DESIGN GUIDANCE

CMDB Design Guidance

Oct 2020

servicenow.

Introduction

Welcome!

In this guide, you'll find practical advice on how to design and deploy your Configuration Management Database (CMDB). We'll cover overall design principles, specific implementation steps, and best practices for rolling out and maintaining your CMDB. By following this guidance, you'll maximize the business value of your CMDB, get better results faster, and avoid common pitfalls.

This guide is not a marketing document. It assumes you already know the value of a CMDB and are looking to get started on implementation. If you do need a better understanding of the benefits of a CMDB or want help explaining these benefits to your organization, please talk to your ServiceNow account representative.

One word of advice. Some people worry that implementing a successful CMDB is a complex—or even overwhelming—task. It isn't. CMDB implementations typically fail because people aren't clear on their objectives or try to go for a "big bang" approach. If you take a structured "crawl, walk, run" approach, you'll begin to see benefits very quickly.

That's what this guide will help you to do. Let's get started.

A CMDB refresher

Before we dig into design principles, implementation steps, and operational best practices, let's take a moment to review the basics. What is a CMDB, and how does it work? Although this guide assumes that you know the value of a CMDB, it's important to clearly understand its role. This will help you to take full advantage of your CMDB while avoiding the consequences of poor implementation.

The purpose of a CMDB

The purpose of a CMDB is to provide accurate and reliable information about digital services and the infrastructure that supports them. In essence, it is the content management system (CMS) for ITIL service configuration management—formerly service asset configuration management (SACM).

Now that we've got the formal definition out of the way, let's tackle a common misperception. Because the CMDB is a configuration management database, people often assume that its primary purpose is to support change management. However, the CMDB's true purpose is to help you deliver data-driven business outcomes, both within IT and more broadly across the business. It has little or no value if it is just a master data repository—to create value, you have to do something useful with the data.

Let's look at a common scenario: a service outage. An application isn't responding, and hundreds of users are affected. How do you figure out what's wrong and get the service back up and running quickly? Has someone upgraded the application recently? Is it the server, or is there a problem with a database halfway around the world? Is there a network problem? So many possibilities, and so little time. Without a CMDB, you don't even know the application version, which server it's running on, or which databases it's talking to. You spend hours gathering information—assuming that you can find it—and meanwhile the clock is ticking. At this point, the value of a CMDB becomes painfully obvious. It gives you instant access to the information you need to quickly resolve the service issue, including infrastructure relationships, service topologies, change histories, software versions, and more.

This is just one example. A CMDB can help you to drive positive business outcomes across a wide range of operational processes. It can also support and improve many other business areas—for example, it can help you to reduce service delivery costs, maximize the ROI of your application portfolio, and accelerate time-to-market for new services.

Configuration items, attributes, and relationships

Exactly what type of information does a CMDB contain? Here's a quick recap:

- As we just said, a CMDB provides accurate and reliable information about digital services and the infrastructure that supports them.
- To do this, the CMDB stores information that describes components—"things" —in your operational environment that are involved in the delivery of digital services. Each component is represented as a configuration item (CI).
- Each CI has multiple attributes that contain specific information about the component the CI represents. CI classes are arranged in a class hierarchy, with each subclass extending the attributes of its parent class. For example, a Linux Server CI class inherits all of the attributes of its parent Server class and adds Linux-specific attributes.
- CIs also have relationships. For instance, an application can run on a server, or a web server can depend on an upstream load balancer. The CMDB stores these relationships between CIs, along with the relationship type (depends on, runs on, etc.).

- Note that these relationships are not the same as the class hierarchy—the hierarchy specifies inheritance between CI classes, whereas relationships are between CI instances (e.g. web server “X” depends on load balancer “Y”).
- CIs can be broadly grouped into two main categories—infrastructure CIs and service CIs.
- Infrastructure CIs represent capabilities provided by physical or logical components such as servers, routers, application software, cloud resources, and so on.
- Service CIs represent digital services and are supported by infrastructure CIs. In ServiceNow, there are three types of service CIs, representing business, application, and technical services.
  - A business service supports a business capability—such as managing customer orders—and is consumed by business users. Business services are typically underpinned by one or more application services (i.e. a business service can include multiple applications).
  - An application service is a full application stack—for example, a stock inventory system that supports the customer order management business service described above. It isn’t just the application—it’s all of the distributed components that make the application work. This is what we typically think about when we talk about a digital service.
  - A technical service is a technical capability that underpins one or more application services. For instance, multiple application services—including the stock inventory system above—could all use a common storage service.

Four more things to remember about configuration items

We’ve already talked about infrastructure and service CIs. Here are four more things about CIs that you need to know before you start out on your CMDB journey.

- **CIs must have a unique name that doesn’t change.** The name needs to be unique so it can be differentiated from other CIs, and it mustn’t change so the CI can be tracked over time. For example, using a hostname to uniquely identify a server is fine provided that the hostname doesn’t change. On the other hand, it’s a bad idea to use an IP or MAC address to identify a server as this can easily change—for example, if you’re using dynamically assigned IP addresses or need to swap out a network interface.
- **CIs must have relationships.** An infrastructure CI represents a component that needs to be managed to deliver or support a service. In other words, each infrastructure CI has a direct or indirect relationship with one or more service CIs. For example, a service could have a “depends on” relationship with a web server CI, while the web server in turn has a “runs on” relationship with a Linux server CI.
- **CIs and assets are related, but they’re not the same.** For instance, consider a physical server that you have purchased. This is tagged and recorded as an asset. Once the server becomes operational, a CI record is created in the CMDB to represent the operational capability provided by the server. The CI —not the asset record—is then referenced in incidents, change requests, and so on. CIs can also exist without corresponding assets—for example, a cloud-based server is represented in the CMDB as a CI, but there is no corresponding asset record since the server does not physically exist.
- **Don’t create CIs for things you can’t configure or monitor.** Ask yourself whether a CI could be the subject of an incident or change request. If the answer is no, then don’t create the CI. More specifically, don’t create CIs for passive datacenter components such as racks. The CMDB is there to help you deliver digital services—it isn’t a datacenter infrastructure management (DCIM) tool. These types of CIs will just clutter up your CMDB, requiring additional time and effort to manage without providing any significant value. Keep in mind that once a CI has been created it will always be a CI.

