

Contentful Certified Professional Exam

study guide



Introduction



WHY BECOME CERTIFIED?

Contentful is the platform of choice for the next generation of content management. As a result, Contentful expertise is increasingly in-demand.

Whether you are in an organization that is implementing Contentful, you work for an agency that serves Contentful clients, or you are an independent consultant, becoming a Contentful Certified Professional can help your career by providing objective proof of your Contentful expertise.

WHO IS THE AUDIENCE?

The Contentful Certified Professional exam is intended for software developers, technical architects, technical managers, content architects, and others responsible for the technical design, development or implementation of Contentful projects. As a result, this Study Guide is written primarily for this audience.

HOW SHOULD YOU PREPARE FOR THE EXAM?

This Study Guide provides a high-level overview of essential Contentful topics, features and best practices. It also contains many links to other learning resources (videos, documentation, articles). To get the full value out of the Study Guide, make sure you use these links to dive deeper into important topics.

While exam candidates who use the Study Guide tend to do better on the exam than those who don't, the study guide is neither a list of exam questions nor an answer key. There may be questions on the exam that aren't covered here.

In addition to the Study Guide, here are three other ways to supplement your learning:

1. Take the Developer and Content Modeling courses available in the [Contentful Learning Center](#).
2. Explore the [Contentful Developer Portal](#). Here you will find links to product documentation, forums, videos covering technical topics, and more.
3. Get hands-on experience with Contentful. If you don't already have access to Contentful, you can sign up for a [free account](#).

Table of Contents

Why Contentful?

A massive market shift.....	5
Stacks beat suites.....	7
Build products faster	8
Enterprises are replatforming on Contentful	9
Contentful's API-first approach	10
Power content across all your digital channels	11
Contentful runs as a SAAS	12
Contentful use cases	13

Content Delivery Architectures

Where and when does rendering happen?	14
Dynamic on server.....	15
Static site generator.....	16
Dynamic on device	17
Dynamic hybrid with Node.js	18

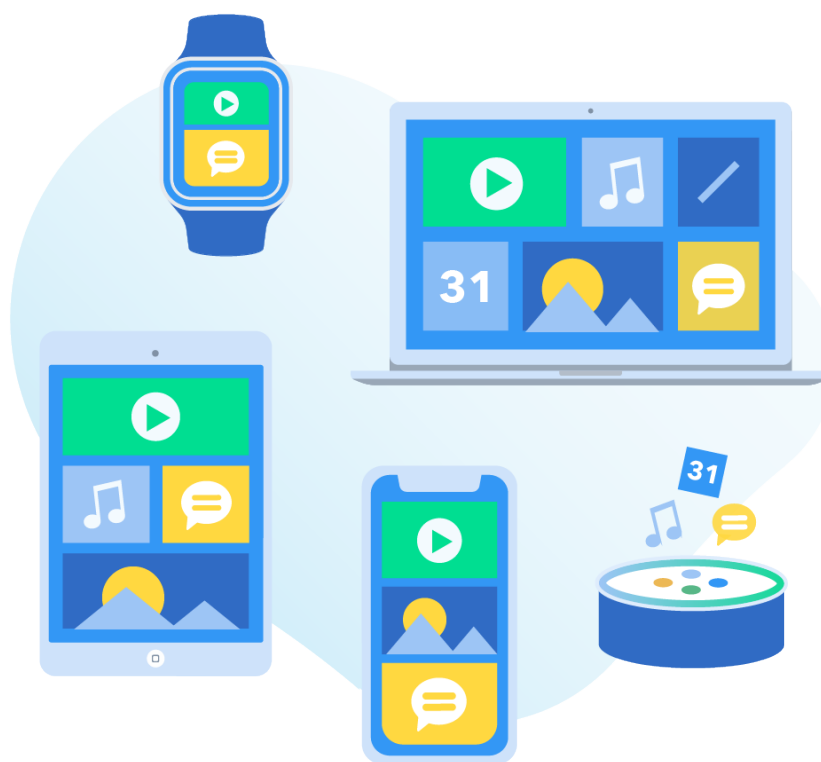
Contentful Core APIs

Four core rest APIs.....	19
Content Management API	20
Content Delivery API	20
Content Preview API	21
Contentful Images API	21
SDKs	22
GraphQL Content API	23

Content Modeling

Structured content.....	24
From structured content to content modeling	25
Content types and fields.....	26
What should you store in Contentful?.....	27
Topics vs assemblies	28
Fixed vs flexible assemblies	29
Inheritance	30
Composition	31
Localization	32
Field-level localization.....	33
Entry-level localization	34
Asset management.....	35
Media wrapper content types	36
Using a DAM.....	37
Rich text.....	38

Content modeling process.....	39
<u>Authoring in the Contentful Web App</u>	
What's in it for content authors?	40
The Contentful web app	41
The authoring experience	41
Nested entries	42
Previews for content authors	43
Localization for content authors.....	44
<u>Extensibility</u>	
Why extensibility?	45
Contentful's extensibility journey.....	45
What is an App?	46
What is the App Framework?	46
Apps vs UI extensions	47
Apps use cases.....	47
Apps locations.....	48
<u>Webhooks</u>	
What is a webhook?	49
Configuring webhooks	50
<u>Roles and Permissions</u>	
What are roles and permissions?.....	51
Configuring roles and permissions	52
Custom roles and permissions.....	53
Teams	53
<u>Managing Content at Scale</u>	
Why content as code?	54
CLI tools	55
Contentful's domain model	56
Space modeling patterns	57
One space to rule them all	58
Separate space per project	58
Multi-tier spaces.....	59
Using environments for agile development.....	60
Risk-free releases and instant rollbacks	61
Use cases for environment aliases.....	62
Managing content with code	63
Refactoring using expand/contract pattern	64



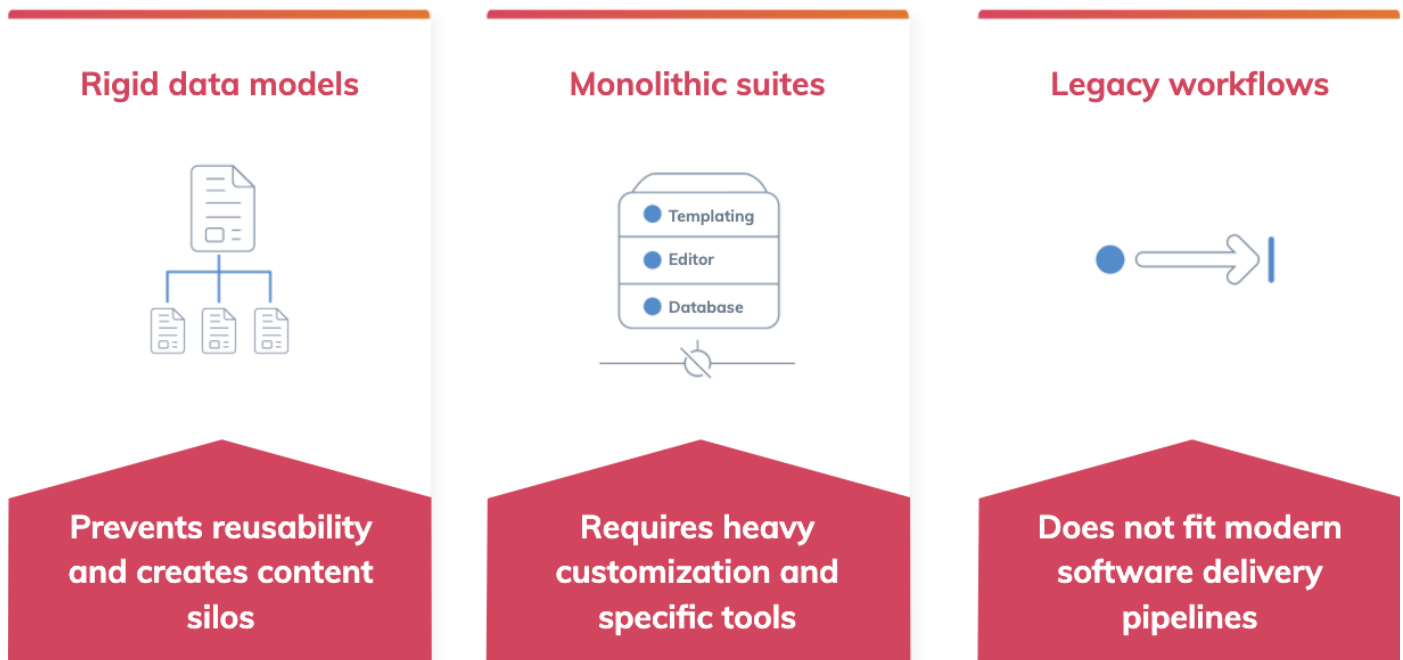
Why Contentful?



A MASSIVE MARKET SHIFT

Contentful was designed to be the content layer of the modern enterprise software stack. As opposed to proprietary and monolithic “suites,” Contentful is an API-first, cloud-native SaaS that was designed for easy integration into modern, microservice-based architectures. While many people refer to Contentful as a headless CMS (content management system), we believe “content infrastructure” is a more accurate term for what we provide.

One of the main drivers causing organizations to move to Contentful is that the nature of digital content is changing rapidly. To keep up, organizations of all kinds find themselves in the business of creating software. In the past, digital content mostly meant websites. But today, digital content can be displayed on a wide variety of devices—phones, watches, conversational interfaces, kiosks, digital billboards and many more. Contentful makes it possible to launch new digital sites quickly, reuse content between sites, and display content on any kind of digital device (websites, mobile apps, IoT devices, VR and AR experiences, etc).



