CHAPTER 3

Engineering Drupal

Drupal is a powerful framework for building enterprise solutions that range from simple web sites to complex web-enabled applications. While Drupal 8 off-the-shelf could be used to build any of the broad spectra of solutions, there are best practices for engineering enterprise class Drupal. The chapter covers the key principles for determining the best approach, and the details involved in successfully engineering a solution that is scalable and adaptable.

Engineering the Foundation

Constructing anything requires an understanding of the requirements of what you are about to build, whether it is a bridge, an automobile, a house, a pizza, or a Drupal site. Building anything without a thorough understanding of the requirements will likely result in having to rebuild some or all of the foundation. If you begin building an automobile and later find out that the true requirements include the ability to tow a travel trailer and haul eight adults then you may have to radically shift the architecture of the two-seater convertible that you just about completed, a task that would likely require starting over. When building enterprise class Drupal, the best approach involves a thorough understanding of the needs of the various constituents that the solution must support. Having spent the past 12 years and 33,000+ hours working as the architect of enterprise class Drupal solutions, it's key that you understand high-level goals and objectives such as:

- What type of sites will the organization create? Are they primarily delivering marketing information? Is online commerce a key consideration? Is there an online community component (user generated content)? Will the sites be multilingual? Are the sites functionally similar or are there wide variations in the types of sites that will be built? Understanding this aspect will help you determine whether all sites can be constructed from a common distribution or whether the variance in functionality will require multiple base platforms on which to build and launch Drupal sites.
- How many different web sites (domains) will the organization construct over the next one to three years? Understanding the number of different sites will help determine whether to build each site independent of the others, whether to use a solution such as Drupal's multisite architecture, or whether a custom enterprise distribution from which each site inherits a majority of its structure and functionality is in order.

T. Tomlinson, *Enterprise Drupal 8 Development*, DOI 10.1007/978-1-4842-0253-1_3

- Is there an existing Drupal distribution that closely matches the functional and technical requirements for the organization’s sites? For example, does Drupal Commerce, Open Scholar, Open Publish, Open Government, Open Atrium, or other distribution closely match your organization’s requirements? Does the organization already have Drupal sites in place? If so, is one or more of those sites a candidate for building an enterprise class Drupal distribution for the organization?
- Will Drupal integrate with other non-Drupal systems in the enterprise? If so, what role does Drupal play? Is it a provider of information to external applications and web sites? Is it a consumer of content from other applications and sites? Is it both? If Drupal is primarily a provider of information to other enterprise application, having a robust user interface may be lower priority than having a well architectures services layer for providing REST APIs.
- What user interface best serves the consumers of information contained in the organization’s sites? Does the Drupal interface suffice? Does AngularJS or another decoupled user interface provide a better interface? Headless or decoupled Drupal is becoming a more popular option for organizations that want to be more creative in the presentation of content to their users than what is typically accomplished through the traditional Drupal frontend. A prime example is `weather.com`, which uses a decoupled approach with AngularJS as the presentation layer and Drupal as the decoupled provider of content.

Answering these questions will not provide the detailed level of specifications required to fully define the approach required to build an Enterprise Drupal architecture; however, it will provide the overall guidance as to how the individual pillars of the architecture need to be engineered to address the organization’s needs.

Defining the Components of Enterprise Drupal

With a general understanding of the fundamental requirements for your enterprise class Drupal 8 site, the next step is to begin the process of examining each component that will form the architecture and define how your organization’s requirements will impact each of the components (see Figure 3-1).

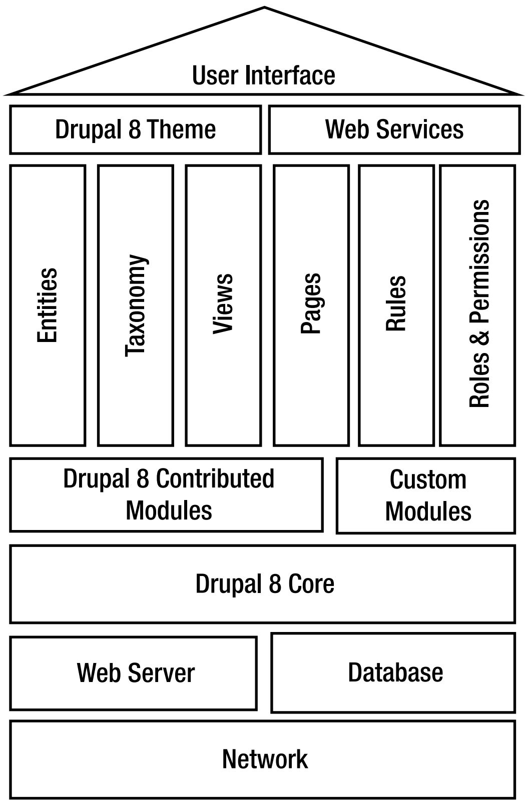

Figure 3-1. *Components of Enterprise Drupal 8*

Network and Web Server

The network and web server architecture required to support Enterprise Drupal 8 plays a significant role in the performance of your site, and there are several aspects that you should consider while engineering your solution. Most Drupal sites use Apache HTTP servers as their web server and Apache does well in that role. However, as your sites' traffic volumes grow, the load placed on the web servers often tax Apache's ability to serve pages quickly enough to ensure acceptable page load times.