Service mapping

Now, let's talk briefly about service maps. As previously mentioned, service CIs represent digital services and are supported by infrastructure CIs. Understanding the relationship between service CIs and supporting infrastructure CIs is critically important. For instance, it helps you to diagnose the root cause of service issues. A service map in the CMDB captures these relationships, showing which CIs support the service and how they are related to other CIs.

Of course, the CMDB already captures relationships between infrastructure CIs, so how are service maps different?

To explain, a service map is a collection of relationships and CIs that represents how a specific application service is delivered. It's a subset of the CIs and relationships in the CMDB.

Let's revisit the "web server X depends on load balancer Y" example. In fact, many different web servers might depend on the load balancer Y—web servers A, B, C, D, and X, for instance. However, only web server X is involved in delivering a specific service. While dependencies may exist with the other four web servers, they are used by completely different services.

In this case, the service map only includes the X to Y dependency relationship. In general, a service map only includes the CIs and relationships that are involved in the specific service, showing how the service flows across your operational infrastructure.

You have probably come across the terms "horizontal" and "vertical" discovery. Horizontal refers to populating all of the point-to-point relationships between CIs—for example, discovering all web servers and load balancers and their dependencies. This gives you a view of how your infrastructure is configured, as previously described in "Configuration items, attributes, and relationships" above. Vertical refers to creating a service map by taking a vertical cut of CIs and relationships that represent the end-to-end service.

The CMDB, Service Graph, and the Common Service Data Model

As a member of the ServiceNow community, you may have heard about Service Graph and the Common Service Data Model (CSDM). If so, you're probably wondering what this means for your CMDB.

Service Graph is ServiceNow's next-generation system of record. Historically, the CMDB has focused on managing infrastructure, assets, and their relationships. However, ServiceNow customers now want a consistent, data-driven approach to managing the entire digital lifecycle, including planning, application development, deployment, performance, cost, business processes, and other areas. Service Graph provides this by implementing the CSDM, a comprehensive data model that covers the complete digital lifecycle. You can think of the CSDM as Service Graph's data schema, in the same way that the CMDB has a data schema.

Here's the good news. As we said, Service Graph is an evolution of the CMDB. Both Service Graph and the CSDM are backwards compatible with the CMDB, so none of your CMDB investment will be lost. In fact, as Service Graph continues to evolve, it will make your CMDB easier to manage and give you powerful new capabilities.

Service Graph and the CSDM also provide enhanced data governance—prescriptive guidance on how to populate data in Service Graph. Even if you're not planning to use the full capabilities of Service Graph today, we strongly recommend that you follow this same guidance when implementing your CMDB. This will reduce risk and improve the quality of your CMDB, and it will also position you to take full advantage of Service Graph in future. The information provided in this document is aligned with this guidance.

What about DevOps?

While DevOps is driving fundamental changes in the way IT organizations deliver digital services, DevOps teams still need visibility of their operational environment. In fact, they face an even bigger challenge than traditional development teams. They often have to deliver multiple software releases every day while minimizing service issues and still ensuring appropriate governance. And, as the name implies, they need to bridge the gap between development and operations, creating a unified whole. The need for a dynamic, accurate CMDB is greater than ever.

This document is not intended to address these issues—it's designed for IT organizations starting out on their CMDB journey. However, ServiceNow does provide comprehensive solutions for DevOps that seamlessly extend ServiceNow's CMDB capabilities. With these solutions, DevOps teams can ensure effective governance without compromising agility and speed, create real-time visibility of microservice architectures, identify the root cause of service failures, manage CI/CD pipeline quality and performance—including organizational performance—and more.

If you would like further information about ServiceNow's DevOps capabilities, please talk to your ServiceNow account representative.

CMDB business and organizational principles

Now that we've recapped the basics—and hopefully given you some insights into other areas such as Service Graph—let's talk about the business and organizational principles that will maximize your opportunity for CMDB success.

Identify areas where your CMDB can deliver business value

As we've already said, if your CMDB is just a master data repository, it has limited value. A CMDB delivers business value, so it needs to support both the strategic objectives of your business as a whole and the tactical goals of your IT organization. This means focusing on the things that matter to your business.

Many organizations struggle with the right amount of configuration management. Often, they try to take an all-encompassing approach. Inevitably, this turns out to be too expensive and time-consuming—no one is willing to spend millions and wait years for a result. Too little configuration management is just as bad. Without sufficient information, you're unable to properly support your business needs and deliver meaningful improvements. Your CMDB will end up being regarded as an expensive and worthless white elephant.

So, using a Goldilocks metaphor, how do you avoid the hot and cold porridge and go straight for the one that's just right? Don't assume you know what's just right. Engage with business stakeholders to identify areas where your CMDB can make a significant difference. In general, look for initiatives that will improve productivity, reduce costs, or increase employee satisfaction—these types of initiatives usually have the biggest business traction.

For example, are you being hit with big penalties for software license non-compliance? Or are you having too many service outages? If so, which services are affected and what's the impact on productivity? Are your employees fed up because their issues are bouncing around between IT support teams? Perhaps you're starting a cloud migration and need to know which services to move first—and how to move them. This isn't a comprehensive list; you'll find your own opportunities and challenges that you can address with your CMDB.