CMSes slow down your team in a number of ways:

Rigid data models

- Inflexible content models don't work well across experiences, leading to silos and low reusability
- When you buy a CMS, you are buying a website and then customizing it to the use cases you need
- With content stuck in rigid data models and silos across multiple systems, it can't adapt to new channels and interfaces

Monolithic Suites

- All-in-one suites come tightly-coupled and highly-opinionated, forcing you to play by their rules
- Requires large upfront investments of

time and money; you can't just start building the MVP of your new digital product

- Comes bundled with many other products, causing customers to over-buy and under-use the solution

Legacy Workflows

- Not built for modern workflows and tools
- Hard to spin off dev environments, hard to do testing at scale, hard to merge content model changes back into production
- Hard to collaborate on complex digital products across multiple team

On-premises systems



Customization

MONOLITHS

Cloud-based stacks



Interoperability

MICROSERVICES

STACKS BEAT SUITES

The way that companies build software has totally changed. It's about stacks, not suites; microservices, not monoliths.

The big benefit of this modern stack is to enable digital teams to have greater agility and impact by focusing on their company's core business value and outsourcing the rest to an ecosystem of purpose-built services.

For publicly-traded or late-stage venture-funded vendors, investors become more important constituents than customers. Lately, investors have been looking for "greater share of wallet" and vendors have been falling over themselves to show that their cross-selling strategies are working.

There is no marketplace here because no enterprise digital leader would actually purchase "digital experience" as a platform. DX is a strategy and approach, and no single platform or vendor on this chart will get you there. We view Contentful as being the content layer in the modern enterprise digital experience stack.

BUILD PRODUCTS FASTER

Contentful accelerates your team in a number of ways:

Structured content

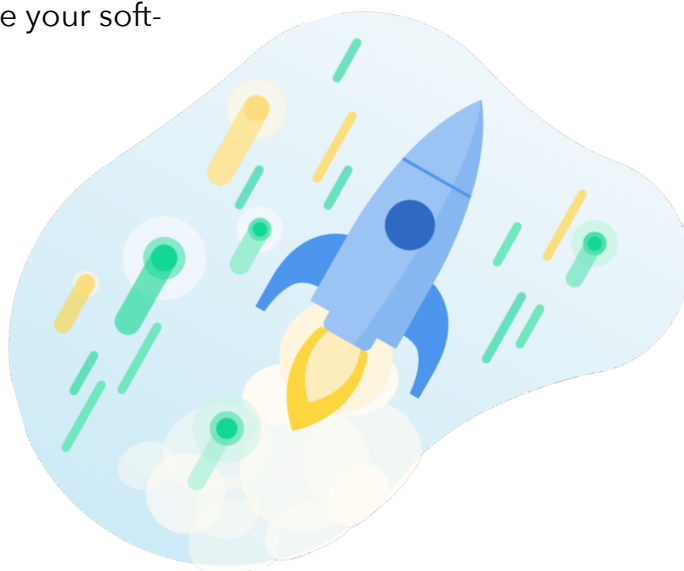
- You get a lot more flexibility by creating your content as components so it can be reused and repurposed across products, channels, and teams
- It's flexible enough for existing apps and future-proof for those yet to come
- It's context-agnostic, independent from mediums, platforms, programming languages, etc.
- It's also not tied to opinionated data models like a traditional CMS

Decoupled architecture

- The content is decoupled from the presentation layer so your team can build unique customer experiences
- By being born in the cloud, it was built to connect with other platforms and services
- It's composable, allowing you to use only what you need, and adopt new features as you grow
- It scales confidently on an enterprise-tested platform

Agile workflows

- It was born agile, purpose-built to integrate with software delivery pipelines, developer environments, and automation
- It changes with your software and workflows
- It lets you iterate and experiment fast, inside your software delivery pipeline





ENTERPRISES ARE REPLATFORMING ON CONTENTFUL

An example of Contentful in action is with Telus, a large wireless provider in Canada. With Contentful, Telus Digital greatly sped up the time it took to bring new products to market. When the Apple iPhone X launched, they were the only company in Canada to offer the new phone during the first 15 minutes after its release. It all started with a single proof of concept for the Samsung S8. In that proof of concept, their team reduced their go-to-market time frame by 14 times, from weeks to days.

By integrating Contentful with Adobe's personalization and analytics tools, Telus has seen a 14% increase in conversions. And with Contentful's infrastructure and CDNs for caching API calls they've seen a 30% increase in page speed.

1. Fast implementation. Fast performance.

- Cloud-native SaaS with CDN
- Use the development tools you know
- Fast, scalable content migration and transformation using code

1. Fast

2. Build any user experience for any device (anywhere).

- A large collection of SDKs
- Flexible content models
- Designed for easy localization

2. Flexible

3. Take advantage of the “modern stack” architecture.

- Combine best-of-breed, single purpose apps (micro-services)
- Built-in integrations with popular platforms
- Connect platforms easily using webhooks and UI extensions

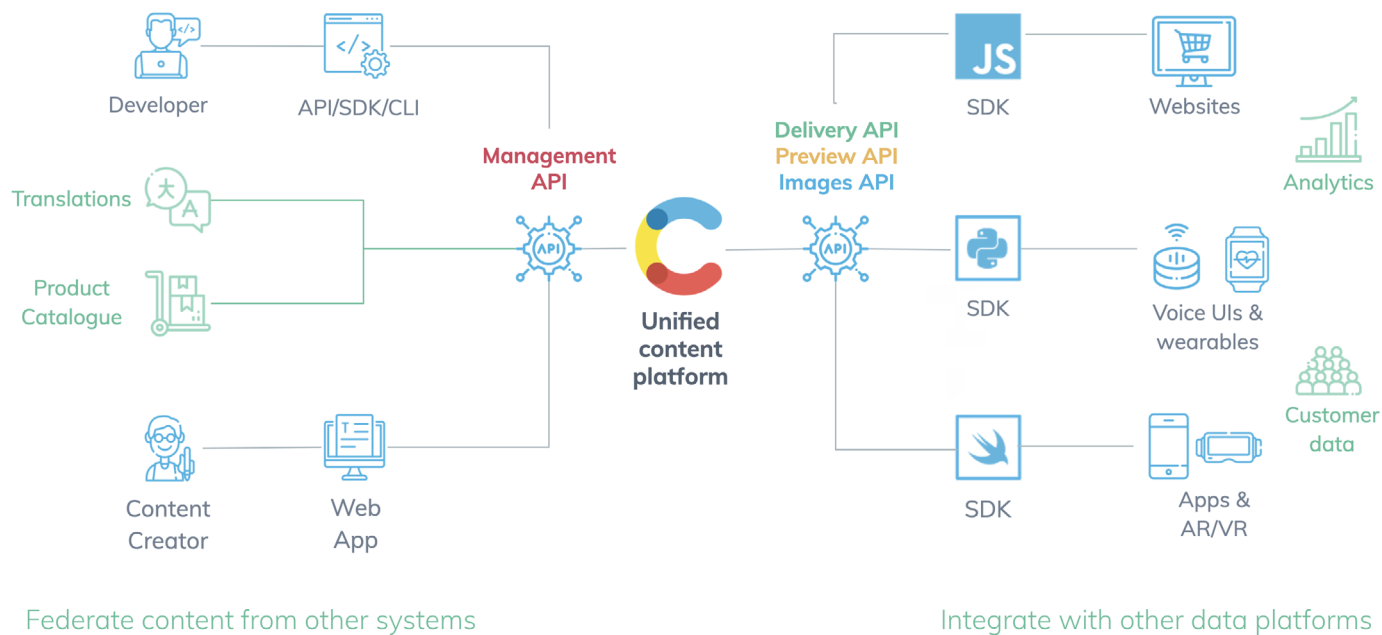
3. Modern Architecture

CONTENTFUL'S API-FIRST APPROACH

An API-first CMS is still a CMS, but one that deviates from a classic CMS (e.g. Wordpress) by not caring about the display of the content. Instead, an API-first CMS focuses on managing content and doing that well. The API-first CMS works by abstracting representations of your data, defining “types” of data that can be stored and instances of data that adhere to these types.

For you to use the content managed through the API-first CMS, you will get an API, typically a REST API that serves the managed content, including functionality to search, filter, sort and paginate the returned content.

