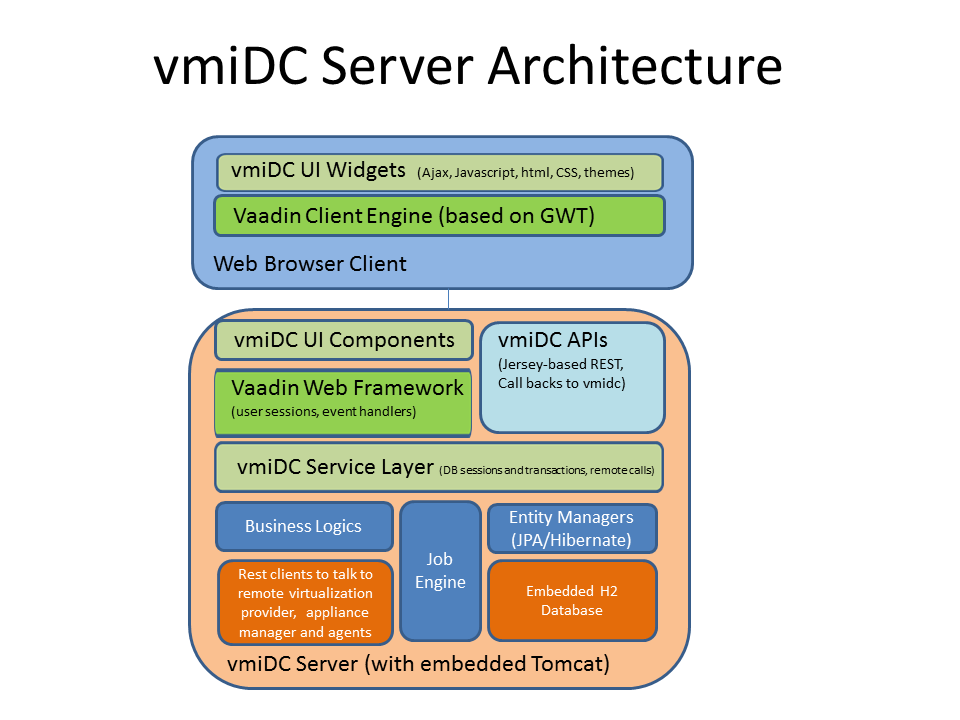
[2.6 Job Engine and Workflows 9](#_Toc368575443)

[3. Install and Upgrade 10](#_Toc368575444)

[4. Data Modeling: 11](#_Toc368575445)

[4.1 Data Model for Virtualization Connector: 11](#_Toc368575446)

# Architecture



# Components

## User Interface

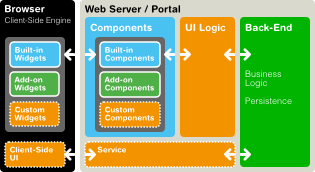
### Vaadin Framework

#### Introduction:

Vaadin Framework is a Java web application development framework that is designed to make creation and maintenance of high quality web-based user interfaces easy. Vaadin supports two different programming models: server-side and client-side. The server-driven programming model is the more powerful one, and essentially lets you forget the web and program user interfaces much like you would program any Java desktop application with conventional toolkits such as AWT, Swing, or SWT.

In VmiDC project we are using server-driven programming model to implement required user interface. This approach allows us to create web pages using POJO and CSS. We are using Vaadin base theme and are overriding it with our own resource settings (using CSS).