Apache often faces what is called the *C10K problem,* which means that Apache has a difficult time supporting more than 10,000 concurrent connections, and in fact in most cases Apache falls far short of delivering adequate performance well before the 10,000 connections limit is reached. Apache's approach is to allocate memory to every additional connection, resulting in swapping to disk as concurrent connections increase. As the number of connections climb the performance quickly spirals downward, leading to unhappy site visitors and headaches for the operations team of your site. Nginx takes a slightly different approach, whereas Apache's approach is to fork a new process for each new inbound connection, where each new fork is allocated resources to process the connection, Nginx queues requests and processes them without allocating resources to each request. The result is lower overhead and faster responses to requests.

Drupal itself also consumes memory and CPU for each request that it receives, similar to Apache, but performance is often negatively impacted at significantly fewer than 10,000 connections. To resolve Drupal's own resource bottlenecks, the best practice is to employ reverse proxy servers. Reverse proxy servers receive request from browsers and then examine each request and determine what to do with it. They either carry out the request itself or send it on to the web server and Drupal for fulfillment of the request. Reverse proxy servers also provide the ability to cache static files (images, CSS files, and JavaScript files) separate from dynamic pages. A reverse proxy server may also cache PHP generated web pages, such as those pages generated by Drupal. By serving up pages from cache, Drupal never sees that request, as the request is fulfilled by the reverse proxy server. Using multiple reverse proxy servers also provides the ability to balance the load across several servers, further reducing the amount of time required to respond to requests.

Many of the biggest Drupal hosting providers, such as Pantheon, use Nginx and reverse proxy servers to ensure that the sites they host perform as desired. Your organization my choose to implement this same architecture in house, or you may rely on hosting providers to provide the infrastructure required to support your anticipated traffic volume.

Database Servers

The web and reverse proxy servers are the first line of defense in solving Enterprise Drupal 8 performance and scalability issues, while the database is a close second as the next area to focus on when engineering Drupal.

As an enterprise class platform, Drupal 8 requires the same level of capabilities and power as any other enterprise application, such as your enterprise resource planning (ERP), customer relationship management (CRM), human resources (HR), or other enterprise class applications.

Selecting the Database Platform

The de facto standard for most Drupal implementations has been MySQL. It was the first database supported by Drupal and continues to be the most popular option for most organizations. Drupal is optimized for MySQL, and while Drupal also supports PostgreSQL and SQLite, not all contributed modules support non-MySQL databases. There are also options for using Oracle and Microsoft's SQL Server databases, although using either of those databases is not considered the mainstream approach for Drupal.

While MySQL meets the performance requirements of most Drupal implementations, there are two MySQL "clones" that provide even higher performance and scalability options as they have replaced key component of the database engine focusing on performance. MariaDB is a fork of MySQL created and maintained by a team of MySQL engineers who left the organization when Oracle purchased the rights to MySQL. Percona is similar to MariaDB, but instead of a fork of MySQL, it is a branch of the main MySQL master branch. The primary difference between the two is that MariaDB diverged from MySQL at a point in time and continues down its own path, whereas Percona shadows MySQL and will continue to be tightly in alignment with the MySQL master branch. Many of the large-scale hosting providers use MariaDB as the database engine for their service offerings.

While MariaDB and Percona typically outperform MySQL, any of the three options are viable candidates to support an Enterprise Drupal implementation.

Clustering MySQL to Improve Performance

Traditionally Drupal sites often ran on a single instance of MySQL, and for many sites, that architecture supported them well until they hit a threshold of page views where the database became a bottleneck. After exhausting the options to tune MySQL to support the transaction volumes, the only alternative is to deploy more than instance of a MySQL server and employ clustering to distribute the workload across servers. This approach provides virtually unlimited database server resources and resolves the issue of the database as the bottleneck. While you may address some of the performance issues through reverse proxy servers and advanced caching mechanisms, it is wise to consider engineering your Enterprise Drupal architecture as a MySQL cluster to avoid having to retrofit your architecture at a later point.