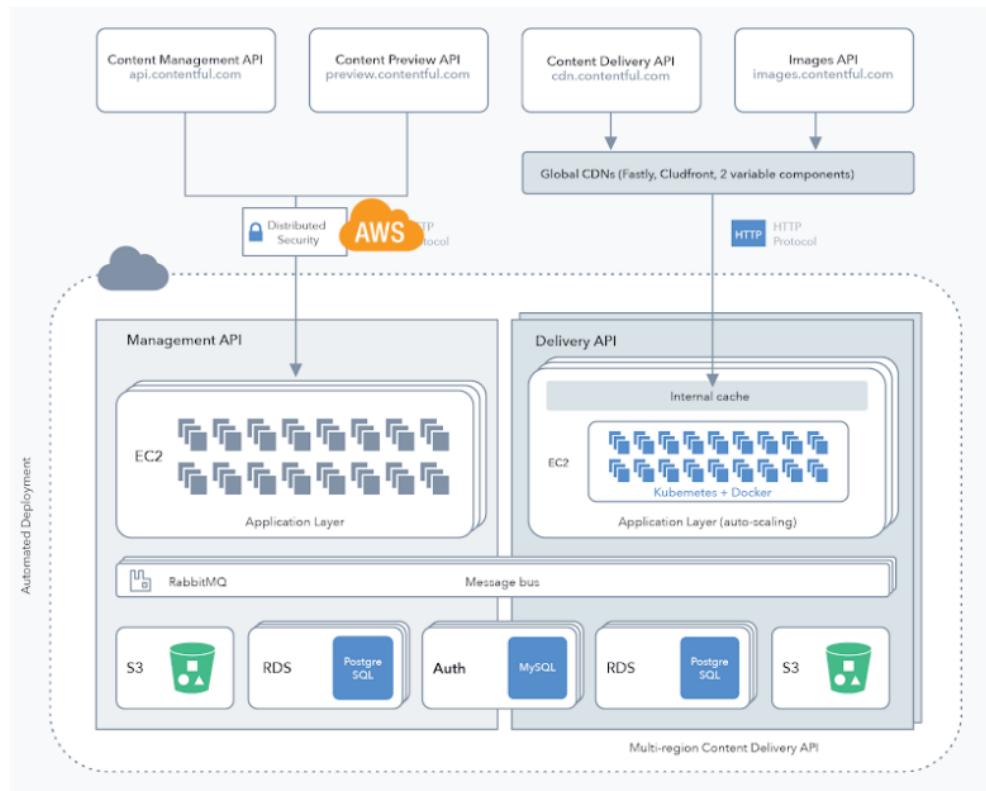
And on top of that API, you are free to use whatever framework, methodology and language you want.



POWER CONTENT ACROSS ALL YOUR DIGITAL CHANNELS

On the left side of the diagram above you see how content flows into Contentful. Developers either use the Content Management API directly, or CLI tools we provide built on top of the Content Management API to create content models, perform automated testing, etc. Content authors typically do all of their content creation in the extensible Contentful web app. They can save content in draft mode, put new content through customizable moderation workflows, and preview draft content before it is published.

After content is published it can be retrieved through the Content Delivery and Images APIs, as well as the SDKs built on top of them.



CONTENTFUL RUNS AS A SAAS

Contentful runs as a SaaS on AWS. Contentful's cloud-native architecture supports auto-scaling so large spikes in traffic can be handled without any need for doing emergency upsizes of server hardware. To achieve high reliability, Contentful runs in three separate availability zones of an Amazon region. Enterprise customers have the option to run in their own separate, dedicated instance of Contentful's infrastructure.

For companies accepting credit card payments, Contentful offer full PCI DSS compliance. The data centers used for storing your content and allowing it to be delivered to your users are certified for compliance with the ISO 27001 standard. And of course we are GDPR-compliant.

Scale and High Availability platform types are both PCI DSS compliant. Multi-Region Delivery Infrastructure (MRDI) is also an option for enterprise customers, where, in the highly unlikely event of an AWS region failure, Contentful will automatically route traffic to a second AWS region. Contentful offers up to a 99.99% SLA for customers on MRDI.

CONTENTFUL USE CASES

Keep in mind

While Contentful is used across a large number of vertical industries in many different usage scenarios, there are some use cases that are not a good fit:

- User-generated content
- Static sites that regenerate extremely frequently
- WYSIWYG author experience
- Countries that censor content

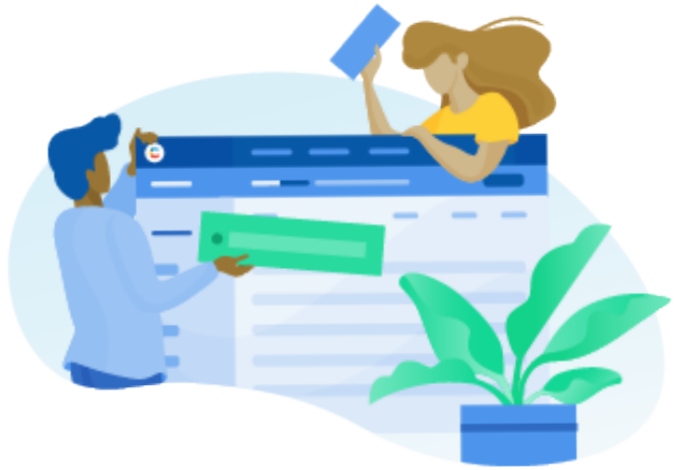
User-generated content has the potential to flood our CMA APIs with large bursts of content posts and exceed the rate limits designed for editorial teams, not thousands of concurrent users. While you could consider adding a proxy server between the posting of user-generated content and Contentful, it is often best to use purpose-built SaaS solution for this.

Static site generators create fast websites and have great SEO. However, it can often take several minutes to generate new sites after content changes. Sites that have large editorial teams with round-the-clock content changes are often not a good use case for them.

Editors love WYSIWYG interfaces, but they have two big problems:

1. They generate HTML which cannot be easily rendered on non-browser devices
2. They generate large blobs of unstructured content that do not promote reusability.

Finally, using Contentful in countries that impose a high degree of censorship on



websites is often not a good use case for us.

Be excited

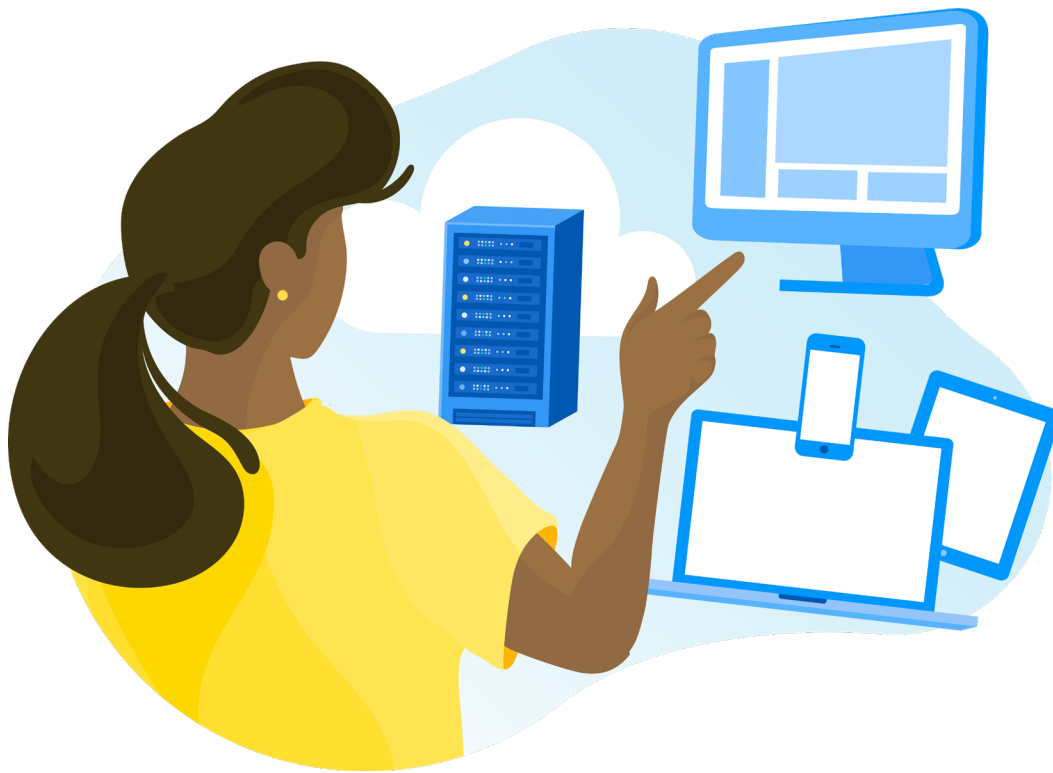
These are examples of some of the very best use cases for Contentful:

- Enabling content operations at scale
- Building engaging, cutting edge applications
- Enabling the digital transformation journey

There's nothing more important than development speed and velocity for forward-thinking companies today. Contentful's API-first model makes it easy for developers to learn how to use the platform. They can typically be making API calls in 30 minutes.

Contentful's multiple environments, webhooks and content migration API let developers treat content as code and enable content operations at scale.

The web is no longer just HTML pages in web browsers. Contentful lets developers render content on any type of digital device, and, with its easy-to-build UI extensions, integrate with modern personalization engines, DAMs, AI tools, etc.



Content Delivery Architectures

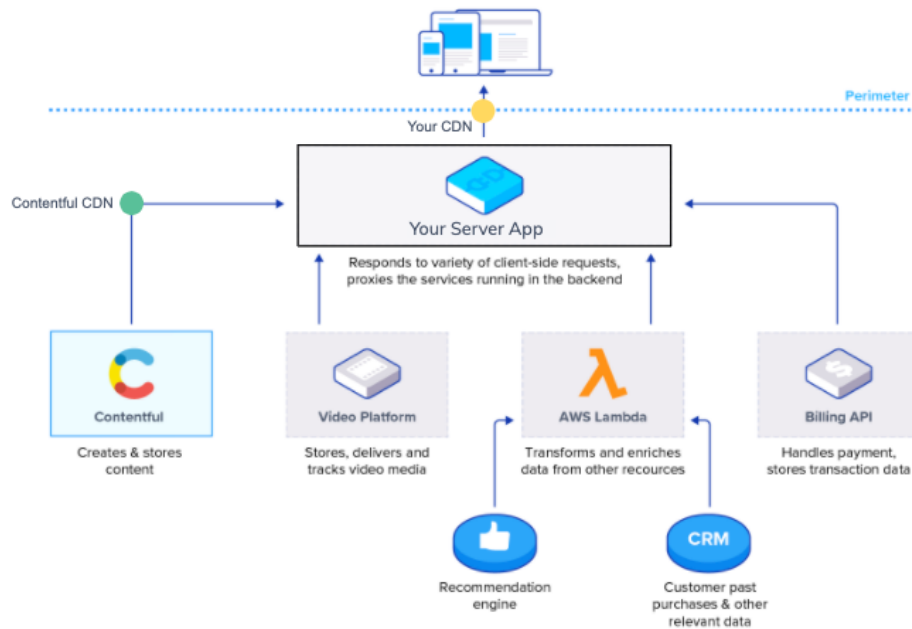


WHERE AND WHEN DOES RENDERING HAPPEN?