#### Application Architecture

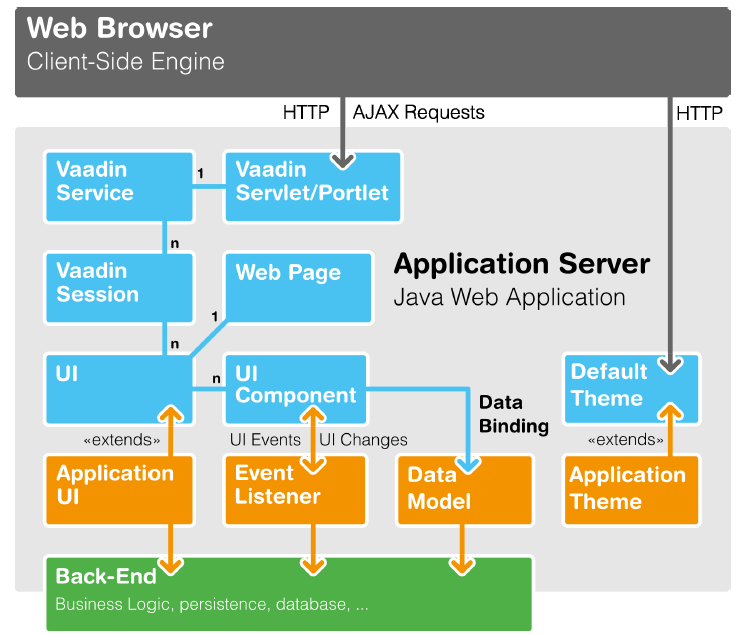


The above diagram illustrates the basic architecture of server-side web applications made with Vaadin. This architecture consists of the server-side framework and a client-side engine that runs in the browser, rendering the user interface and delivering user interaction to the server. The user interface of an application runs as a Java Servlet session in a Java application server, and the client-side engine as JavaScript.

As the client-side engine is executed as JavaScript in the browser, no browser plugins are needed for using applications made with Vaadin. This gives it an edge over frameworks based on Flash, Java Applets, or other plugins. Vaadin relies on the support of Google Web Toolkit for a wide range of browsers, so that the developer does not need to worry about browser support.

In VmiDC project we are using server-driven programming model to implement user interface. This approach allows us to create entire user interface using POJO and CSS. We will be using default Vaadin theme and will override it with our own resource settings. Behind the server-driven development model, Vaadin makes the best use of AJAX (Asynchronous JavaScript and XML) techniques that make it possible to create Rich Internet Applications (RIA) that are as responsive and interactive as desktop applications.

#### VmiDC UI Framework Architecture



The above diagram illustrates detail architecture of implementing VmiDC UI framework using Vaadin framework.

### Session Management

We will use session management provided by Vaadin framework. We can set session attributes in Vaadin application similar to the following code snippet:

// Save to VaadinServiceSession

ui.getSession().setAttribute("myValue", value);

// Save to HttpSession

VaadinService.getCurrentRequest().getWrappedSession()

.setAttribute("myValue", value);

// Save to VaadinSession

VaadinSession.getCurrent().setAttribute("user", u.getLoginName())

Vaadin offers two ways to access the UI object: with getUI() method from any component and the globalUI.getCurrent() method.

Example of getting values using both the methods:

* getUI()

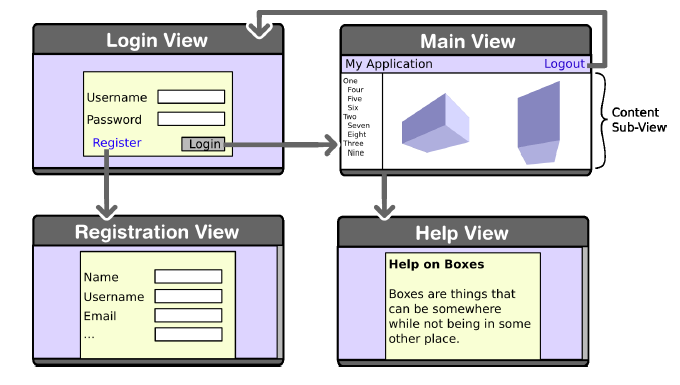
data = ((MyUI)component.getUI()).getUserData();

* getCurrent()

data = ((MyUI) UI.getCurrent()).getUserData();

### User flow, Views, Widgets and Components

#### Introduction:



The above diagram illustrates sample user flow of our web application. User can register/login using login view. After a successful logon user will see the main view which is the home page of our application. This page consists of a Navigator which will allow user to navigate between sub views and Main CRUD toolbar. It also consists of a Sub-View layout which is changes according to the selected sub-view.