Again, remember you're not trying to boil the ocean. Don't spend a huge amount of time identifying every possible use case. Come up with a top 'N' list and prioritize it. Things to consider when prioritizing include:

- **Quantified benefits**—for example, how much money will you save?
- **Amount of effort**—what's the return on investment and do you have the resources?
- **Timeline**—will this deliver a quick win for your business?
- **Risk**—how complex is it, what could go wrong, and what are the unknowns?
- **Support from the business**—is someone asking for this and are they willing to go to bat for it?

If you're looking for an executive sponsor for your CMDB initiative, you may also want to prioritize use cases that deliver specific benefits for your target sponsor.

One last word on this—you may be tempted to draw up a comprehensive configuration management plan (more on that in a minute) before you engage the business. The choice is yours, but consider this. Until you have business alignment, that next level of detail is likely to change and be wasted. You need to give your business enough information to make decisions—and you have to have confiden in that information—but you don't need to have everything planned up front.

Think about data and processes

As we said at the start, the role of a CMDB is to provide accurate and reliable data about digital services and the infrastructure that supports them. In some cases, this data alone delivers value—for example, providing an inventory of deployed software licenses. However, you create much more value when you use this data to help drive and automate business processes. Sticking with the software asset management example, it's interesting to know what software is deployed, but it's even more useful to be able to reconcile these licenses with your purchased license entitlements—or to automatically reclaim licenses so you can reuse them.

Obviously, the role of your configuration management team is to provide the data that fuels these processes. However, unless these processes exist and have an owner, you're wasting your time. Identify process owners and work with them to define a solution that combines data and processes—and use ServiceNow to automate these workflows. That's what ServiceNow excels at, and it's how you'll maximize the value of your configuration data. And, if you don't want to build the workflows yourself, work with the process owner to identify resources that can build them for you.

You'll also be asked to participate in strategic technology projects. It's important to devote resources to these projects early on so you can assess any changes that are needed to your configuration management capabilities—for example, additional CIs, new data sources, or enhanced reporting capabilities. This is referred to in configuration management parlance as "project tailoring". You'll need to feed these requirements back into your prioritized list of configuration management initiatives, making it a living roadmap rather than a static to-do list.

Organizing for success

Now that you have initial business alignment on what you're going to do, it's time to plan how you're going to do it. We're not talking about technical execution here (that comes next). Instead, let's look at the organizational and planning aspects of successfully launching your CMDB.

We'll start with governance. Some organizations formalize their configuration management governance structure by creating a Configuration Control Board (CCB), in much the same way that many IT organizations have a Change Advisory Board (CAB). Creating a CCB does promote success, but it's a significant undertaking that is, quite frankly, better suited to large organizations with mature and extensive configuration management processes. If you're just starting out on your journey, creating a CCB may be too big a hurdle—and many IT organizations never create a CCB at all.

So, what's the alternative to creating a CCB? How do you start out lean and still achieve success? There are two key elements:

- Clearly define roles and responsibilities, not just within your configuration management team, but across the business. Get alignment on this, and make sure that the business is going to support you. For example, it's no use having everyone nod when you say you need a business owner—get a name and a commitment.
- Create a Configuration Management Plan (CMP). Think of this as your program blueprint. This needs to be comprehensive, but it also needs to be flexible. There's no point locking yourself into a three-year program in minute detail—you're going to need to update and evolve your plan as business priorities change.

Let's look at each of these points in turn.

Defining roles and responsibilities

Think about how you want to distribute your configuration management processes. Do you want a central team to handle everything, or do you plan on having individual teams? For example, will your storage team look after their own configuration management? Or do you want a hybrid model, with distributed teams feeding into a central team?

Here are some things to consider before making this decision:

- If you have a centralized team, you're still going to have to engage many subject matter experts. This is particularly true if you are dealing with on-premises infrastructure and plan to populate your CMDB manually (as opposed to automating discovery with ITOM Visibility). This requires strong project buy-in from your stakeholders and heavy project management. Even if you do automatically discover your infrastructure, you'll still need to engage subject matter experts to validate that the deployed infrastructure matches the intended design. Otherwise, as we said before, all you have is a master data project with no business outcomes.
- Distributed teams do have advantages. With a number of small, distributed teams, each team can make quick decisions and take responsibility for resolving issues. And, because each team is also a CMDB client—they use the CMDB data—they are best positioned to prioritize what data goes into the CMDB. And, if you are using discovery, the burden on these teams can be quite light since the CMDB is updated automatically.
- Many ServiceNow customers opt for a hybrid model. Even with discovery, distributed teams often don't have all the expertise needed to maintain their slice of the CMDB. For example, functions such as security span your entire IT infrastructure—and it's unrealistic to expect each team to have its own security expert. The same thing goes for monitoring, software asset management, and many other functions that depend on the CMDB. A hybrid model that combines centralized core competencies with the domain expertise of distributed teams often provides the best balance, and it also ensures process consistency and uniform governance.

Creating and maintaining a Configuration Management Plan

As with any major initiative, you need a clearly defined, structured and documented plan to succeed. In the case of configuration management, this is referred to—unsurprisingly—as a Configuration Management Plan (CMP). This needs to be a top priority *once you have initial business agreement on your goals and objectives* and have built *a* prioritized list of configuration management initiatives.