Legacy CMSes typically only support a single delivery architecture: dynamic on server. This architecture makes database calls to retrieve content. A templating language is then used to render an HTML page, and send the page back to the requesting browser.

In contrast, Contentful supports a variety of delivery architectures:

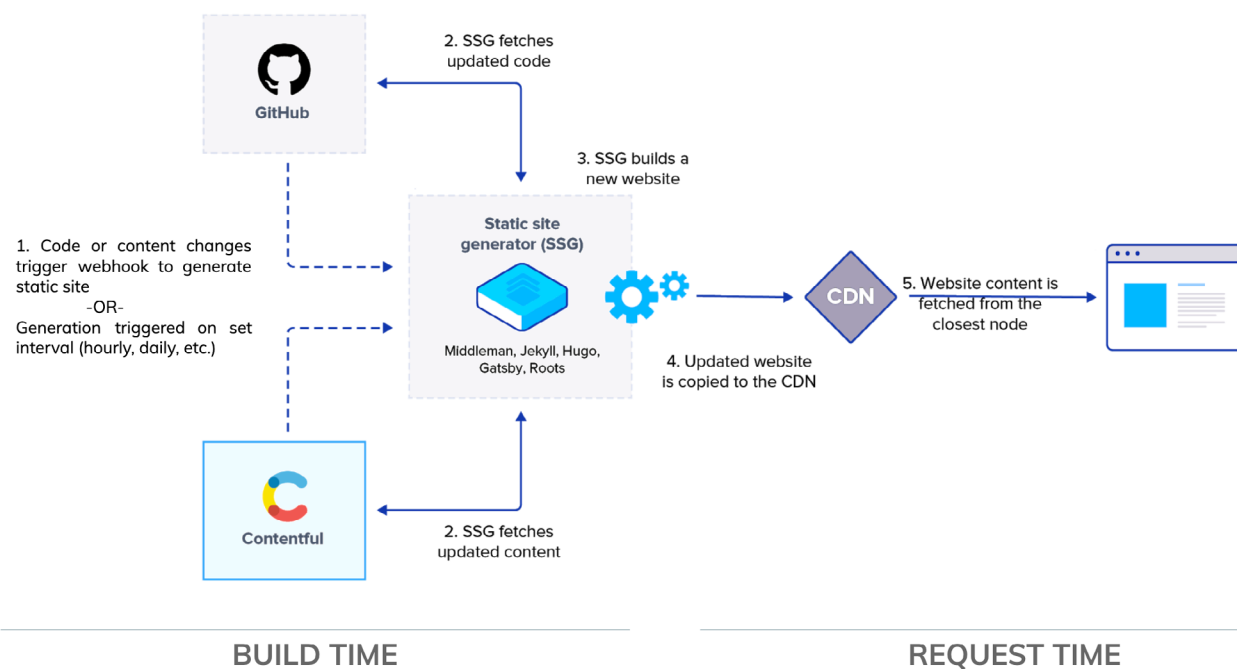
1. **Dynamic on server:** A server-side app receives a request from an end user, retrieves content from Contentful (and possibly other data sources), renders the content, and sends it back in response.
2. **Static site generator:** A static site generator fetches content from Contentful at timed intervals and then builds the rendered HTML pages and stores them on a static web host.
3. **Dynamic on device:** A client-side app using the Contentful SDKs requests content from the Contentful APIs and renders the content on the device.



DYNAMIC ON SERVER

Dynamic on server architecture (such as backend for frontend or server-side rendering) is based upon some kind of server (typically based on Java or Node.js), which makes API calls to Contentful and often to other web-based services. Content from the various API calls are combined and then rendered for the requesting device, such as generating an HTML page for a web browser.

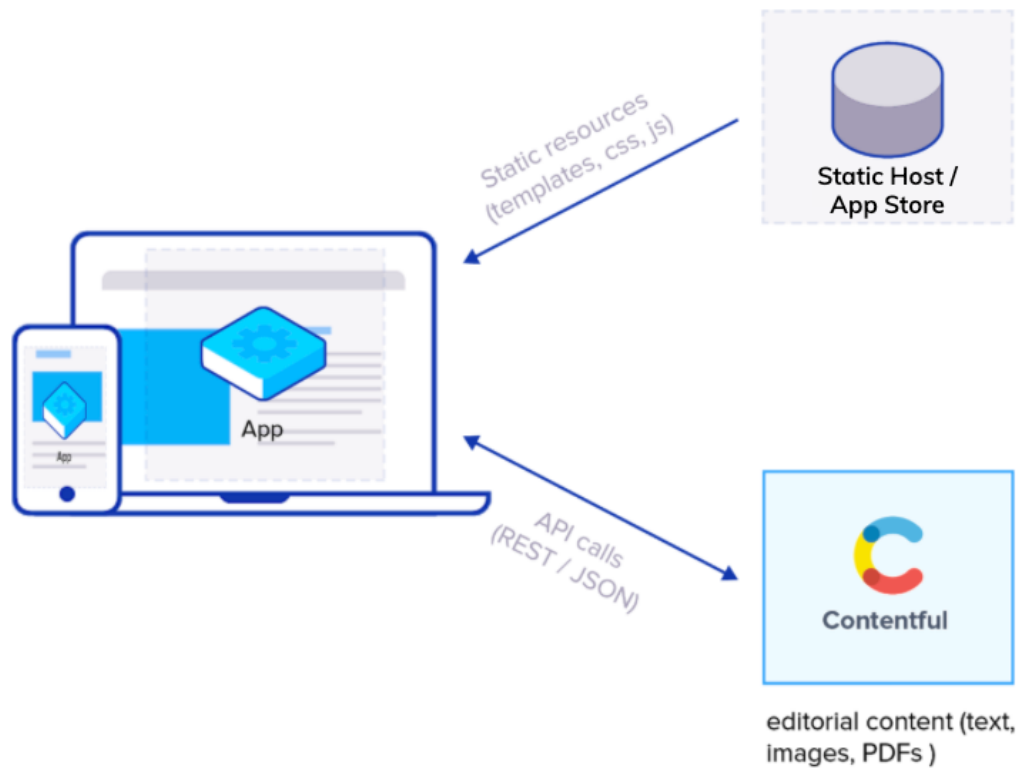
PROS	CONS
<ul style="list-style-type: none"> • Lots of languages and framework options • Reactive - no stale data • Easier SEO • Secure secrets (API tokens not exposed to end users) 	<ul style="list-style-type: none"> • Server-related scalability (\$\$\$) • Higher page latency • Requires external caching or CDN • Potential single point of failure



STATIC SITE GENERATOR

Static site generators (SSG) are a type of delivery architecture that is well-suited to Contentful's API-first architecture. An SSG such as [Gatsby](#) retrieves content from Contentful, renders an HTML page server-side, and then ships the fully rendered pages to a CDN. Since users are accessing web pages from the worldwide caches of CDNs, site response is blazingly fast, resulting in benefits such as higher conversion for e-commerce sites and higher SEO scores. This [video](#) shows how Contentful, Gatsby, and Netlify work together, via webhooks, to rebuild websites after a content author edits a piece of content in Contentful.

PROS	CONS
<ul style="list-style-type: none"> • Easier SEO • Offline capability • Static HTML is fast and simple to serve • Secure secrets (API tokens not exposed to end-users) 	<ul style="list-style-type: none"> • No immediate preview/potential for stale data • Web page-centric • Generation time can be problematic for frequent content changes • Requires external hosting, caching, and CDN



DYNAMIC ON DEVICE

Contentful also supports the Dynamic on Device architectures, where the device (e.g. native mobile app) will directly make Contentful API calls and combine the result with API calls to other services. Single Page Applications (SPA) are the best known example of this type of delivery architecture.