For more information about MySQL clustering, visit `mysql.com/products/cluster`. As a point of reference, a standalone MySQL server may be tuned to deliver 250,000 to 500,000 queries per second, whereas a MySQL cluster, configured properly with the right number of servers and resources, can deliver 200 million queries per second.

Drupal 8 Core

There are several aspects of Drupal 8 core that you should carefully examine and consider while engineering your Enterprise Drupal 8 platform, and many of those options are discussed throughout this book. However, when launching your Enterprise Drupal 8 initiative, there is one aspect that will dictate how you engineer and build your Drupal sites. That aspect is how you want to build sites across your organization. There are three general alternatives:

- Single site
- Multisite
- Distribution

Single Site

A single-site architectural approach focuses on building each site or application by starting with Drupal core and adding the contributed and custom modules required to address the functional and technical requirements for that specific site or application. This approach works well and has been the de facto standard for many organizations. A single site solution framework works best for organizations in which every site and application is significantly different and there is little opportunity to leverage a common framework across all sites and applications. In this case, a common framework would likely be limited to Drupal core and a small number of contributed modules. While there may still be value in developing a common platform, the benefits are not as significant as the other architectural approaches. While it may seem as the easiest alternative to building sites in your organization, you will likely come to the realization that having to maintain dozens or even hundreds of independent sites is overwhelmingly complex and costly. Fortunately there are better ways, as described in the next two sections.

Multisite

Drupal multisite is an approach that has been around for nearly 10 years and is employed as the primary structure for hosting sites on Acquia. A multisite architecture consists of a single codebase with each site or application having its own database and configuration.

The benefit of this approach is that you only have to maintain a single instance of Drupal and contributed modules. An update to Drupal core, contributed, or custom module applies to all sites hosted in a multisite-based architecture. The benefits of a single codebase is often the primary benefit of a multisite architecture; however, there are potential pitfalls, such as:

- A single erroneous update to a module can take all of your sites down, as all sites share the same codebase.
- Scalability may be an issue, as all sites are running on a single instance. A distribution-based approach, on the other hand, provides the ability to spin up independent containers as increased demands warrant additional resources.
- Administrative access to a multisite architecture is difficult to restrict to single sites for tasks like updating a custom module.

Multisite is widely used in large organizations and is a viable approach, but there are tradeoffs that may be addressed by using a distribution-based model.

Distribution

Using a common distribution is the third approach and is based on the concept of assembling a "packaged" solution that addresses a majority of the functional and technical requirements for all sites and applications in an organization. This approach is nearly identical to using one of the community contributed distributions as the foundation for your site—for example, using Drupal Commerce Kickstart, Open Atrium, or Open Public as the upstream distribution on which you build all of your sites.

A distribution based approach starts with engineering a common Drupal footprint that addresses a majority of the functionality across the types of sites in your organization. You then create that site with the core building blocks to address that functionality, such as:

- Drupal 8 core
- Contributed modules
- Custom modules
- Entity types that address the common content requirements
- Taxonomy that addresses a consistent enterprise categorization of content using a common terminology
- Views that render content in ways that are consistent across the organization
- Page templates that address the common layouts used in the organization
- Common enterprise-wide navigational elements (menus)
- Common blocks
- An enterprise-wide search framework
- Integration with legacy enterprise applications and content

It is possible to assemble a common distribution that addresses a majority of the needs of the organization, fulfilling 80% of the common requirements. Creating a new site using a distribution is relatively straightforward—you clone the distribution from a centralized source code control system such as GitHub, install the distribution, and expand on the functionality provided by the distribution where necessary to address a site's specific requirements.

By setting the upstream master of the cloned site to the distribution's repository on, for example GitHub, you have the ability to pull updates and enhancements from the distribution into localized versions of the distribution, making the process of rolling out updates, patches, security updates, and expansion of functionality a relatively simple process. There are hosting providers, such as Pantheon, that provide this capability as part of their enterprise hosting packages, or you can build it yourself.

Profiles

If you select Drupal multisite or a distribution as the approach for building your Drupal 8 platform, you may consider creating one or more installation profiles. Installation profiles combine core Drupal, contributed modules, themes, and pre-defined configuration into one download. Installation profiles provide specific site features and functions for a specific purpose or type of site. They make it possible to quickly set up a complex, user-specific site in fewer steps than installing and configuring elements individually.

As an enterprise is it likely that there won't be a "one-size-fits-all" profile to address every type of site in your organization. For example, you may have a site that is primarily a marketing web site, while another site delivers technical product information to customers who purchase your products, and yet another site is primarily a commerce web site where you sell products and services. While it is possible to build three different distributions to address those three very divergent sites, it is more effective, efficient, and less complex to build a single distribution using installation profiles.

