servicenow

White Paper

# CMDB Design

# Table of Contents

Abstract ..... 3
CMDB Requirements ..... 3
Governance Requirements ..... 4
Inventory and Asset Requirements ..... 4
Incident, Problem, and Change Management Requirements ..... 4
Service Management Requirements ..... 5
CMDB Design ..... 5
Define CI Classes ..... 5
Define CI Levels ..... 7
Define CI Attributes ..... 7
Define CI Relationships ..... 9
Putting It All Together ..... 9
Summary ..... 10

# CMDB Design White Paper

**Audience:** Configuration Management Process Owner, Configuration Manager, Configuration Analysts, and Key CI Owners


## Abstract

This white paper is intended to provide guidance and advice on how to design a CMDB, from defining requirements to development of the data model. It assumes that the Configuration Management leadership team has already articulated the goals, benefits, expected uses of the CMDB, and policies that will govern its scope.

Scoping decisions are often dictated by policies that address an organization's business drivers, contractual obligations, service commitments, governing laws, regulations, and standards. Required scoping decisions include:

- What environments will Configuration Management control?
- Which CIs in the CMDB need to be managed at the relationship level and which CIs require only Inventory- or Asset-level management?
- What IT services will be included?
- Are there geographic considerations to take into account?
- Are there regulatory or compliance requirements that must be met?
- Are there specific levels of control required for traceability and auditability?
- What security issues must be addressed?
- Are interfaces to internal and external service providers required?

Many of the answers to these questions may be stated as policies that will govern the design and development of the Configuration Management system, if they do not already exist.


## CMDB Requirements

As the 'system of record' for all information about the IT infrastructure, the CMDB is a foundational tool that supports the effective execution of all service management processes. Because of this, gathering requirements from all stakeholders prior to designing the CMDB is a critical first step for success. At a minimum, the following requirements must be determined.

As the 'system of record' for all information about the IT infrastructure, the CMDB is a foundational tool that supports the effective execution of all service management processes.

A CMDB enables you to create a service dependency map so that you can relate events, incidents, problems, and requests for change to service impact.

### Governance Requirements

The CMDB can be a very powerful tool in helping meet various regulatory and governance requirements that affect the IT organization. For example, the Sarbanes-Oxley Act is a federal mandate that requires evidence of controls for processes that affect financial reporting. There are many industry-specific regulations that will also dictate specific CMDB requirements (e.g., HIPAA controls for patient information, GDPR controls for European Union data protection). The CMDB team must gather these specific requirements from the appropriate governance stakeholder(s), such as the internal audit team, legal counsel, or an IT representative responsible for IT governance and regulatory control.

### Inventory and Asset Requirements

If improved IT asset management is one of the goals of your CMDB initiative, then IT asset management-related requirements will need to be defined. Inventory, asset, and configuration management functions all use a common subset of attribute data for each CI. However, each function has different additional requirements for different CI management purposes. For example, inventory-related CI data usually describes the unique physical properties and location of the CI (e.g., serial number, model number, location, and owner). Inventory management applications track ownership, location, condition, lifecycle, and stock levels. Asset-related CI data focuses on financial and governance information such as cost, present value, maintenance cost, contract terms, depreciation, and replacement information. Configuration management, on the other hand, requires information about how CIs are related to one another. Relationship information enables incident troubleshooting and change impact analysis. As a result, incidents can be resolved faster and change requests can be reviewed and approved more reliably.

In ServiceNow, inventory, asset, and configuration management applications all leverage a single CMDB. For the implementation to be successful, however, the requirements for these functions need to be defined upfront and the CMDB must be designed to accommodate them. Some of the information required by these functions probably already resides in other databases. In this case, you will need to decide whether to replicate that data in the CMDB or reference it from the original source on behalf of the CMDB. While replication enables easier reporting and ad hoc searching, it requires controls to ensure timely synchronization of data.