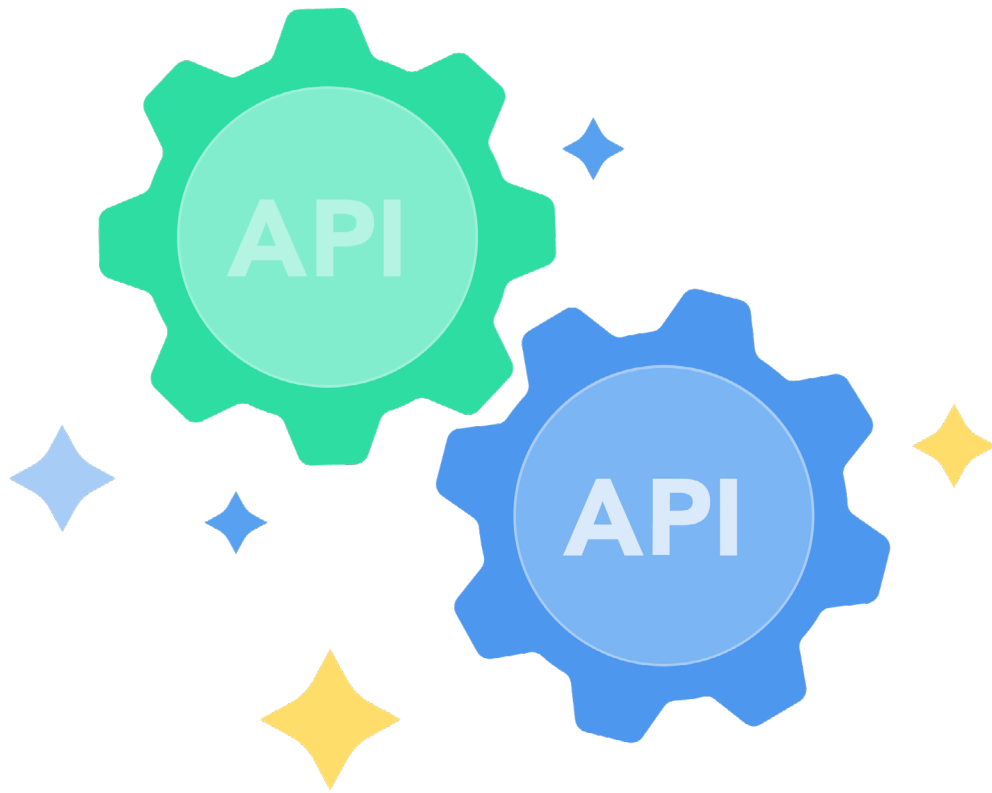
PROS	CONS
<ul style="list-style-type: none"> • Simple • Reactive - no stale data • Fully leverages the Contentful CDN 	<ul style="list-style-type: none"> • SEO support requires more effort • Possible browser compatibility issues

1**Dynamic on Server****2****Dynamic on Device**

DYNAMIC HYBRID WITH NODE.JS

A hybrid approach of using a JavaScript framework such as React, which supports isomorphic execution on both the server and front-end device, combines the benefits of both the Dynamic on Server and Dynamic on Device approaches. Initial page rendering can be executed on the Node.js server and sent to the device, with the JavaScript framework taking over rendering from that point on.

PROS	CONS
<ul style="list-style-type: none">• Lots of languages & framework options• Reactive - no stale data• Easier SEO• Fully leverages the Contentful CDN	<ul style="list-style-type: none">• Complexity of architecture• Potential single point of failure



Contentful Core APIs



FOUR CORE REST APIS

Contentful has four core REST APIs: Content Management API, Content Delivery API, Content Preview API, and Images API. While this sounds like a lot, in reality the Content Preview and Content Delivery APIs are exactly the same - they just have different end points and different behavior. And the Images API is essentially a single HTTP request with different query parameter options.

There are many benefits to splitting up the APIs into these four segments. The CMA is the only one that has write-access to Contentful's infrastructure, and we protect your content by putting a web application firewall in front of it. There's no need to do that with the read-only APIs.

Content authors and developers always want the latest content. They don't want to deal with stale, cached content. At the same time, content authors and developers need to provide users with the fastest performance possible. That's why we've put CDNs in front of the Content Delivery API and Images APIs, while there is no CDN in front of the Content Management API and Content Preview API. This [video](#) provides an overview of all four APIs.

CONTENT MANAGEMENT API

The Contentful web application is built on top of the [Content Management API](#). Developers can access the CMA securely via HTTPS, and it will be available to clients authenticating with an access token.

To access the Content Management API and store content created in your apps, you need a content management token that represents the desired account of your user. This token will have the same rights as the owner of the account.

There are two types of content management tokens :

- Personal access tokens - use if you're using the Content Management API to access data from your own Contentful user account
- OAuth tokens - Use if you're building a public integration that requests access to other Contentful user's data

You can create personal access tokens using the Contentful web app. Open the space that you want to access (the top left corner lists all spaces), and navigate to the APIs area. Open the content management tokens section and create a token.

If you are creating apps for changing content stored in Contentful, you will need to create a custom OAuth application.

An OAuth 2.0 application has a number of benefits:

- OAuth 2.0 access tokens are linked to your app
- You can request the correct OAuth 2.0 scopes for your application (content_management_read or

content_management_manage)

- You can specify a custom redirect URL that will receive the access token as part of the URI's hash fragment
- You can specify a custom name and description
- You can specify whether your application is confidential or public

CONTENT DELIVERY API

[The Content Delivery API](#), available at [cdn.contentful.com](#), is a highly available, highly scalable, read-only API for delivering content to apps, websites, and other channels. Content is delivered as JSON data, and images, videos and other media as files.

The Content Delivery API is available via a globally distributed CDN. The server closest to the user serves all content, which minimizes latency and especially benefits mobile apps. Hosting content in multiple global data centers also improves the availability of content.



CONTENT PREVIEW API

In addition to the Content Delivery API for published content, the [Content Preview API](#) is for previewing unpublished content as though it were published. It maintains the same endpoints and parameters as the Content Delivery API, but delivers the latest draft, updated, and published entries and assets.

This is a read-only API. Calls to this API are not cached so preview will always have the most up-to-date content.

CONTENTFUL IMAGES API

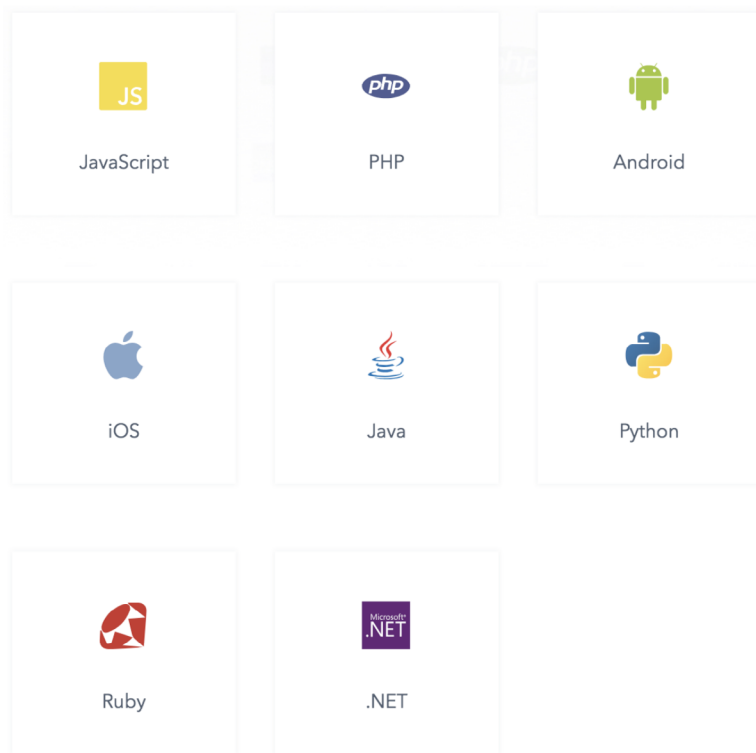
Most legacy CMSes require you to upload multiple renditions of an image at different resolution and quality settings. With Contentful's read-only [Images API](#) you can upload a single, high-resolution image and then use the Images API to request different versions with scaling, cropping, quality level and file format changes. The derived images are created in real-time and then cached in a CDN for future requests.

Contentful allows you to request an image in a different file format from what you uploaded. You can convert between the following raster image formats: JPEG (including progressive JPEG), GIF, PNG, and WebP. WebP images are often 45% smaller in size than the equivalent PNG version.

Note that all of the above file formats are raster image types. You can upload images in vector file formats such as SVG, but they will be treated as binary assets and cannot be used with the Images API.

In addition to scaling and cropping images, the Images API allows you to choose the focus area for cropping (e.g. top left of the image). You can also instruct the API to do facial detection for the cropped focus area. The default is the center of the image.

The Images API supports returning an image with a varying level of quality with the "q" parameter. Values can be from 1 to 100, with 100 being highest. Smaller quality values will result in small file sizes. The quality parameter is ignored for 8-bit PNGs.



SDKS

While you can directly call Contentful's REST APIs using any programming language, you will generally want to use one of our eight SDKs built on top of our REST APIs.

Using a language-specific SDK has a number of benefits over directly calling the REST APIs. There is typically a lot of boilerplate code required to set up and make the API call, check the HTTP response of the result of the call, handle errors consistently, and so on. That's why the use of an SDK will significantly speed up your development, while also helping ensure good coding practices.