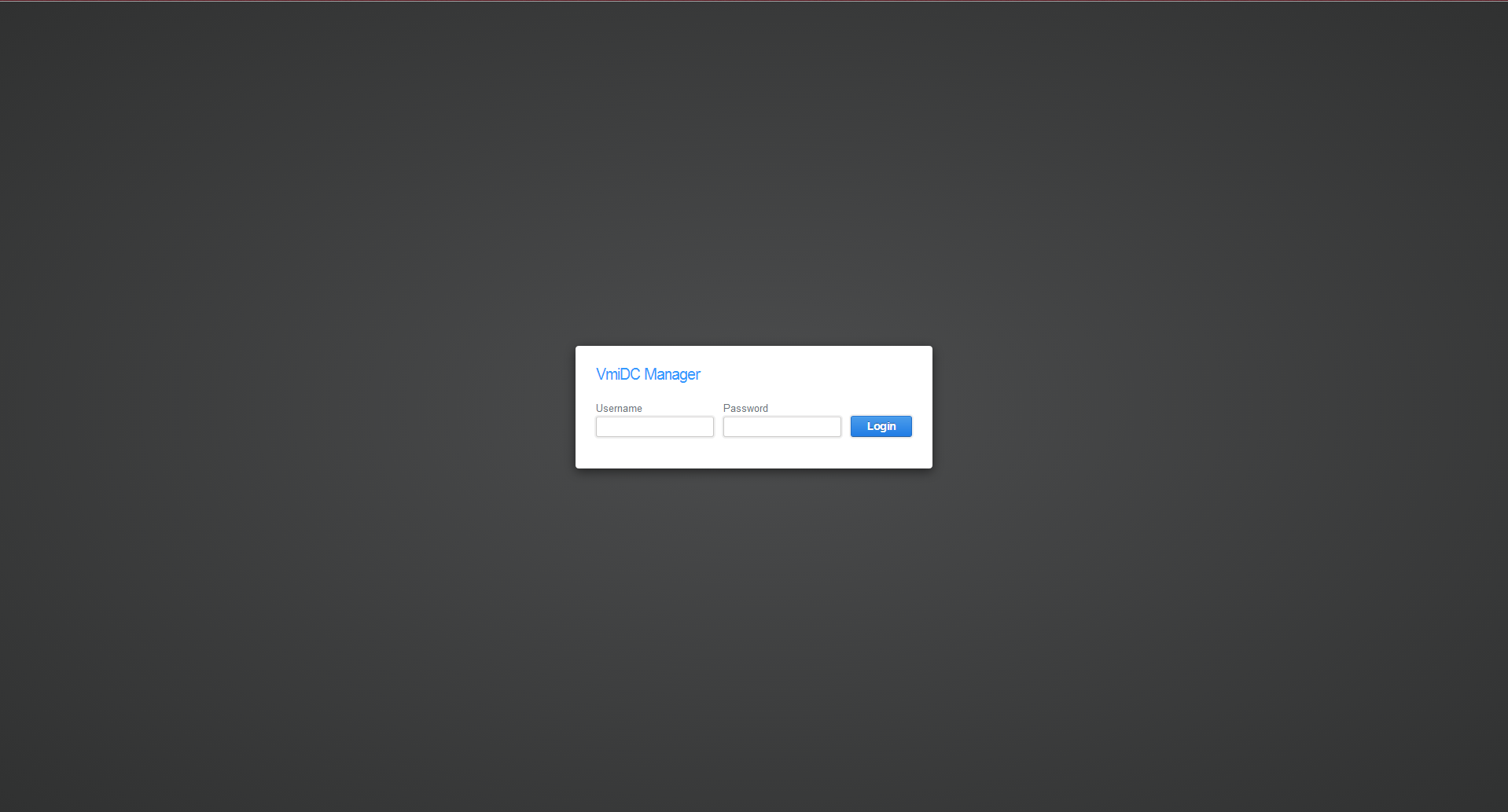
#### Views:

VmiDC web application consists of the following views and sub views:

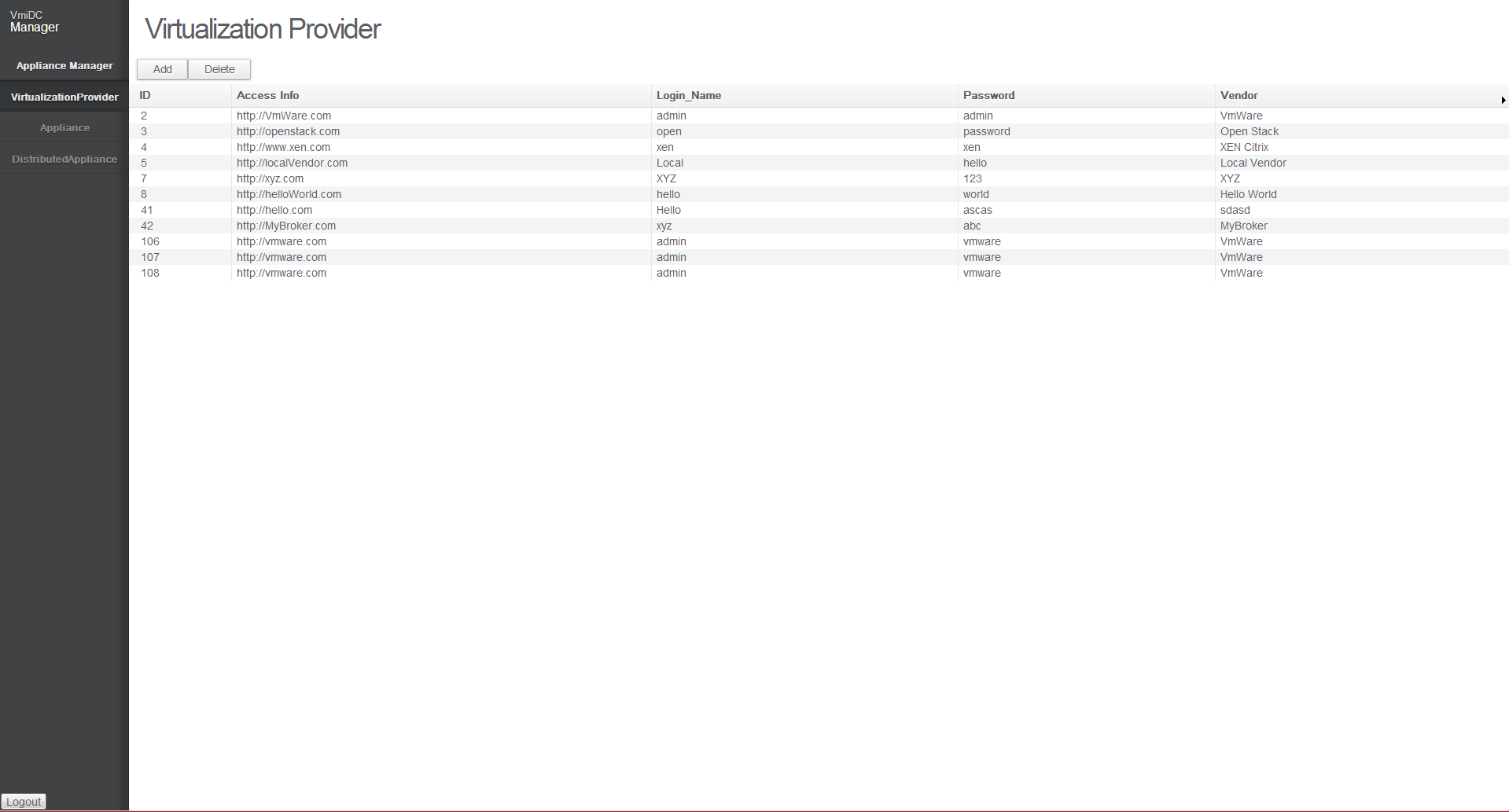
* **Login View**
  + Register View (Sub- View as a new window)
* **Main View**
  + Virtualization Manager Connector View
    - Add View (Sub- View as a new window)
    - Update View (Sub- View as a new window)
    - Delete View (Sub- View as a new window)
    - Reload View (Sub- View as a new window)
  + Virtualization Provider Connector View
    - Add View (Sub- View as a new window)
    - Update View (Sub- View as a new window)
    - Delete View (Sub- View as a new window)
    - Reload View (Sub- View as a new window)
  + Appliance View
    - Add View (Sub- View as a new window)
    - Update View (Sub- View as a new window)
    - Delete View (Sub- View as a new window)
    - Reload View (Sub- View as a new window)
  + Distributed Appliance View
    - Add View (Sub- View as a new window)
    - Update View (Sub- View as a new window)
    - Delete View (Sub- View as a new window)
    - Reload View (Sub- View as a new window)