Here are some examples of things that your CMP should include:

- **Purpose:** Why have you created the plan and how do you intend it to be used?
- **Goals and objectives:** What are you trying to accomplish as per your agreement with the business? Provide a list of well-defined goals and associated success criteria.
- **Scope:** What does the plan cover? For example, is it just for production environments, or do you also intend to provide configuration management for development and test?
- **Roadmap:** What will you deliver, and when?
- **Applicable policies, standards, and processes:** Which policies, standards and processes will you follow as you execute your configuration management plan?
- **Roles and responsibilities:** Document roles and responsibilities you have defined. This needs to cover your configuration management team, as well as any roles you interact with—for example, application owners or business owners. A RACI matrix is a good way to do this.

- **Governance:** What are your governance processes? As already mentioned, you don't necessarily need to have a Configuration Control Board, but you do need some mechanism for making decisions and ensuring that things stay on track.

This isn't a comprehensive list. You'll want to add other things such as communications and training strategies, verification, and audit processes, how you respond to data integrity issues, and so on. Your plan should be sufficiently complete that someone else can pick it up and understand how you do configuration management—both your current state and your planned future state. However, don't overcomplicate your plan unnecessarily. If you build in too much detail, your plan will become inflexible, and you'll end up making major revisions every time there's a new business requirement. Remember, your top priority is to deliver business value, not to spend your time buried in documentation.

One last point on your CMP. This is a perfect place to clearly define your terminology. This is extremely important since you need an agreed common language to talk about configuration management. Otherwise, you'll end up having misunderstandings, which will inevitably lead to misaligned expectations and poor execution.

If you're looking for more inspiration for your CMP, you'll find a sample CMP outline in Appendix A. Obviously, you'll need to tailor this to your own specific needs, but it's a good checklist of the things you need to consider.

CMDB design principles

Now let's shift gears and talk about how to design your CMDB. How do you decide what information to store in your CMDB, and how do you structure that information?

Design with the end state in mind

Put simply, if you don't know where you're going, you're never going to get there. Create a clear picture of what you want to achieve and align your design with this target. Obviously, this includes the type of configuration data you're going to manage, but it's more than that. Ultimately, this data has to be useful and understandable. In fact, this is probably the single most important success criterion.

Think about who—or what—will use this data. For example:

- Does it need to flow into other ServiceNow processes? While these processes are designed to work out of the box with the CMDB, you still need to ensure the data meets your specific requirements. For instance, how fast do you need to discover infrastructure changes to support Event Management—do you have a relatively stable environment with infrequent changes, or is it highly dynamic?
- If people are using the data directly, how does it need to be visualized? For instance, all screens and reports need to be easily understandable and have a similar design. And each field must have a single defined purpose—don't change the meaning of fields depending on the context.

Align your data with your use cases

Of course, to design for the end state, you need to collect the right data. This may seem obvious, but your data needs to support your desired outcomes—the goals and objectives you agreed to with your business.

What exactly do we mean by aligning your data with desired outcomes? Here's an example. If your top priority is to reduce software costs, you'll want to collect information about your deployed software applications. It doesn't make sense to try to map your services in this case. What would you do with service maps? On the other hand, if you want to migrate services to the cloud, service maps are critical—they show you what components need to move, and they help you to avoid breaking services when you move them.

Make a list of the CIs and relationships that you need, along with the specific attributes that support your use case. Once you have this list, figure out where you can get the corresponding data. Again, this may seem obvious, but it will help you to focus your data gathering strategy. For instance:

- What information can you collect directly from your IT environment? You can discover this type of data using ServiceNow® ITOM Visibility. This includes infrastructure configuration, infrastructure relationships, and service maps.
- Which information do you need to bring in from third-party systems? For example, if you are working on a security use case, you might want to enrich CIs with information from vulnerability scanners. You can import this type of data using Integration Hub ETL, either by creating your own integration or using an existing integration from the ServiceNow® Store.
- What information isn't discoverable or available from third-party systems? You'll need to set up processes to collect this data manually and keep it up to date. One example of this type of data is organizational information. For instance, who owns a specific service?

At this point, also decide which CI attributes need to be put under change management. Obviously, discovered attributes and attributes imported from third-party systems don't need to be put under change management from a pure configuration management perspective—the data is what it is and no one can update it manually in the CMDB.

However, there may be separate operational change management processes associated with this data—for example, someone may need an approved change request to upgrade a software application. In these cases, you should be prepared to track changes to associated attributes—the application software version in this case—so that other teams can detect unplanned or unauthorized changes.

That leaves non-discoverable attributes: for example, the business owner of a particular digital service. Be selective when you decide which of these needs to be under change management from a configuration management perspective. Only choose those that are important since each attribute under change management creates additional work.

Out-of-the-box vs. custom CI classes

The ServiceNow CMDB comes with a rich set of out-of-the-box CI classes. Wherever possible, use these out-of-the-box classes as is—you'll save significant effort and simplify upgrades. Also keep in mind that these CI classes are designed to work seamlessly with other ServiceNow apps, so when you use out-of-the-box CIs, you have access to the full power of these apps.

However, you may come across cases where out-of-the-box CIs aren't sufficient —for instance, if you have a custom application or want to enrich CIs with data from third-party sources. In this case, you have two options: create a new CI class or subclass an existing out-of-the-box CI class. Which is the best approach?

In general, subclassing an existing CI class is the preferred approach. Why? Because you can still manage the new CI class at the superclass level. For example, let's assume you took the Linux Server CI class and created a new Embedded Server class to represent servers that are embedded in IoT devices. These servers might have additional attributes, such as shock tolerance or orientation (we're making this up). By defining the new Embedded Server CI class as a subclass of Linux Server, any app that can manage a Linux server can also manage this embedded server. It just won't know about the new attributes.