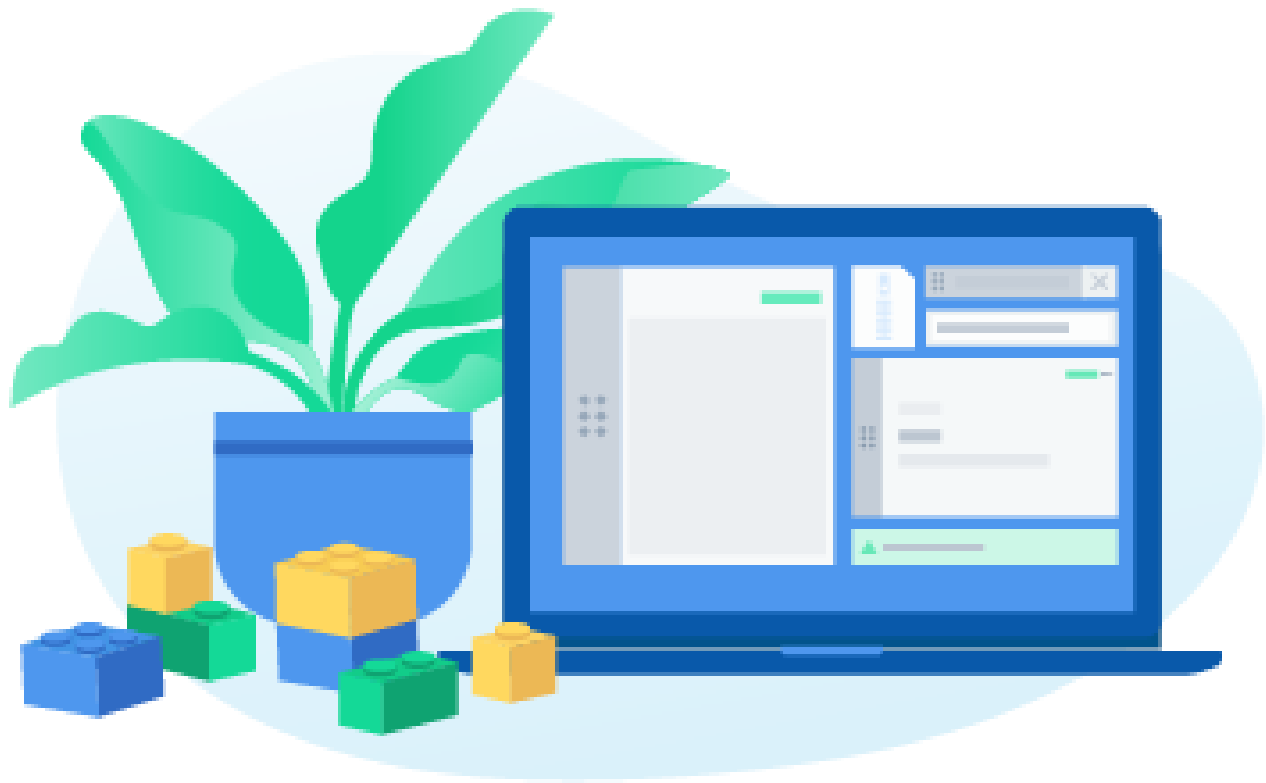
We have SDKs for JavaScript (Node.js), PHP, Android (Java), iOS (Swift), Java, Python, Ruby and .NET.



GRAPHQL CONTENT API

GraphQL is a query language for APIs, and a server-side runtime for executing queries by using a type system you define for your data. With GraphQL you can state precisely what data you want returned in a query, preventing excessively large amounts of data being returned. GraphQL schemas are strongly typed, so you can reliably know what type of data is returned. GraphQL also supports schema stitching, allowing a single GraphQL call to multiple underlying GraphQL APIs.

Contentful's implementation of GraphQL is built on top of our core REST APIs. There's no need for you to create a GraphQL schema – we generate it for you based upon your content model. Since our GraphQL implementation is built on top of our Delivery APIs, you get the benefit of high performance due to the caching of delivery API requests using multiple CDNs. While you can access draft content through our GraphQL implementation, we don't support underlying Content Management API calls that would allow for creating or updating content.



Content Modeling



STRUCTURED CONTENT

To understand content modeling, you have to understand structured content.

As developers and content modelers, we want to make sure we are getting the right content to the right audience. But, often, when we think about our content, we focus too much on where the content will live. We fixate on the interface: a website, a mobile device, a digital screen. But this interface-centric thinking is what often leads us to trouble – endless redesigns, siloed content, and wasted time and money.

At its simplest, structured content is about separating the “content” from its “context.” To separate content from context, think of your content as pieces or chunks of data and think of your context as the interface.

Imagine an article. The “content” is the pieces of data, such as the title, the subtitle, the image, etc. The “context” is the mobile device, a computer, a newspaper, or any other interface.

FROM STRUCTURED CONTENT TO CONTENT MODELING

So how do we create structured content at Contentful? We create content models.

Before we go any further, here's some basic vocabulary you should master:

- **Content Model** is the overall architecture of your content. It divides up your project's content into chunks we call content types. In other words, a content model is a collection of content types.
- **A Content Type** is the structure or container for a piece of content. A content type is made up of fields. Content types are always made from scratch by your development team. There are no out-of-the-box content types.
- **Fields** (aka attributes) are different properties or characteristics for a piece of content. Each field will be assigned a data type such as text, rich text, number, date, location, media, boolean, JSON object, or reference.
- **An Entry** is a piece of content based on a content type. You might think of an entry as an "instance" of that content type. You might have hundreds or thousands of entries based on a single content type. Entries will be created by content authors.
- **A Reference** is a link between two content types via a field (specifically a "reference field"). References create relationships between content types.
- **An Asset** is any media file that has been uploaded to Contentful, such as images, video, audio files, .pdfs, and more. Content types can include fields for "media," which allow content authors to then link to uploaded assets.

Name	Description	Fields
Call to Action		3
Header		3
Headline		3
Hero Image		3
Hotel Landing Page		4
Image Carousel		2
Image with Caption		3
Link		3
Paragraph		4
Paragraph with Headline		3
SEO Metadata		5
Set of Three		2
Set of Two		2
Text with Image		4

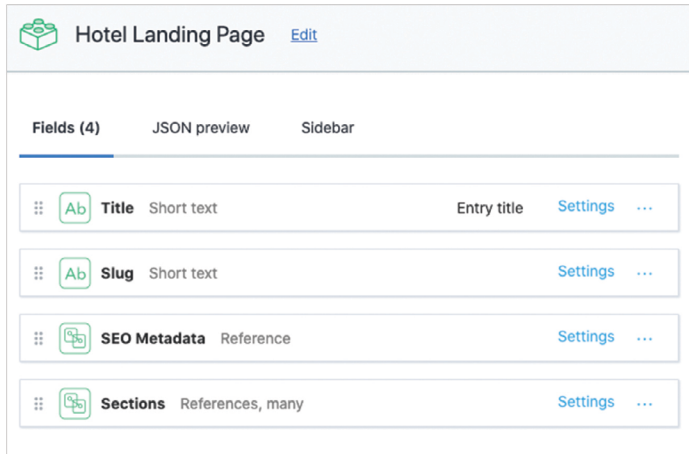
A sample content model for the landing page of a hotel's website.

Each item here is a content type.

CONTENT TYPES AND FIELDS

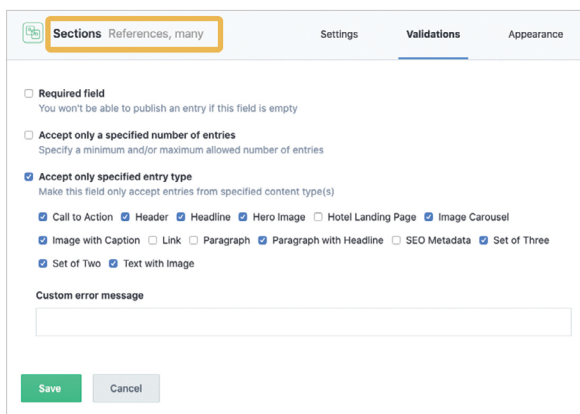
The building blocks of our content models are content types. Content types are what help us break up our “blobs” of content into reusable components.

For example, the image below is from the Contentful web app. It is a content type called “Hotel Landing Page” that belongs to the content model we showed on the previous page. This content type has four fields. “Title” and “Slug” are text fields while “SEO Metadata” is a one-to-one reference field and “Sections” is a one-to-many reference field.

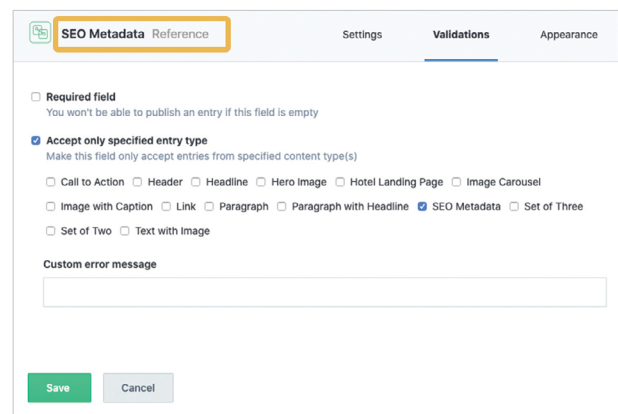


A content type called “Hotel Landing Page” with four fields

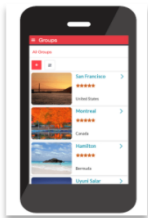
The one-to-one reference field only allows content authors to link to a single entry for the content type “SEO Metadata” while the one-to-many reference field allows content authors to link to multiple entries for a number of different content types. To create these parameters in the Contentful web app, navigate to the specific field you want to modify and go to *Settings > Validations*. The images below show the validations for these two reference fields.



Validations for a one-to-many reference field “Sections”

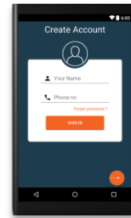


Validations for a one-to-one reference field “SEO metadata”



Content

- Text
- Images
- Audio/Video
- Geo location



Microcopy

- Field labels
- Button text
- Navigation



Localized Content

- Localization strategies
- Translation permissions/workflows



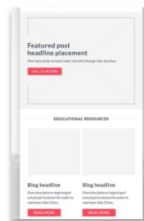
Content Metadata

- SEO & Social Media
- Images as content types
- Tags for navigating the content model



Process Metadata

- Site functionality (sort, search, etc.)
- Editorial workflow
- Webhooks



Content Modules ("Assemblies")

- Atomic design – chunks not blobs
- Giving content creators control over layout
- Visual design systems

WHAT SHOULD YOU STORE IN CONTENTFUL?

The image above shows the different categories of "stuff" you can store in Contentful.

You can manage a variety of content in Contentful. Contentful offers many different types of fields with different functionality and options. For example, text fields can be short or long, full-text searchable or not.