### Screen shots of implementation

#### Login View (POC)



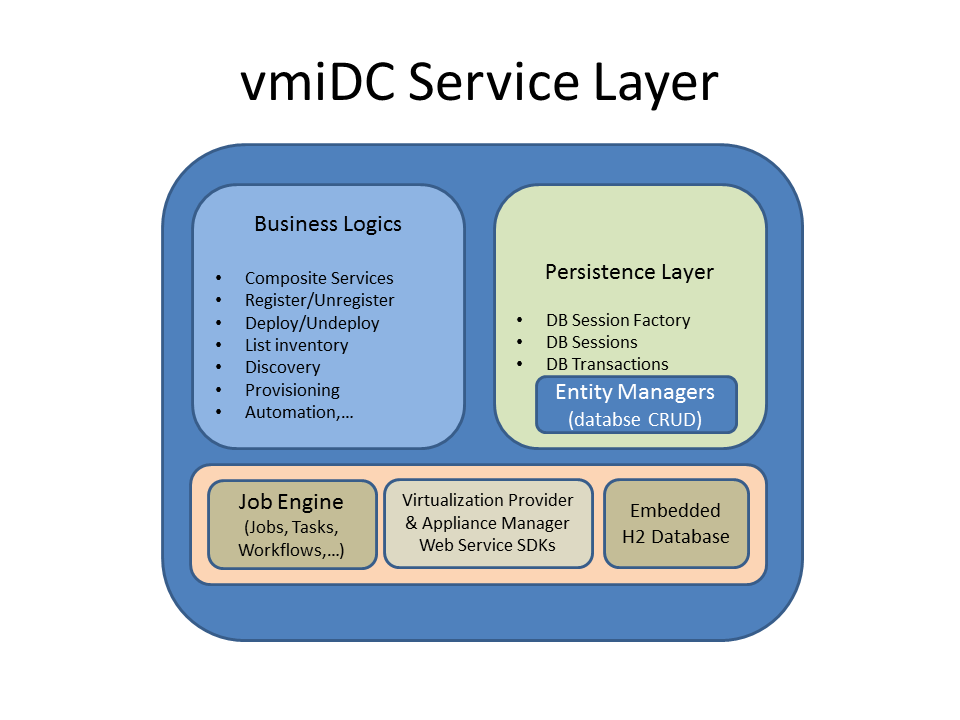
#### Main View (POC)



### UI controller layer

UI controller is an abstract layer to communicate with business logic. This module binds user interface implementation with hibernate entities. This controller provides a generic interface to the UI components to communicate with the service layer.

## Service Layer



* Database sessions.
* Database transactions.
* Invoke Entity Managers to access or persist data in database.
* Encapsulate business logics.
* Use Jersey-based REST or SOAP clients to communicate to virtualization providers, appliance managers, and appliance agents.

## Persistence Layer

* Use embedded H2 database.
* Entity Managers.
* Hibernate session factory.
* JPA based annotations and Hibernate ORM.
* For database schema, refer to Data Modeling session below.

## Internal vmiDC Callback APIs

* We will use Jersey framework to publish our callback Restful APIs.
* API details:

## Business Logics and Integration

* Integration with appliance managers.
* Integration with virtualization providers.
* Integration with appliance agents.
* Registration, deployment, provisioning, and monitoring flows and scenarios.
* Alert response actions.
* Policy and tenant interactions with appliance managers.
* We will use Jersey-based Rest clients to talk to NSM and management agents.
* We will use open source vijava to talk to VMWare vCenter/ESX hosts.
* We will use NSX SDK to talk to VMWare NSX.

## Job Engine and Workflows

* Job and task definitions and interfaces.
* Job engine design.