Don't underestimate the power of installation profiles, as they may save your development team countless hours of spinning up new Drupal 8 sites for the various constituents in your organization. See Appendix C for details on how to create a Drupal 8 installation profile.

Drupal 8 Contributed Modules

When engineering your Drupal 8 solution, it is likely that you will need to step outside the capabilities of Drupal 8 core to address the functional requirements of your organization. While Drupal 8 core is feature rich, it can't address every possible requirement from every conceivable use of Drupal 8 in organizations around the world. Combining Drupal 8 core with contributed and custom modules will provide the foundation for addressing your organization's specific needs.

Many organizations fall short when engineering their Drupal footprint by overlooking contributed modules that may solve their functional and technical requirements and, instead, developing custom modules that must then be maintained by their organization. The task of finding the right contributed module or combination of modules is often a tedious one, but the long-term payoff of using contributed modules instead of developing custom modules is significant, especially when considering the cost of upgrading your custom modules to the next major version of Drupal.

There are no easy shortcuts to finding the proper contributed modules to address your functional requirements, other than searching through `drupal.org` and finding other similar use cases and how people solved those issues. When evaluating contributed modules, there are a few things to keep in mind:

- How many sites report that they are using the module? If the number is small, for example less than 50, closely examine the functionality to determine why more people aren't using the module.
- Check the issue queue and read through the bugs that people are reporting. If they seem significant and there are a lot of them, you may want to consider a different path. The sheer number of issues may not always be a good indicator though, because many heavily used modules have issues that number in the hundreds. They key is to look for critical issues and determine how actively people are working on them.
- Check the date of the last update to the released (non-dev) version of the module. If the module hasn't had a release in several months and there are several outstanding bugs that have been reported and worked on, understand that you may have some additional work to do to apply the patches that developers have submitted to address critical functional and technical bugs.
- Look for known conflicts with other contributed modules in the issue queue. If you have the modules that are reported as conflicting you may want to look for an alternative solution, as implementing that module may break other functionality on your site.
- Look for hooks that provide you with the ability to augment the module. A *hook* is a function that allows you to directly interact with the module to modify some aspect of the module's functionality, such as adding or modifying content that is being processed by that module. A module that provides 80% of the required functionality but has hooks is better than a module that provides 90% of the functionality without the ability to modify the functionality through a hook.

- When presented with multiple options to solve a functional requirement, examine the two modules carefully. Not every module solves the problem the same way and there are likely differences that will sway you one way or another. Another key indicator is the number of contributors to the module. More contributors means more arms and legs to work through the issue queue and update the module. There are also well known developers in the community who are known for the quality of their modules. While everyone can contribute a module, sometimes it pays to stick with the veterans who have consistently delivered high-quality modules to the community.

If at the end of your evaluation you've come up empty handed, custom modules are the acceptable path. Follow Drupal's best practices, which can be found at `drupal.org/coding-standards`, and consider contributing your custom module to the community. It's highly likely that someone else in the world is facing the same functional requirement and could benefit from your solution. Conversely, you have the opportunity to collaborate with others in the community to augment your custom module to make it even better.

Custom Modules

Early in my career shift to Drupal I partnered with a friend who was out building Drupal web sites and doing so quite successfully. What impressed me about his web sites was that there was no custom code; the sites were robust and complex and relied solely on off-the-shelf Drupal. While possible, it is likely that you will need to venture off into the realm of developing custom modules to address the unique functional requirements in your organization. When engineering your Drupal footprint and considering custom modules, it is important to consider a few key points:

- Is there a way that this can be accomplished with off-the-shelf Drupal? For example, can you use a combination Drupal capabilities such as web services, views, webforms, and rules, which provide the foundation for solving many functional requirements that cannot be addressed with existing modules.
- Can the requirements be modified to match the capabilities of an off-the-shelf solution? In my 30,000+ hours of building enterprise class Drupal solutions, when asked, requirements often shift to fit an off-the-shelf capability without having to address the functionality through custom code.
- Can the functionality be accomplished through extensions to an existing contributed module? Instead of starting from a blank slate, consider contacting the module maintainer for an off-the-shelf module that addresses a majority of your requirements. It is likely that you are not the only organization that requires additional functionality.

After exhausting an evaluation of alternative approaches, custom coding may be warranted. In that scenario, it is wise to follow Drupal 8 best practices when developing your custom solution. Chapter 4 discusses developing custom Enterprise Drupal 8 modules.