Of course, that still doesn't fully answer the question. For example, is it better to subclass the Linux Server class or its parent Server class? The answer is… it depends. If the embedded server behaves like a Linux server and you want to manage it like a Linux server, then subclassing the Linux Server class is the right approach. On the other hand, if it really isn't a Linux server but just shares a few unique attributes with a Linux server—say less than 30%—then subclassing the Server class is probably a better idea.

In general, choose an appropriate level in the class hierarchy when subclassing— one that provides appropriate superclass management while limiting the number of irrelevant attributes. If you take this approach, you should almost never have to create entirely new CI base classes—in extreme cases, you should still be able to subclass the existing out-of-the-box Application or Hardware base classes.

One note of caution. Make sure that you really need a subclass. For instance, we talked about subclassing if you have a custom application. You may need to do this in some cases, but if all you need to know is that this application is running on a server, the number of application instances, the location, and who is responsible for the application, then you can do this using the existing Application base class.

Implementation tip: You can easily subclass existing CI classes using the CMDB's CI Class Manager. Visit https://docs.servicenow.com/bundle/paris-servicenow-platform/page/product/configuration-management/reference/ci-class-manager-landing-page.html for more details.

What is the right level for your data?

We've just talked about managing your CI class hierarchy. What about the relationships between CI instances—and specifically the granularity of your data? For example, should you model a server as a single CI, or should you decompose it into its constituent parts—a top-level server CI with other CIs underneath it representing the power supply, hard drive, and so on?

In traditional configuration management practices (not just IT), deciding the granularity of CIs is referred to as "leveling". According to a well-established axiom, CIs should be leveled at the lowest level of independent change. Based on that principle, you would have a single high-level server CI if you don't want to manage change at the server component level. In fact, a significant number of ServiceNow customers have adopted this approach, rather than—for example—having a separate CI for a server's network interface card (NIC).

But is this the right approach? It really depends on whether or not you are using automated discovery.

Here's why. Creating and maintaining CIs is a laborious process if you do it manually. The effort of creating fine-grained CIs can be unsustainable in this case. And, if you don't need the extra CIs at the moment, there's a strong argument for just having a top- level server CI.

However, there is also a downside with this approach—and it has to do with data model governance. Let's come back to that NIC card example. Best practice says that IP addresses should be stored at the NIC level. Why? Because the NIC is actually configured with the IP address. If you had two NICs—for example, in a dual-homed host—then each NIC would have its own IP address. And of course, NICs aren't limited to servers. For example, routers have many network interfaces, even if they aren't separate cards. But, if you only have a top-level server CI, your only option is to store IP addresses against that server CI. This can lead to situations where IP addresses are sometimes stored against server CIs, and sometimes against network interfaces. This isn't hypothetical—it's a real issue that we've seen. And it makes it exceedingly difficult to manage IP addresses consistently in the CMDB since you don't know where to look.

This is only an example. In fact, the CSDM addresses this and many other similar issues by providing data governance guidelines. In the case of IP addresses, it mandates that IP addresses are stored against the network interface. And that's where ServiceNow applications expect to find IP addresses. If you store IP addresses against a top-level server CI, this may limit the ServiceNow functionality that's available to you.

So, what's the bottom line? If you use ITOM Visibility to discover your infrastructure and map your services, it will generate a fine-grained model that's aligned with the CSDM. There's no real downside to this—even though there are more CIs, you can still limit your processes to the top-level server CIs. For instance, there is no need to define processes to replace NICs if you don't want to. On the other hand, if you don't use ITOM Visibility, a single server CI may make more sense provided that you're willing to accept any consequent loss of functionality and the potential need to restructure your CMDB data in future.

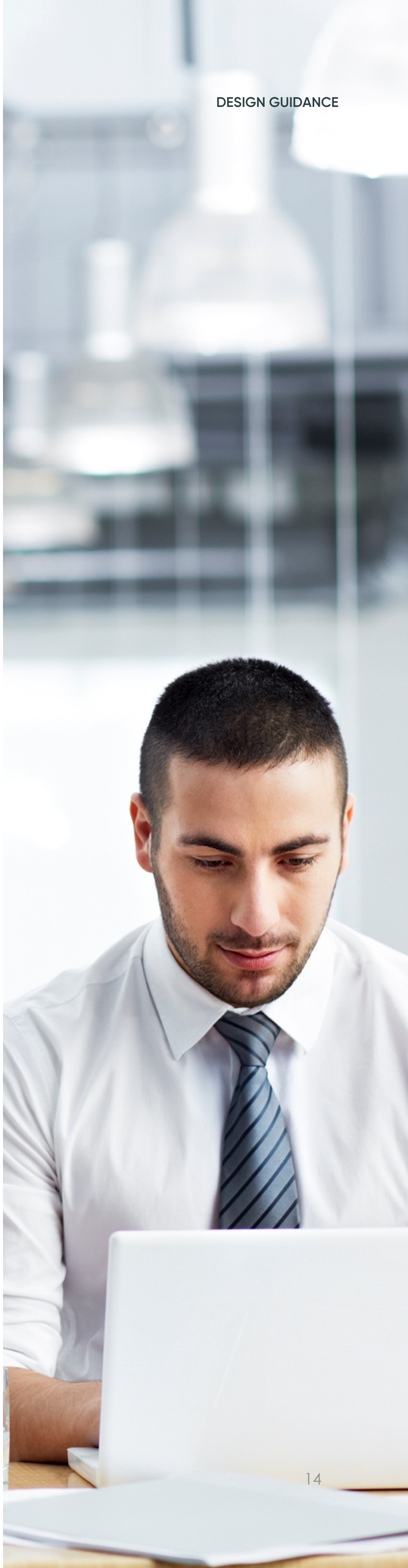
On a related note, we strongly recommend that you *do use* ITOM Visibility to automate infrastructure discovery and service mapping. Unless you have a small operational environment, manually maintaining thousands of CIs just doesn't scale. And, while mapping a service with ITOM Visibility requires upfront effort, once the service is mapped, ITOM Visibility automatically tracks subsequent changes without additional manual effort. Contrast that with manual service mapping, where service maps are often out of date as soon as they are created.