# Install and Upgrade

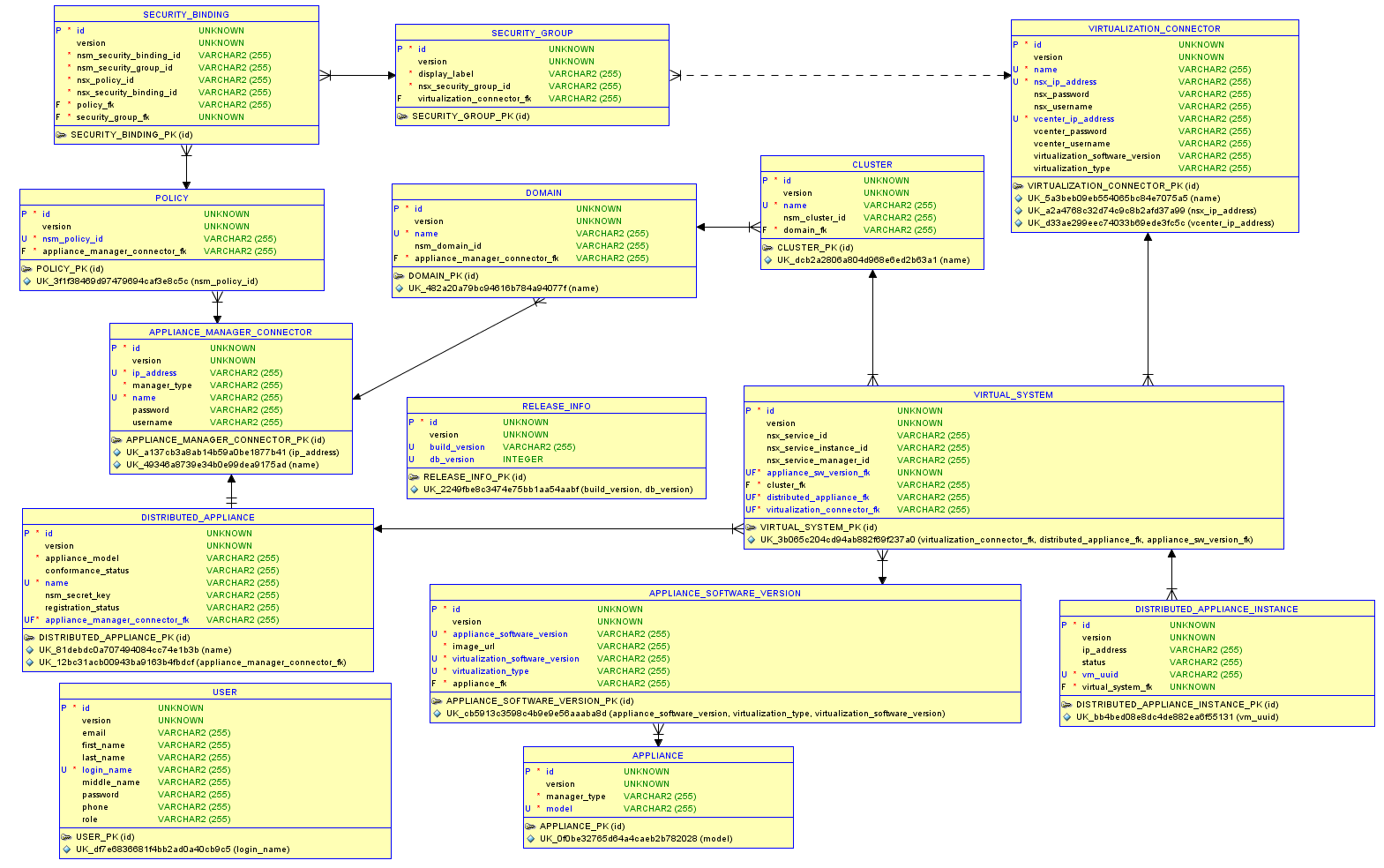
* Database schema rebuild and data migration.

# Data Modeling:

## Data Model for Virtualization Connector:

ER Diagram:

We have to persist service ID for sequence of callbacks.



Class Diagram:



Notes:

* For ApplianceConfiguration table:
  + ResourceSettings have the following fields: cores, memory, disk\_size, nics
  + ApplianceSettings have the following fields: max\_throughput, license.