### Incident, Problem, and Change Management Requirements

A CMDB enables you to create a service dependency map so that you can relate events, incidents, problems, and requests for change to service impact. To determine the specific CI information required to support these capabilities and build organizational buy-in, you will need to involve the right stakeholders. This stakeholder group should include the process owner, process manager, and other subject matter experts for each of the processes. At a minimum, consider the following required support capabilities:

- Incident management
  - Access to support team information for resolution responsibility assignment
  - CI end-to-end service mapping for quicker incident troubleshooting and resolution
  - Access to resolution targets or SLAs and business impact information for appropriate priority and escalation assignment
- Change management
  - Ability to associate the affected CI to the request for change for impact assessment purposes (e.g., maintenance/outage windows, SLAs, etc.)
  - CI end-to-end service mapping to understand potential upstream or downstream impacts of a change

One of the most important and challenging areas of CMDB design is choosing the right CI classes, levels, attributes, and relationships.

A general understanding of the processes' current maturity level and any goals or plans to increase maturity should also be taken into consideration when prioritizing the processes that will influence CMDB requirements.

**Service Management Requirements**

If a CMDB initiative goal is to enable improved services, then the CMDB requirements must include service details and relationships. The CMDB is a great repository to hold the breakdown of the service offerings and show the linkages with the underlying infrastructure. Work with the people who own the various services described in the service catalog to identify the monitoring and reporting information needed to improve service levels. Stakeholders could include service planners, portfolio managers, program or service owners, business relationship managers, and service level managers.

The general approach for using the CMDB to support a service catalog is to create service CIs in the CMDB that correspond to the services in the catalog. Service CIs can be further sub-divided and linked to underlying infrastructure CIs, creating a service hierarchy. The first step is to identify the types of service offerings provided. These types typically include business services that are ready to be consumed, and IT services that must be combined with other service elements to form a business service. Select at least one service from each type of offering, and then break it down to the underlying infrastructure level. This information will be used to determine the CMDB service structure and level design, and it will help determine the types of CI relationships needed.

Service level agreements must also be taken into consideration. Many CMDB design efforts identify adequate attribute information to manage the technology and the infrastructure, but they often overlook important service management attributes such as:

- Service level targets and priority
- Service operational information (hours of service availability, required approvals, etc.)
- Service-related notification and communication information
- Service owners

## CMDB Design

One of the most important and challenging areas of CMDB design is choosing the right CI classes, levels, attributes, and relationships. This requires the right balance between controls, information availability, and the cost and effort required to maintain the data. That balance is achieved when the IT business process and service management requirements for your organization are met. Too much information will create unnecessary cost and effort to maintain and could possibly undermine the acceptance and use of the CMDB, while too little information will result in an inadequate level of control and an inability to optimize service performance. Determining the appropriate CI classes, levels, attributes, and relationships is an iterative process. A solid understanding of these four dimensions of CMDB development is required before finalizing the CMDB data model.

**Define CI Classes**

The CMDB is organized by a class structure where each class is a predefined, standardized category containing similar CIs. Each class can be further divided into sub-classes through parent-child relationships, where children are specializations of their parent. For example, the top-level class of hardware may be divided into sub-classes of computer, network gear, printer, etc. The further down this hierarchy, the more specialized the components become.

Use classes to refine searches and queries through optimized filtering. IT components may be physical, logical, or conceptual.

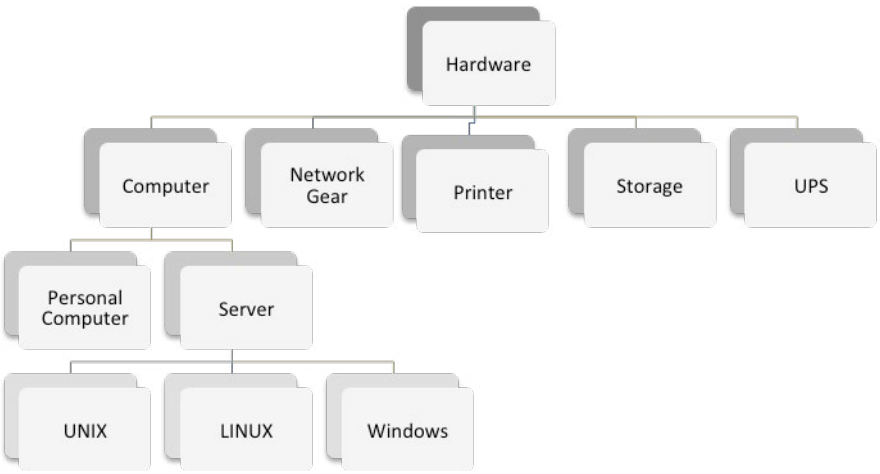


Figure 1: Example Hardware Class and Sub-Class Structure

Use the following guidelines when defining your CMDB class structure:

- The more top-level classes in the CMDB, the more challenging it may be to search and find CIs. Limiting the number of classes at the highest level of your CMDB will simplify the search criteria.
- Use classes to refine searches and queries through optimized filtering. IT components may be physical, logical, or conceptual.
  - *Physical* components have a specific location, take up space, and can be seen (e.g., servers, desktops, network devices, etc.).
  - *Logical* components do not take up physical space. They perform specific functions requiring physical components in order to operate (i.e. database instance, software release, corporate intranet, etc.).
  - *Conceptual* components are representations of physical and logical components that have been combined to create a unique concept (i.e. business service, system, environment, etc.).

Some stakeholders may be interested only in the physical CIs and therefore would benefit from classes that represent only physical assets, for example.

- Create sub-classes only when unique data exists. For example, in Figure 1, the class 'Personal Computer' could be further sub-divided into desktop, laptop, workstation, tablet, etc. However, creating a new class for each of these types of personal computers may not be necessary when they all have similar data requirements. A better solution may be to add a 'Type' attribute to the 'Personal Computer' class to identify each of the specific types (e.g., desktop, laptop, etc.).
- Use auto-discovery tools to populate your CMDB with physical (and some logical) CIs. With well-defined CMDB classes, you can simplify the auto-discovery linkages. You will need to enter conceptual (and some logical) components manually. The good news is that these components tend to change far less frequently than physical components do.
- Standardize the highest-level classes to ensure that all components conform to a consistent taxonomy. Without consistency, use of the CMDB becomes very difficult. The ability to correlate CIs is diminished making interrogation and analysis results less effective.

Routinely review your CI levels to ensure that information is at the appropriate level to be useful and accurate.

### Define CI Levels

Levels refer to the depth of CI classes you choose to include in your CMDB. A CI can be a single component or a complete system. For example, a CI could be a workstation, or each component of the workstation (e.g., the processor, monitor, keyboard, and mouse). When determining the depth of CI class levels, you will need to answer questions such as: Should a workstation be a CI, or should there be a CI for each workstation component? Should the Microsoft Office Suite be one CI or should there be one CI for each component of the suite, such as Word, Excel, PowerPoint, etc.? Criteria to consider include:

- **Cost:** Is the cost of the component such that it needs to be tracked individually? For example, an expensive graphics card could be a CI even if it resides in a workstation.
- **Value:** Is the value of adding the component less than the overhead to create and manage it? If so, then don't add it. Instead, represent the additional detail as one or more attributes of the parent CI.
- **Change considerations:** Are changes to the component made frequently, especially location changes?
- **Traceability:** Does the component need to be traced for audit reasons?
- **Governance and compliance requirements:** Is this CI impacted by Sarbanes-Oxley compliance? Is it subject to contract and usage compliance?
- **Delivery quality:** What level of CI is sufficient to record incidents and changes and to support effective problem management activities?
- **Management of service commitments:** Is the component a critical part of a service offering that you would need to consider when analyzing the impact of a change?

Best practice suggests that changes should be made at the CI level recorded in the CMDB. For example, if you decide to record software CIs down to the detailed level of each module, then the change would be made and recorded at that level. However, if the CMDB software CI leveling extends only to the level of the program, then a change at the module (component of the program) level would require the whole program to be recompiled and the change recorded at that level.

Routinely review your CI levels to ensure that information is at the appropriate level to be useful and accurate. However, at the beginning of your CMDB initiative, try to identify the lowest level of CI class you will require and create it in the CMDB even if you do not plan to use it right away — this will minimize the need to redesign the CMDB later.

### Define CI Attributes

Attributes are data elements that describe CIs. They help to identify and detail the characteristics of CIs that are needed to manage and support services such as what is in use, the status of the items, and their location. Samples of hardware CI attributes could include make, model, serial number, location, version, license number, etc.