What about business services?

Up to this point, the guidance we've provided is related to infrastructure, technical services, and application services. All of these are directly linked to technology—whether we're discussing how to model network interfaces or build service maps, the technology dictates the options. Yes, we add organizational information to operationalize these types of CIs, but we're still talking about technology.

Business services are different. As we said before, a business service supports a business capability—such as managing customer orders—and is typically underpinned by one or more application services. However, there's no way to discover which application services make up a business service. It's a business view of capabilities, not a technical view of applications and infrastructure. In theory, a business service could have no underlying IT components at all—it could be provided by a shared services team that does everything manually.

Because they are not technical, business services need to be defined by the business, with IT playing a supporting role. However, you may find that no one in your business has defined business services—or even fully understands the concept. If this is the case, don't try to create business services on your own. Instead, focus on use cases that involve infrastructure, technical services, and application services. Work with your business to develop a long-term business services strategy, but don't make this a top priority. And make sure that you have high-level sponsorship when you do this. Otherwise, your efforts are likely to be wasted.



Operationalizing your CMDB

Now that we've covered key business, organizational, and design principles for your CMDB, let's look at how you turn these principles into practice. How do you populate and update data in your CMDB, ensure data quality, and manage the overall lifecycle of CIs?

Populating your CMDB

As we've already said, we strongly recommend populating your CMDB automatically whenever possible. Use ITOM Visibility to collect information about your infrastructure and digital services, and enrich this data with information from third-party systems where appropriate.

Start by establishing your discovery strategy. For example, do you want to discover all of your applications in one shot across multiple data centers, cloud providers, and end-user devices? Perhaps you want to focus on a specific data center, discovering all of the infrastructure and services in that data center before you move on to the next one. Or do you just want to map a few critical application services and their underlying infrastructure?

Again, the answer depends on what you are trying to accomplish—the configuration management use cases and priorities you have agreed with your business. Because of this, we can't tell you exactly what the right strategy is—you'll need to determine this based on your needs. However, there are several key things to consider as you develop your strategy:

- **It's relatively quick and easy to discover cloud infrastructure.** Major cloud vendors have well-structured APIs (for example, the AWS Config API) that let you discover cloud infrastructure and track changes in real time. ITOM Visibility is already integrated with these APIs, allowing you to populate your CMDB quickly and easily with cloud infrastructure data. Because of this, you may want to prioritize any use cases that involve cloud infrastructure.
- **Discovering on-premises infrastructure requires more time and effort.** Rather than having homogeneous APIs, on-premises discovery requires you to integrate with multiple IT components and systems. While ServiceNow® ITOM Visibility comes with a rich set of discovery integrations, this still introduces additional complexity. You will also need to ensure that ITOM Visibility can communicate with all of these components and systems. For example, you'll need to open firewall ports and provide device credentials. This may take a significant amount of time due to organizational barriers—while the physical effort may be minimal, there is often resistance from security teams and system owners due to security policy compliance.
- **ServiceNow customers typically begin with horizontal infrastructure discovery rather than vertical service mapping.** To map a service, you need to discover its infrastructure components first. Service mapping works by determining the service-level relationships between CIs. These CIs must already exist in the CMDB, and ITOM Visibility needs to be able to communicate with the corresponding IT components to determine their service-level context. In other words, the CIs should already be discovered. Note that this also has implications for geographic discovery strategies. For example, if you limit discovery to a specific data center, this can result in incomplete service maps if a service spans multiple data centers.
- **If you already tag your cloud resources, you can use these tags to create basic service maps quickly and easily.** Tags are key/value pairs that are used to label cloud resources. For example, a {service, x} tag could indicate that a specific cloud resource is part of service "x". Tags are supported by all major cloud vendors and are widely adopted. If you already have a cloud resource tagging policy, ITOM Visibility can use this policy to create basic service maps. These

maps show you each service's cloud resources, but they don't show you the relationships between these resources. While limited, this information can still be extremely helpful in a number of use cases—for example, understanding service costs.

As previously discussed, ITOM Visibility follows CSDM data governance guidelines when populating the CMDB. This helps you to avoid data quality issues. However, if you plan to import data from third-party systems using Integration Hub, there is no way to automatically ensure CSDM compliance. In this case, you are responsible for complying with CSDM data governance.

Note that ServiceNow has recently launched a Service Graph Connector Program to certify vendor integrations for CSDM compliance, timeliness, and throughput. We suggest that you use these certified connectors where available.

The Identification and Reconciliation Engine (IRE)

Data quality is critical for CMDB adoption. If your internal customers don't trust your CMDB data, they won't use it. And, once you lose that trust, it's often impossible to regain it. That's why it's crucial to design in data quality from day one. You don't get a second chance. And, leaving aside trust, the consequences of poor data quality can be catastrophic—prolonged service outages, misinformed business decisions, low productivity, wasted expenditures, and more. Just think about your use cases and the benefits you're anticipating. You're not going to achieve these benefits if your CMDB data quality is poor.

Data governance—as we've just discussed—is a crucial component of data quality, but it's only part of the picture. It's not enough for data to be consistent and timely—it also has to be identified correctly. For example, consider a hypothetical example—a "transmitter". One data source might refer a specific transmitter as "TX 01", whereas another data source uses "TMTR A" to identify the same transmitter. Unless these two conventions are reconciled, 'TX 01" and "TMTR A" will appear to be different transmitters, rather than different names for the same transmitter. In the context of the CMDB, this results in duplicate CIs: one for "TX 01" and one for "TMTR A". Again, this has serious consequences—for example, it complicates diagnosis of service issues and can result in duplicate work. And, of course, it makes it impossible to generate accurate reports.