The Pillars of a Drupal 8 Solution

The items described previously are often considered the foundation of a Drupal 8 solution, and while contributed and custom modules provide a significant portion of the functionality associated with a Drupal site, the pillars of the solution are:

- Entities
- Taxonomy

- Views
- Pages
- Rules
- User roles and permissions

Entities

Entities form the basis for content, taxonomy, and users on your Drupal 8 site and therefore are often considered the critical pillar of a Drupal site. When engineering your Drupal solution it is important to consider the following:

- What types of content will be authored, stored, managed, and displayed across the enterprise and what is the structure of each type of content?
- What taxonomy vocabularies will be used to categorize content across the enterprise and what is the structure of the terms contained in each vocabulary?
- What information do you want to collect about your users, the ones who physically log on to your Drupal sites?

Content Entities

Understanding the types of content required across an entire enterprise may seem like a daunting task, but it is a necessary activity when creating a Drupal 8 solution that will address your organization's needs and speed the delivery of sites across your organization. While the effort may seem overwhelming, the reality is that most organizations have a small set of entity types that address nearly every piece of content authored by the organization. The process for distilling all of the information published across the organization starts with gathering a representative sample of the content that is currently published and looking for patterns of how that content is organized and the key attributes of each common pattern. The more you distill the better the outcome will be for those who are responsible for building the platform and for those who use the capabilities provided to them through the editorial interface.

As a general rule of thumb, having fewer entity types means the effort to manage and maintain your site will be easier. When examining the results of the distillation process, you will likely find that a vast majority of the patterns fit into a simple structure of a title, a body, a featured image, the date content was published, an article type field driven by taxonomy, and the author who created the content. If you look at this simple pattern, you'll likely see that it can be applied to news articles, press releases, blog postings, product overviews, new product announcements, and a host of other types of content. In this case you might consider creating a simple multi-purpose article entity type that can be used across a wide variety of use cases.

Not all entity types may be distillable down to a single article entity type. There are specific cases where it makes sense to have individual entity types for content that does not fit the simplistic format of an article. For example, an event has other information such as start date, start time, location, duration, file attachments, and other fields that are typically not applicable to an article. Instead of adding complexity to the article entity type, a separate event entity type is warranted, but constrain the team's desire to construct several content entity types. Instead, look for flexible ways to deliver content through the fewest number of content entity types.

For details about creating entity types in Drupal 8, read the Apress *Beginning Drupal 8* book, which can be found at `apress.com/9781430265801`.

Taxonomy Entities

Taxonomy is the second pillar in the enterprise-Drupal framework. It is also an entity. I'll cover the usage of taxonomy in the following pages. It also warrants an examination of how taxonomy can play a larger role than just providing terms that an author can use to categorize content. Taxonomy, like content entities, supports the concept of additional fields. Additional fields enable you to utilize taxonomy for broader purposes, such as being the source of certain elements. You could, for example, have a common page banner on all the site pages that are associated with a specific taxonomy term. There could be other attributes associated with a term that may be useful across the site such as category descriptions, related terms, images, or other information.

This is one area where many people miss the opportunity to fully leverage the entity aspects of taxonomy and take a more complex approach to what might be simplistic by utilizing additional fields on a taxonomy vocabulary and using taxonomy for more than just a repository of terms to categorize content. Expand your thinking about how a taxonomy term may provide additional supportive information that would enhance and enrich the content delivered to your end users. Follow the same steps as when adding fields to an entity type when creating taxonomy vocabularies. For additional details on creating custom fields on taxonomy vocabularies, check out the Apress *Beginning Drupal 8* book.

User Entities

Users are another class of entities, and like content and taxonomy, they are also fieldable and expandable beyond the base user entity defined by Drupal 8 core. There may be additional fields that would collect additional information that would enable enhanced functionality on the site, such as filtering content based on some attribute of their profile. Carefully consider the additional elements that may be added to the user entity to facilitate delivering functionality on your Drupal 8 site.

Defining entities is a key step in the process of engineering your overall solution, so do not overlook or shorten the process and take full advantage of Drupal 8 entities.

Taxonomy

Taxonomy is the second pillar of the solution footprint and it plays a critical role that many overlook. Many overlook taxonomy because they do not understand it or they don't think about the power that it brings to Drupal. I have been asked by many organizations over the years to solve common problems that could have easily been addressed by fully utilizing taxonomy. Here are some examples:

- Why is it so hard to author content and have it show up where it is supposed to on my site?
- Why is it so hard to assemble lists of content based on common characteristics?
- Why can't I automatically create a list of related products or articles?
- Why can't I have faceted search and make it easier for my visitors to find the information they are looking for?
- Why do I have to manually create all of these pages on my site? I thought Drupal could magically assemble pages of content without me having to physically place every piece of content on a page?