Review the CMDB requirements previously gathered to identify the specific CI attributes needed. Take measures to ensure that attributes are only added to the CI if they are required to manage and support your IT services. Also, consider whether the information you need already exists in other records (such as incidents) and whether it needs to be replicated in the CI record. Start by defining attributes for the highest-level CI classes. Many CI attributes can be inherited from their parent CI or passed on to their child CI.

A critical component to enable sound CI level and attribute design is an understanding of the ServiceNow CMDB class inheritance model. CMDB classes are groups of CIs that share the same characteristics, and the CMDB class structure provides the building blocks in the form of tables and columns or classes and attributes. The CMDB begins with a base or parent class where all common CI attributes are defined. The base class is called 'Configuration Item' and exists in a database table called cmdb_ci.

Classifying CIs using the concept of inheritance will help you consistently define your CMDB data model and simplify the management of CI attributes.

The Configuration Item class contains dozens of attributes that can be associated with any configuration item, whether hardware or software, physical or logical. An example of an attribute defined in the Configuration Item class is 'Description'. Rather than requiring a unique description field or attribute to be created for every CI class level in the CMDB, cmdb_ci defines the Description attribute and all sub-classes that extend Configuration Item inherit and can use that attribute.

An extending table or child class adds attributes that are unique to that particular class. For example, the Unix Server class extends the Server class, which extends the Computer class, which extends the Hardware class. The Hardware class extends Configuration Item, and every attribute available in its four 'parent' classes is available to the Unix Server class in addition to any unique attributes added to it.

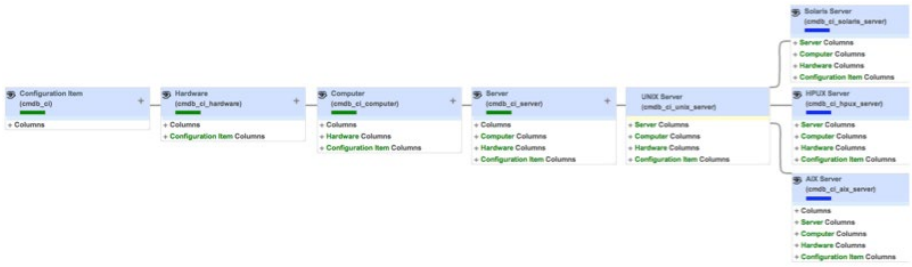


Figure 2: Schema Map for the Unix Server Class

Taking this example further, many organizations employ Unix Servers from more than one manufacturer. The CMDB addresses this by including unique classes that extend the Unix Server class for HP-UX, Solaris, and AIX Servers, and each of those classes inherits the attributes from its five 'parents'.

If a suitable CMDB class doesn't already exist for a particular type of CI in your organization, a new class can easily be created using Tables and Columns. The best practice when adding custom tables to the CMDB is to extend the most logical parent class to benefit from class inheritance. For example, the CMDB contains a Load Balancer class that extends the Server class. The Load Balancer class has a child class for F5 BIG-IP load balancers. If your organization leverages load balancers from Barracuda Networks, then it would be most appropriate to create a new class that extends from the Load Balancer class.

Some useful opportunities emerge when you apply the concept of inheritance to attributes. Think of them from these three perspectives:

| Type | Description | Examples |
|---|---|---|
| Root-level | Mandatory attributes that apply to every CI in the CMDB | • Name<br>• Description<br>• Status |
| Class-level | Attributes that are unique to the specific CI class | Server<br>• Manufacturer<br>• Model<br>• Serial number |
| CI-level | Attributes that are unique to a specific CI instance Server NT00490 | Server NT00490<br>• NERC regulation code<br>• Last audit date<br>• Security clearance level |

Classifying CIs using the concept of inheritance will help you consistently define your CMDB data model and simplify the management of CI attributes.

Without relationships, it is impossible to understand how assets are combined to deliver an end-to-end IT service that provides value to the business.