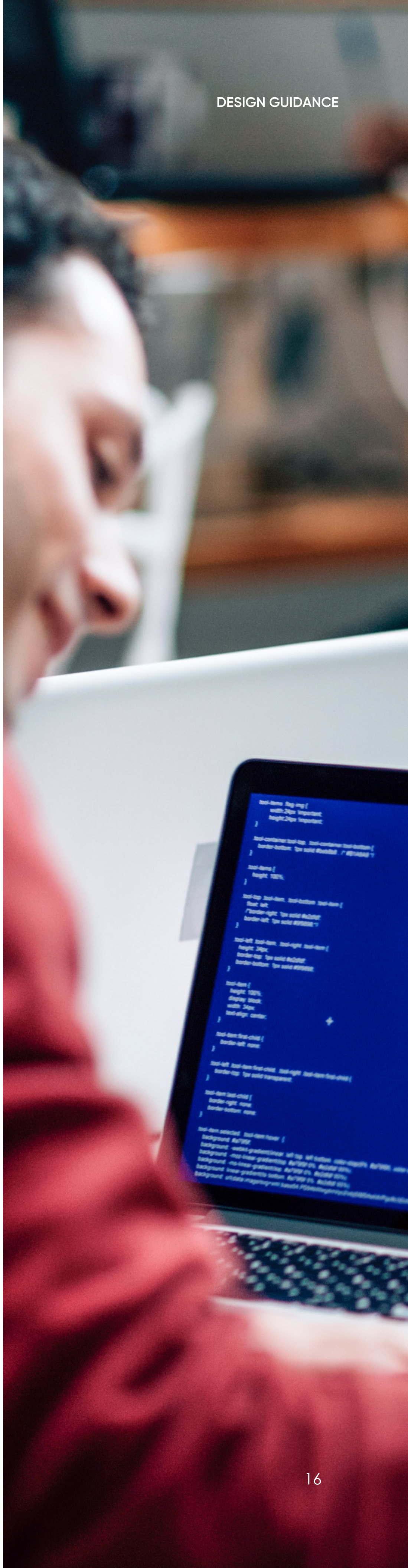
The ServiceNow® Identification and Reconciliation Engine (IRE) addresses this issue. It provides a centralized framework for CI identification and reconciliation across multiple data sources. It does this with configurable identification rules tied to specific data sources, which it uses to uniquely identify CIs and prevent duplicates. The IRE also reconciles CI attributes by only allowing authoritative sources to write specific attributes to the CMDB. This prevents attribute values from oscillating when two different data sources report conflicting values.

The IRE works out of the box with ITOM Visibility, and you can also configure it to handle third-party configuration data that you ingest via Integration Hub. You can find more information here about the IRE, including how to create identification and reconciliation rules.

Managing the CI lifecycle

The job of your configuration team doesn't end when you populate a CI into the CMDB. You need to discover attributes and relationship changes in a timely manner. You also need to ensure that attribute owners keep non-discoverable attributes up to date. And, if a non-discoverable attribute is under change management, you'll also need to ensure that any change is approved before it is made in the CMDB. You can automate this using ServiceNow® ITSM Change Management.

Even with these processes in place, there is always the possibility of configuration drift—for example, if someone forgets to update the CMDB or makes an error while updating it. That's why it's important to periodically validate non-discover-



able CIs attributes with CI owners. You'll also want to periodically confirm that discovered CIs and relationships match design intent.

Use ServiceNow Data Certification to do this. This provides on-demand and scheduled validation of CMDB data by automatically assigning certification tasks to appropriate data owners. These owners certify the data by answering a series of questions, with the results recorded in ServiceNow so you can review them and initiate remedial actions as required. You can also forward the results to ServiceNow® Governance, Risk, and Compliance to provide automated evidence for audits and other purposes.

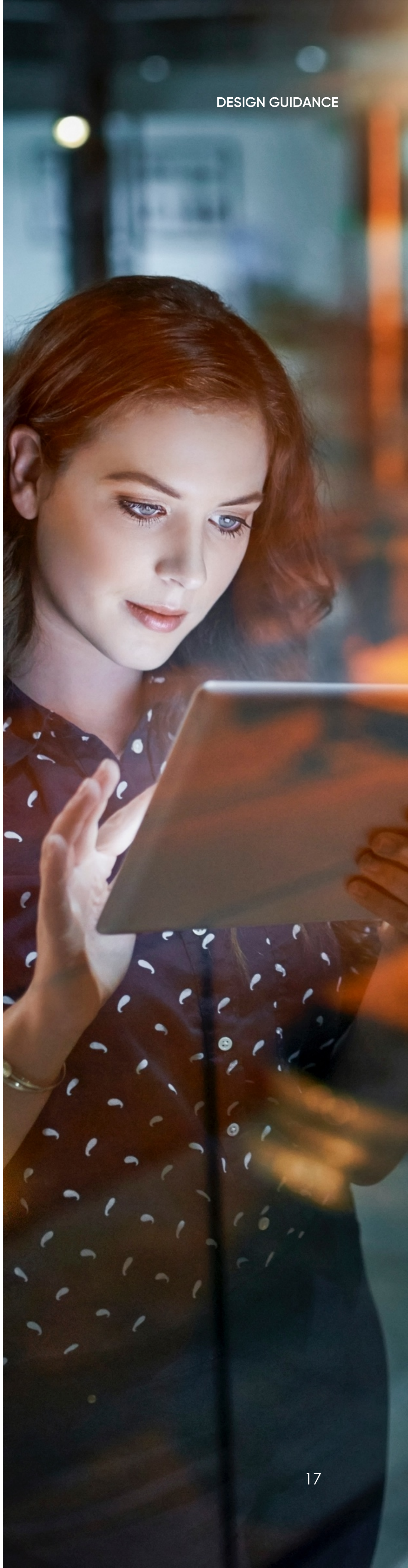
In addition to periodic validation, you'll also want to archive your CMDB data when you no longer need it for day-to-day operations. This releases system resources and declutters your configuration management environment. For example, you may decide to archive CIs when they're no longer part of your operational environment, or to archive historical data after a set period of time—for example, one year.