It's important to note that content can be more than just traditional editorial content. It can be process metadata, such as content moderation workflow data that would otherwise have to go into code.

Microcopy is buttons, fields, navigation. These are little bits of copy scattered across websites or applications that need to be maintained or even localized. You can do microcopy from Contentful instead of baking it into code so it can be freed from the development cycle.

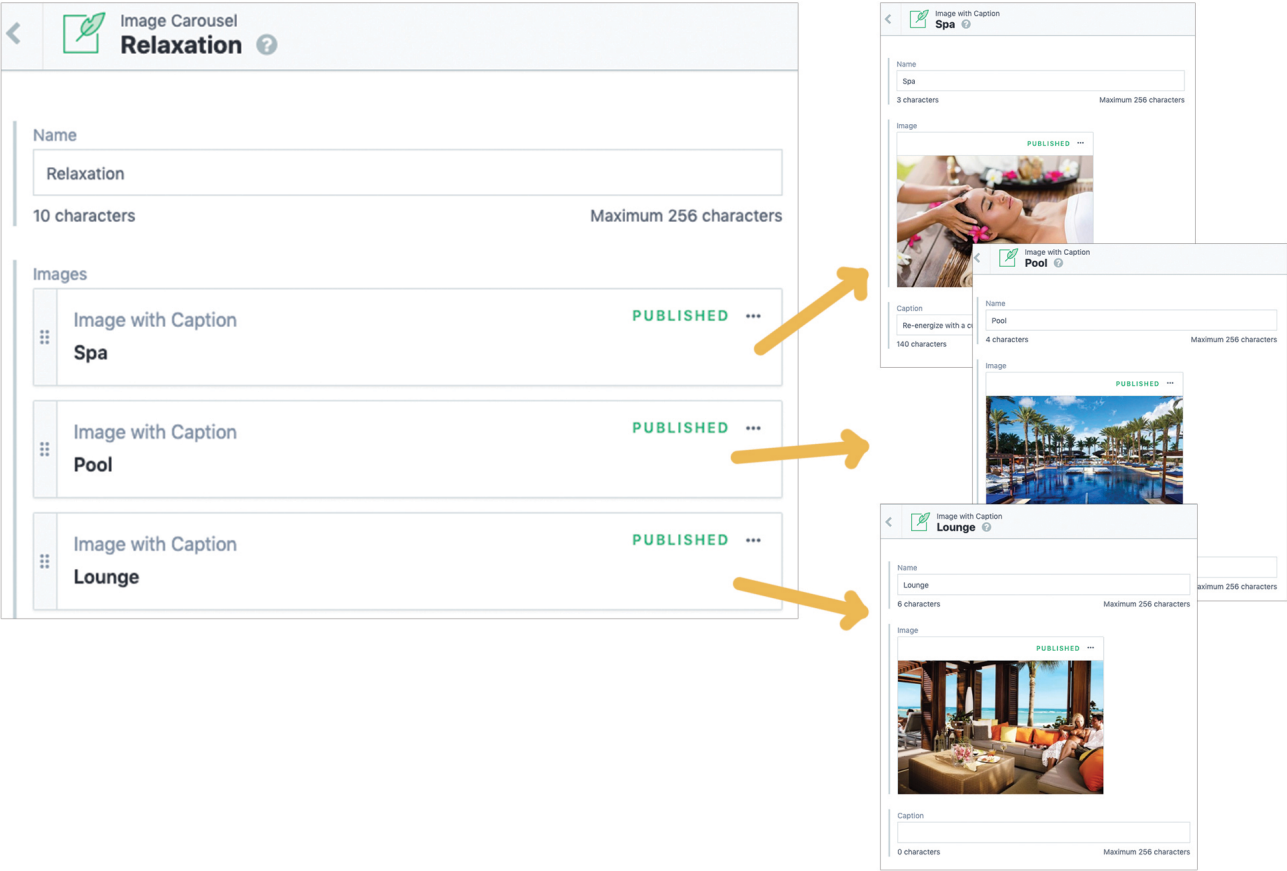
You can include fields that store SEO and social media data such as the title and meta description for a page, Open Graph meta tags to ensure the correct thumbnail image is used for sharing content on social media channels, as well as additional metadata for images to help optimize SEO.

TOPICS VS ASSEMBLIES

To create a well-structured content model we want to break up our content into well-defined fields and distinct content types that reference one another. At Contentful, we categorize content types as either "topics" or "assemblies." The chart below summarizes the key differences between the two:

TOPICS	ASSEMBLIES
<ul style="list-style-type: none">Individually authorablePure contentNo presentation optionsBuilding blocks for assemblies	<ul style="list-style-type: none">Not individually authorableContains topics or other assembliesCan include presentation options

As shown in the image below, one common example of an assembly is an image carousel. On the left we have an entry for an assembly "Image Carousel" and on the right we have three entries for the topic content type "Image with Caption." The entry for this "Image Carousel" assembly contains references to entries for the topic content type "Image with Caption." This allows content authors to create multiple instances of it using different photos for each instance while the front-end code renders the carousel appropriately for each device or context.



Typically, one of the first steps in content modeling is identifying reusable content and assemblies.

There are two strategies for this:

1. Modeling content by focusing on the relationships inherent in the content itself
2. Starting with a display mockup and identifying the content model required to build that display (while also thinking ahead to future needs)

Either of these approaches can lead to a good content model, depending on the skills and preferences of those involved. In practice, content modeling often involves bouncing back and forth between these two perspectives.

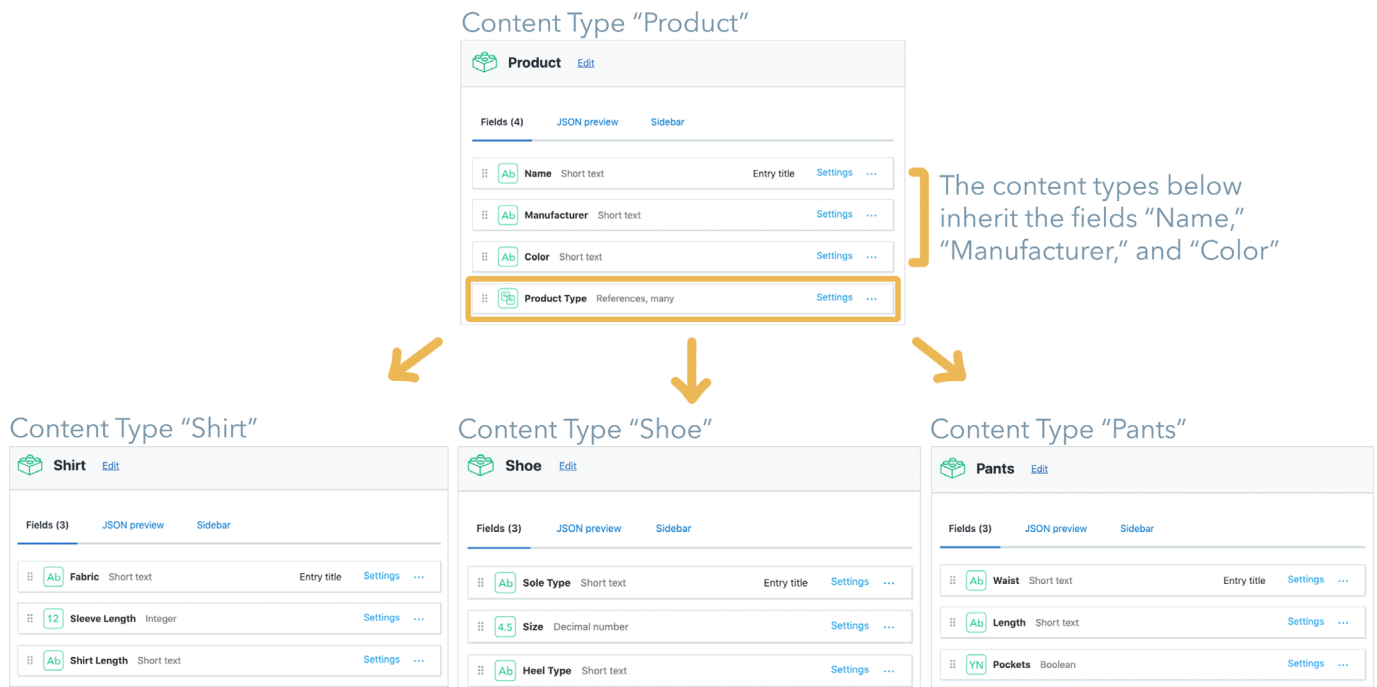
FIXED VS FLEXIBLE ASSEMBLIES

Depending on how you create your content model, you can give content authors leeway to create a variety of layouts with your components (often referred to as a “design system”) or you can lock down a layout into a template where the components always appear the same way.

One way to give content authors more control over the layout of components is to use a “flexible” assembly with a one-to-many reference field. This allows content authors to reorder entries by simply dragging and dropping.

If you want to lock down a layout into a specific template, you can create a fixed assembly with a one-to-one reference field. In this scenario, the content author can only link to a single entry and as a result cannot reorder content by dragging and dropping. This [video](#) shows the differences between fixed versus flexible assemblies.

In practice, many assemblies are a hybrid of both “fixed” and “flexible” with both one-to-one and one-to-many reference fields. The content type we looked at before, [“Hotel Landing Page,”](#) is a good example of a hybrid assembly.