The question that I ask those organizations that are crying out for help is "How are you using taxonomy on your sites?" Their answer is typically, "What is taxonomy?" Which is exactly why taxonomy is one of my favorite capabilities of Drupal as it solves so many problems that exist when you don't effectively use it. Using it requires studying the organization and its content, including how editors want to curate content, how end users want to find content, and how the organization thinks and talks about its content.

Understanding the taxonomy of the organization and engineering it into the very DNA of your site will result in a significant reduction in the effort needed to create your new sites and will enhance your end users' ability to find the information they are looking for.

So where do you start? The first step is to understand how your organization talks about content. When they talk about content do you hear things like, "When we publish a news article we are trying to target a specific market segment, which includes a break down by industry, application, and problem faced by that target customer". In that simple sentence there are four uses of taxonomy:

- They publish news articles, where "news" represents a type of article
- They target industry segments, which is another categorization of content
- They narrow the focus in that industry by application
- Within the application they narrow the target to a specific problem that their organization addresses through a product or service offering

Using taxonomy and defining specific vocabularies (article type, target industry, application, and problem area) and terms in each of those vocabularies, content authors are able to pinpoint their target. Site builders can use those terms to construct views that display that content, filtered by taxonomy terms, on specific pages on the site.

There are other uses of taxonomy such as providing the capability for content authors to specify which sections of the site a specific content item is to appear, and further refining the placement of that content to a specific section of that page. You may, for example, have a vocabulary for "site section" and one for "content placement," where terms for site section might include the following:

- Homepage
- About us
- Products
- Services
- News

The taxonomy terms for content placement might include these:

- Featured
- Latest
- Call to Action
- Hero
- Recommendations
- Related

When you combine site section (for example Homepage) with content placement (Featured), you can begin to build patterns of filters that may be used in views to enable content authors to automatically place content on the right page, in the right section of that page, by simply tagging the content with the right terms and extracting that content for placement on the page through a view.

Taxonomy also plays a key role in enterprise search, providing the ability to filter content based on taxonomy terms, or through the use of search facets. This provides the ability to drill down through content to get to a content item of interest.

The challenge of taxonomy at the enterprise level is gaining consensus across the organization as to what something is called. The classic example that I often use is the word *rain*. While it seems like a good term to use to classify the water that falls from clouds in the sky, there are others that argue that the correct word is precipitation. There are others who prefer specific terms as to the volume of velocity of rain, such as downpour, drizzle, or mist. And there are others who will add modifiers to the word rain such as driving rain, freezing rain, light rain, intermittent rain, and others. While there isn't a single right answer, the issue becomes how do we consistently convey the word rain to all of our internal and external site visitors so that they can find everything on our site related to the word rain without having to dig through all the variations of the word? The answer? Agree to a common term and stick with it. This concept is often easier said than done due to human nature and our stubbornness in holding on to what we believe is the right answer even if it isn't the best answer. As the engineer of this solution you will have to gather the input from across the organization, distill the list of taxonomy vocabularies and terms into a single enterprise-wide list, and gain buy-in across the organization.

For large organizations, it may warrant the creation of a new job role called *taxonomist*, who is responsible for assembling the enterprise taxonomy, maintaining it, and enforcing it. Your job as the architect or engineer is to leverage the taxonomy to its fullest potential to simplify the process of building, maintaining, and finding content on your site.

Views

Views are the workhorse of most Drupal sites. Without views it would be difficult to extract content from the Drupal database and display it on pages. While not impossible to do, the level of effort exceeds the benefit of not using views to generate that output.

Using views to extract content is a relatively simple process. View's administrator's user interface provides a series of configuration options that identify what content, taxonomy, or users you want to extract and display, as well as which elements of those entities are important to display, the order in which they should appear, and how many should appear, all without writing a single line of code. For details on creating views, check out the Apress *Beginning Drupal 8* book.

When engineering your Drupal 8 solution, it is important to consider the following:

- Whether to create a view per use case, or whether you will create a view per entity type with view displays in that view that address a specific use case. Creating a standalone view for each use case typically results in an unmanageable number of views, which makes it difficult to locate a specific view that is rendering content on a page. Using one view per entity type minimizes the difficulty in locating the right view to use for a specific use case, as they are all contained in a single view.
- Whether to render individual fields through a view display or to use a module like Display Suite (`drupal.org/project/ds`) to control how content is rendered externally from the view. In the latter case, you would use the Display Suite layout to render the content. The separation of the physical layout of the output externally from a view display has significant benefits, including simplifying the creation of a view. All you need to do is output content and render it using the display created through Display Suite. You can make changes to the layout through Display Suite and apply that updated layout automatically to every view that renders content using that display. The alternative is to add every field that must be displayed through the view display and, when changes occur, update every display with the revised approach.
- Use views as your default approach for extracting content from the Drupal database, even in custom modules. While an entity query makes it relatively easy to query the Drupal database and return content based on specific criteria, views make it even easier when extracting complex use cases. As an example, when rendering a list of content that pulls information from other entity types and taxonomy terms, views makes it relatively simple to assemble that output. Writing the view and using the `views_get_view_result($name, $display_id)` function provides a relatively simple way to leverage views in custom modules.