You can do this with ServiceNow's Data Archiving app. This allows you to create rules that control what is archived and when. It also gives you control over what you do with archived data—for instance, retain it for regulatory purposes for five years and then destroy it.

Monitoring the health of your CMDB

Once you have the tools and processes in place to maintain a healthy CMDB, you need to resolve issues as soon as they arise. The best way to do this is to monitor your CMDB using the ServiceNow® CMDB Health Dashboard. This shows you KPI scorecards for your CMDB, including CMDB completeness, correctness, and compliance. You can also drill into health details for specific services, groups of CIs, and individuals CIs. This allows you to pinpoint CMDB health problems and take corrective action—for example, by following up with CI owners or service owners when CIs become stale (i.e. haven't been updated recently).

You can also gain actionable insights from the CMDB and CSDM Data Foundations Dashboards, which can be downloaded from the ServiceNow Store.



Let’s recap

Your CMDB is the cornerstone of successful digital service delivery. It’s more than a master data repository. It enables data-driven business outcomes, strengthening your internal IT processes and helping you to realize major business benefits beyond IT.

Implementing and maintaining a successful CMDB isn’t an overwhelming task, but it does require a structured approach that starts with identifying business needs. Key steps along your CMDB journey include:

- Working in partnership with your business to identify areas where your CMDB can deliver value
- Thinking about how data and processes need to work together to deliver business outcomes
- Clearly defining your configuration management processes, including roles and responsibilities
- Documenting your configuration management plan and evolving it to keep pace with business needs
- Designing your CMDB with your desired end state in mind
- Aligning your CMDB data to support your use cases
- Following data governance and design best practices
- Populating data automatically wherever possible
- Taking proactive steps to avoid data quality issues and keep your CMDB up to date and accurate
- Monitoring the health of your CMDB and taking immediate remedial action when needed

We hope that this document has helped you to chart your course to a successful CMDB. If you have further questions or need additional help, learn more here.

servicenow

Appendix A—A sample Configuration Management Plan outline

1. **Introduction**
    a. Purpose
    b. Background and Context
    c. Scope
    d. Definitions
    e. Goals and Objectives
    f. Future State Roadmap
    g. References
    h. Applicable Policies and Standards
    i. Hierarchy of Configuration Management Plans

2. **Configuration Management Organization**
    a. Organizational Structure
    b. Current Assignments
    c. Roles and Responsibilities
    d. Configuration Management Job Family
    e. Configuration Control Board
    f. Interfaces to Other Governance Mechanisms

3. **Configuration Planning**
    a. Overview
    b. Project Tailoring
    c. Continuous Process Improvement
    d. Training Strategy
    e. Communications Strategy
    f. Policies
        i. Configuration Management Plans
        ii. Configuration Management Plan Reviews
        iii. Deviations and Waivers
        iv. Organization and Assignment Changes
        v. Communications
        vi. Service Requests
    g. Procedures

4. **Configuration Identification**
    a. Overview
    b. Policies
        i. Configuration Item Classification
        ii. Configuration Item Identification and Registration
        iii. Configuration Item Naming
        iv. Decommissioning and Archiving
        v. CI Relationship Modeling
        vi. Configuration Baselines
        vii. Definitive Media Library and Hardware Store
        viii. Relationship with Asset Management
        ix. Data Confidentiality
    c. Procedures

5. **Configuration Control/Change Management**
    a. Overview
    b. Policies
        i. Required integration with Change Management
        ii. Configuration Control Processes
        iii. Baseline Control
        iv. Interface Control

v. Version Control
vi. Supplier/Sub-Contractor Control
vii. Release Authority
viii. Relationship with Release and Deployment Management

a. Procedures

6. Verification and Audit

a. Overview
b. Policies
 i. Verification of Changes
 ii. Scheduled Audits
 iii. Data Integrity Incidents and Response
 iv. Audit Responses and Closure
c. Procedures

7. Status Accounting

a. Overview
b. Policies
 i. Standard Reports
 ii. Customized Reports
 iii. Dashboard Reporting
 iv. Integration with Other Processes
c. Procedures

8. Interface Control

a. Overview
b. Policies
 i. Life Cycle and Procedural Interfaces
 ii. Organizational Interfaces
 iii. Hardware Interfaces
 iv. Software Interfaces
c. Procedures

9. Supplier/Sub-Contractor Management

a. Overview
b. Policies
 i. Supplier/Sub-Contractor Acceptance of Work
 ii. Supplier/Sub-Contractor Issue Management
 iii. Supplier/Sub-Contractor Periodic Reviews
 iv. Supplier/Sub-Contractor Audits
c. Procedures

10. Configuration Management System (CMS) and Tools

a. Requirements for the CMS
b. Logical Design of the CMS
c. Physical Design of the CMS
d. Use Cases
e. Testing
f. Test Plans
g. Test Scripts
h. Current Operational Constraints
i. Maintaining the CMS

11. Document Information

a. Approvals
b. Revision Log
c. Distribution List