**Define CI Relationships**

Defining the relationships between CIs is what distinguishes a CMDB from an asset repository. Without relationships, it is impossible to understand how assets are combined to deliver an end-to-end IT service that provides value to the business. It is this relationship data that makes the CMDB a powerful decision support tool. Understanding the dependency relationship among your CIs can tell you, for example, exactly who and what are affected by the loss of a bank of disk drives. When you find out that a router has failed, you will be able to assess the effect of that outage. When you decide to upgrade the processor in a server, you can tell who or what will be affected during the upgrade outage.

In many cases, the relationships between CIs can be automatically discovered. If you use ServiceNow's Discovery product, many relationships will be automatically loaded into the system through the discovery process. Likewise, if you pull your CMDB data from another system, you may receive some form of relationships as part of your import. Regardless of the data source, you may want to augment these automated relationships with others that you define.

ServiceNow's CI relationship editor makes it easy to do this and provides more than a dozen relationship types to choose from (e.g., used by, runs on, connects to, depends on, etc.). Take caution when choosing relationship types as too many may become unmanageable and confusing to users. A recommended best practice is to start with the generic depends on relationship. This approach allows you to define a basic dependency map where relationships can go both upstream and downstream from a given CI. This relationship type enables the Business Service Map (BSM) in providing a graphical representation of the upstream and downstream dependencies for the given (root) CI. This functionality supports incident troubleshooting, problem root cause analysis, and change impact analysis — functions that are critical to the successful operation of the Incident, Problem, and Change management processes.

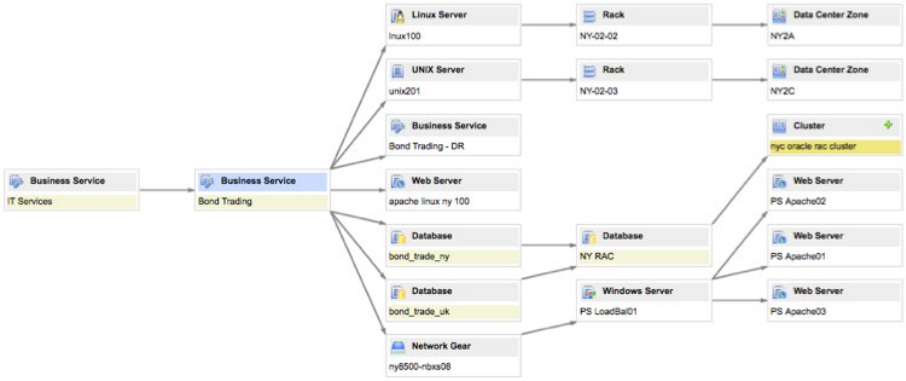


Figure 3: Example Business Service BSM Map

**Putting It All Together**

After defining the CI classes, levels, attributes, and relationships, the next step is to integrate these details into a final model. Many organizations prefer to represent their CMDB model in a diagram or 'picture' that is a visual representation of the CMDB structure. There are a myriad of sample CMDB model diagram structures that can be used for this activity and a quick Internet search or query to the ServiceNow Community will surely result in one that will work for your organization. At a minimum, the CMDB model can be represented in a list format that contains three columns of information: the CI class structure (i.e., levels), CI attributes, and the relationships.

Validating the model with your stakeholders will help to guide any final refinement, provide planning input for the CMDB construction and population effort, and ensure stakeholder understanding of the model.

The model must then be validated against the entire set of requirements identified at the start of your CMDB design initiative. Validating the model with your stakeholders will help to guide any final refinement, provide planning input for the CMDB construction and population effort, and ensure stakeholder understanding of the model. To do this, work with your stakeholders to create a series of specific use cases based on their requirements and CMDB needs. Running through these real-world scenarios will ensure that the CMDB structure, class levels, attributes, and relationships are sufficient to meet both the service management and IT business process requirements. After completing this step, you can proceed with building your CMDB, with the confidence that it will support the needs of your organization.

## Summary

By engaging key stakeholders and utilizing clearly articulated requirements, you have ensured a comprehensive and extensible CMDB design for your organization. This model will be the authoritative source for CMDB construction and population planning. Further, when coupled with effective control mechanisms, it will ensure that the CMDB contains the critical information you need to manage your services and infrastructure today and into the future.

servicenow

www.servicenow.com