- When a standard view doesn't quite do what you need it to do, instead of writing database queries directly, consider using hooks to modify the view. By using hooks (`api.drupal.org/api/drupal/core!modules!views!views.api.php/8.2.x`), you have the ability to modify and alter nearly every aspect of a view.

Using views as the "glue" between the content stored in your database and the user experience that you are delivering to your targeted visitors will greatly speed and simplify the development process for your base Drupal 8 platform, as well as the sites that you construct.

Pages

Defining common page templates that fulfill a variety of purposes, rather than constructing each page on your site as a single template, speeds the creation of your sites as well as simplifies the process of building and maintaining pages. As you examine the site maps of the sites targeted for your new Drupal 8 platform, consider the following:

- How many different page layouts are contained in the overall design of the sites that will be constructed? Distilling the number of layouts to a small set of flexible layouts provides the ability to leverage a preexisting template for page construction. In my 30,000+ hours of Drupal experience, most organizations are willing to limit the number of page layouts to 10 or fewer, which includes very large sites for very large international corporations.
- When defining what appears on the page, look for patterns that may be fulfilled using entities, taxonomy, and views. If you did your due diligence and defined a set of entity types that leverage taxonomy for attributes such as site section and content placement, you may be able to use views with views arguments (taxonomy terms in the URL) to automatically generate hundreds or even thousands of pages. All without having to do anything other than create a single page template with views in each region on that page that smartly extract content based on that criteria. For example, a page with a URL of `example.com/products/%category/%application` could be used to render every product related page on your site, assuming that content has been tagged with a product category and an application. There is no need to manually construct each page individually.
- Leverage tools such as Panels and Page Manager (`drupal.org/project/panels`) to simplify the process of creating and maintaining pages on your site. While it is common practice to hand-craft Twig templates for pages (e.g., `page—node—1.html.twig`), you can eliminate nearly all the need to code HTML and Twig to render a page on your site.

As you engineer your solution, the key point about pages is to do your best to constrain the number of page layouts used on the site to a reasonable number (e.g., fewer than 10) in order to simplify the process of creating the base platform. This also makes it easier for those responsible for building sites on your Drupal 8 platform.

Rules

While you can do almost anything you want to do on your site by hand-crafting custom modules, one of the often overlooked Drupal modules is Rules. The Rules module provides site administrators with a powerful tool for creating automated workflows on a Drupal site without having to touch a single line of code. Rules are "reactive," which means an event happens under a certain condition, which then triggers an event. When an event happens, a rule may conditionally manipulate data or execute tasks such as sending an e-mail to someone. Rules may also be scheduled to execute at a future date, making them even more flexible and powerful. As you engineer your Drupal 8 solution, it is important to think about common workflow related tasks and to consider using Rules as an alternative to custom code.

Examples of where the Rules module may play a key role include:

- Sending an e-mail to a site administrator when someone has requested a new account
- Sending an e-mail to a content administrator when a site visitor has posted a comment that is waiting for review and approval
- Changing the value of a field on a node when a date has passed
- E-mailing a customer a copy of their order when the order status has changed
- Deactivating a user account when certain conditions have been met

Rules are a powerful tool when employed on a Drupal 8 site. Don't overlook Rules as a solution to common workflow related activities.

User Roles and Permissions

The last pillar is user roles and permissions. They are not the last pillar because they lack power or importance to the overall architecture of your Drupal 8 solution; they take the last position because they impact users' abilities to perform nearly every function on your site.

Roles provide the ability to categorize users by common activities that are performed by that group. For example content editors have the ability to author, publish, and manage content. The actions of authoring, publishing, and managing are examples of the permissions that may be assigned to a user role. When engineering your solution it is important to consider and document the requirements of user roles and permissions as you build the framework. Often, roles and permissions are hastily implemented at the very end of a project and the implications and ramifications are often greater than what were expected.

As you engineer your solution, consider which roles are required in your organization:

- Who should have the ability to install and configure elements on the site?
- Who should have the ability to create key components of the site, such as entities, taxonomy, menus, pages, views, blocks, and user accounts?
- Who should have the ability to author and publish content?
- Who should have the ability to view content?
- Who creates and manages user accounts?

As you implement each component of your solution, update the user roles and permissions to address the functionality that you are building.

Drupal 8 Theme

Engineering the visual look and feel of your Drupal site focuses on the structure of pages, the elements that appear on those pages, and the visual representation of each element on the page. The theme is ultimately responsible for how everything on your site is visually rendered, including how it is rendered on every device that an end user may use to view your site.

When engineering your theme, it is important to consider the following:

- What devices are end users going to utilize when browsing your site?
- What is the screen resolution of those devices?
- How much content can reasonably fit on each of those devices?
- Is all of the content applicable to all of the devices? Or is there a use case where some content is not displayed on smaller screens?
- How will end users interact with the content on your site?
- What styling will be applied to elements on the page?
- What administrative control do you need to provide to site administrators? Should they be able to adjust the color attributes such as colors or fonts?
- What versions of browsers is the site going to support? Does the theme need to support older versions of Internet Explorer?

Chapter 5 covers the details of constructing a Drupal 8 theme. It's important to understand the requirements for visualization before beginning the process of creating a theme.

Web Services

In today's interconnected world, it is unlikely that your Drupal sites will live on their own deserted island, disconnected from other sources of content and providing content only to users who visit the sites through a web browser. A more likely scenario is that your Drupal web sites will live in an interconnected world, pulling content from external sources such as corporate applications, other Drupal sites, or sources outside of your organization. It is also likely that your Drupal sites will provide content to corporate applications, other web sites, systems outside of your organization, and will have other interesting interactions such as sending content to a digital sign in the lobby of your corporate headquarters.

Through the web services capabilities that are inherent in Drupal 8 core, as well as various contributed modules, your site may easily participate in this distributed digital environment by:

- Serving content to requests through a standard web services interface such as REST
- Consuming content from external sources through web services interfaces provided by those systems and applications
- Serving as a headless content repository by serving content to applications built in frameworks such as AngularJS

When engineering your solution it is important to consider the flexibility and capabilities that are presented through web services. I cover web services in more detail in Chapter 8. At this juncture, carefully consider and document the interfaces that may augment the capabilities and content on your site.

User Interface

While the Drupal theme is often considered the user interface to a Drupal site, a popular trend is to utilize Drupal as a content repository and editorial tool but not necessarily as the provider of the user interface to that content. The terms "decoupled CMS" and "headless Drupal" have been bantered about the Drupal community for the past few years and are beginning to build momentum. Sites such as `weather.com`, NBC's Tonight Show with Jimmy Fallon (`nbc.com/the-tonight-show`), Radio France (`rfi.fr`), and others are early adopters of using technologies such as Node.js, Backbone.js, Angular.js, Symfony, and others as the presentation layer for content that resides in Drupal.

When thinking about and engineering your Drupal architecture, consider the benefits of a decoupled CMS model. It provides breakthrough user experiences that would be difficult to accomplish using the Drupal theme engine, and it helps site owners future-proof their builds by allowing them to refresh the design of their sites without having to rebuild the whole CMS. The CMS, in this case Drupal, does not have to radically shift in order to accommodate a whole new user experience.

There are three main components for decoupling Drupal from the user interface:

- *Decoupled frontend.* In this approach the presentation of content may be handled through various means such as interactive JavaScript frameworks (Angular.js), static page generators, mobile applications, and even another CMS. In this approach multiple user interfaces can peacefully coexist with Drupal without disrupting each other.
- *Content delivery via a web service API.* In this approach the content housed in Drupal is accessible through a web service API, typically a RESTful interface, in a format (JSON) that is friendly to most presentation layer tools.
- *CMS backend and database.* Drupal is used as the content authoring and management platform.

The benefits of this solution are significant enough to strongly consider this approach:

- Future proofs your web site implementation and lets you completely redesign your sites without having to rearchitect and implement a CMS.
- Allows frontend developers to design and develop user interfaces that are free from the constraints presented by the user interface capabilities of the backend.
- Speeds up the performance of your site by shifting display logic to the client side and simplifying the backend.
- Allows frontend developers to build truly interactive experiences through in-browser applications.

Summary

There is a lot to consider when engineering your Drupal 8 solution framework that will become the foundation of the platform you deliver to the enterprise. There are elements of the analysis that, if left unanswered, may cause a significant amount of rework if the requirements differ from the foundation. Consider the wisdom of those who have gone before you and plan your site before embarking on the journey of building the platform.

The next chapter begins the discussion of the foundation of Drupal modules, including how to create custom modules to address unique functional requirements that cannot be accomplished using off-the-shelf Drupal.