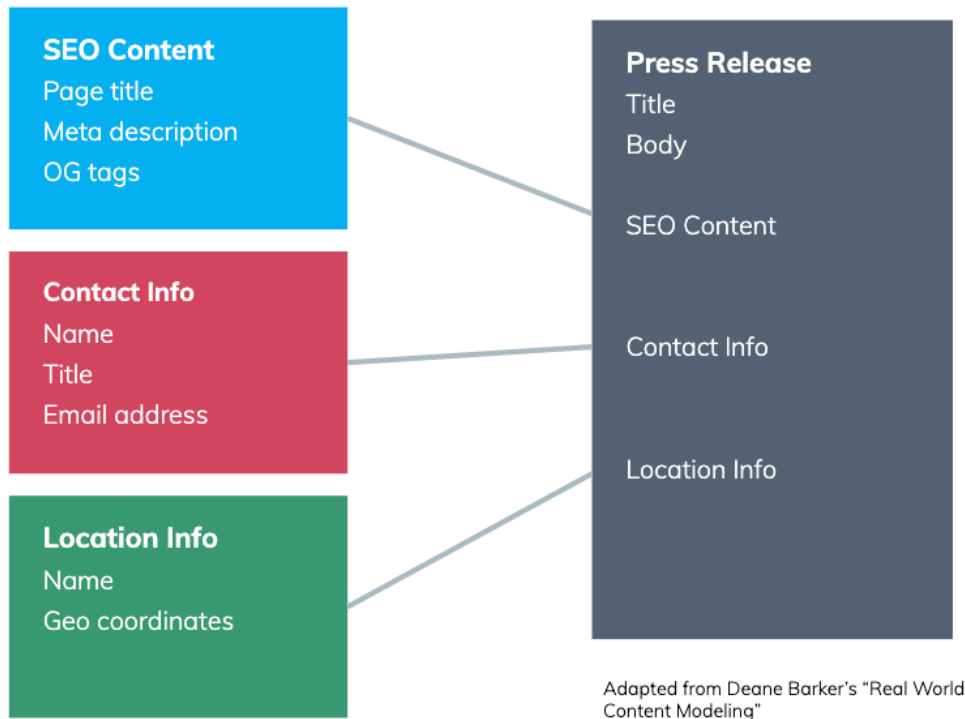
INHERITANCE

In some instances, it's useful to have some content that is shared across multiple content types (for consistency) while other content is specific to a single content type.

Similar to the concept of inheritance in software programming, at each lower level, you only need to specify things that are different from the levels above. In Contentful, you can achieve this by linking content types using reference fields. With inheritance you can dramatically reduce the number of fields you need on each type. Common attributes get moved up to parent types which are inherited, and maintain these common fields from the top down. Contentful only supports single inheritance.

Imagine that you are starting an ecommerce site to sell shoes. Your first content model might have a single "Product" content type with fields for "Name," "Manufacturer," "Color," "Shoe Size," "Sole Type," and "Heel Type." At this point it wouldn't make sense to add the additional fields needed by pants and shirts to the "Product" content type.

Instead, as shown in the image above, you can put fields that are common to all products in the "Product" content type, and create additional content types for specific kinds of products with fields that are specific to that content type.



COMPOSITION

Unlike inheritance, composition does not have content types that inherit attributes from a single path back up the tree. With composition, your content types are “free floating” and can be inserted into any content type.

While inheritance is a very powerful technique to use when designing a content model, most models will primarily be based around the composition of content types. In the image above, the “Press Release” content type is an assembly, while “SEO Content”, “Contact Info” and “Location Info” are all topics.



LOCALIZATION

It's critical to be able to create and share content with a global audience.

In Contentful we use the term locale to describe the different versions of content for different locations or, more simply, a language-region pair. Contentful enables publishing content in multiple locales with localization.

Every Space in Contentful has its own set of locales, and each locale is uniquely identified by its ISO code (e.g. en-US). It's important to note that locale does not just mean language. While German is a single language, there are many different German locales. For example, there's de-DE for German in Germany, de-AT for German in Austria, de-CH for German in Switzerland, and so on.

There's always one default locale defined when you create a space, shown by default in the Contentful web app and used for Content Delivery API queries that do not request a specific locale.

You can add a new locale to a space in the Contentful web app or by using the Content Management API. Contentful also supports custom locales, which some Contentful users employ to target audiences at the state/province level. This [video](#) shows how to add and set up locales.

FIELD-LEVEL LOCALIZATION

Field-level localization is built into Contentful. Using this design pattern, the fields which have localization enabled will be duplicated for each locale that is selected by the content author. For example, if you have two fields, one for title and another for body, there will be a duplicate of each field for every selected locale. The image below shows an entry for a content type where field-level localization has been enabled for the fields “Title” and “Body.”

To set this up in the Contentful web app, navigate to the field you want to have content authors be able to localize and go to *Settings*. Here you will want to check the box that says *Enable Localization of this Field*.

This design pattern is best for cases where you want publishing of locales to be done synchronously and you have smaller content types with fewer fields. Larger content types with many duplicated localized fields might start to become too cluttered for content authors.

This video shows how to create a content model with [field-level localization](#).

The screenshot displays the Contentful web app editor interface. At the top, the article title is 'The life of Jane Austen'. Below this, the editor shows three distinct sections for localization:

- Title - English:** A text field containing 'The life of Jane Austen' with a character count of 23 and a maximum of 256 characters.
- Title - Spanish:** A text field containing 'La vida de Jane Austen' with a character count of 22 and a maximum of 256 characters.
- Body - English:** A rich text editor area containing the text 'Jane Austen...'.
- Body - Spanish:** A rich text editor area containing the text 'Jane Austen...'.

Each section has its own toolbar with formatting options (bold, italic, underline, link, unlink, list, quote, indent) and an 'Embed' button.

When field-level localization is used, each field is duplicated for each locale that is enabled by the content author.

In this case, the locale Spanish (es) has been turned on in addition to the default locale of English (en).

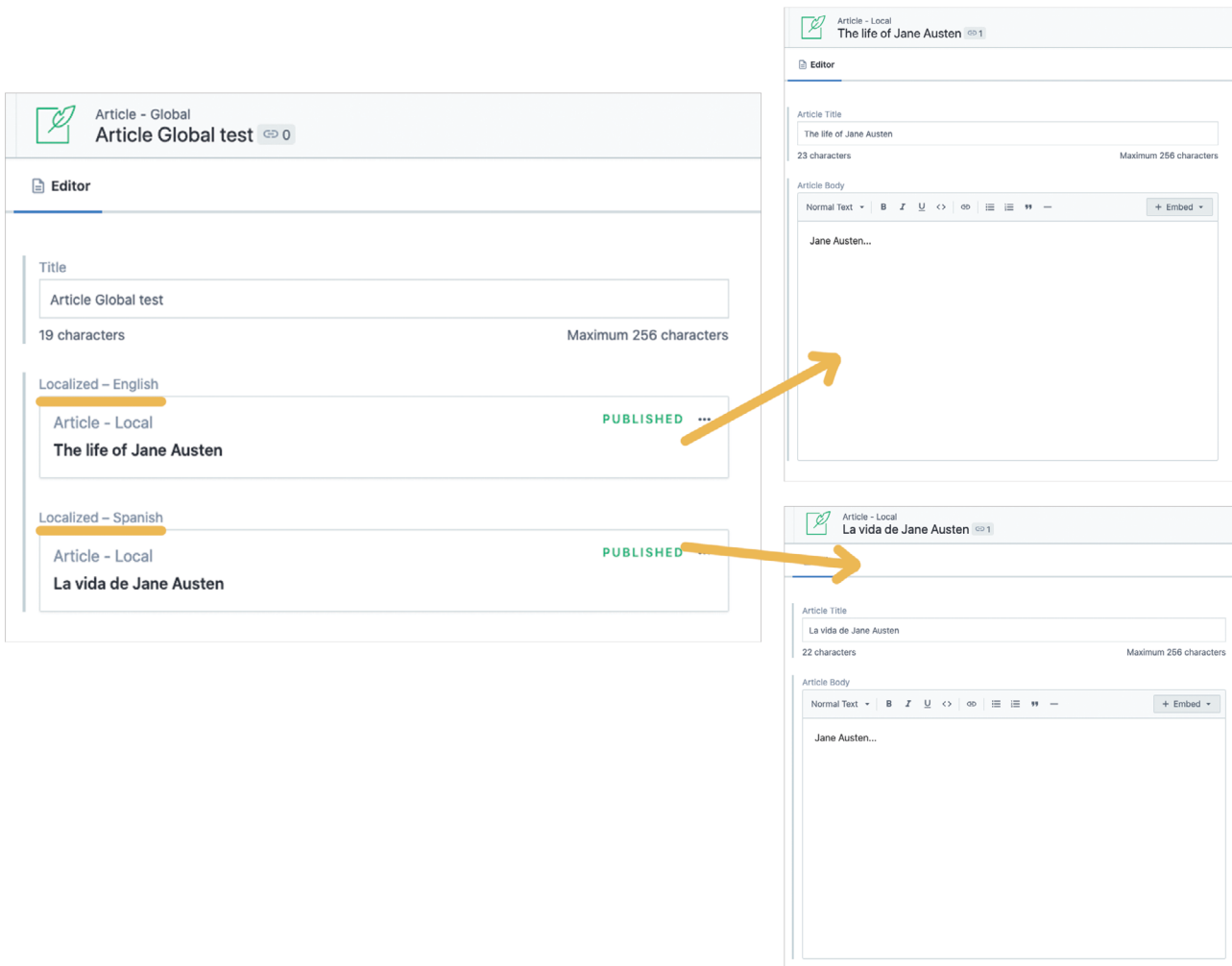
ENTRY-LEVEL LOCALIZATION

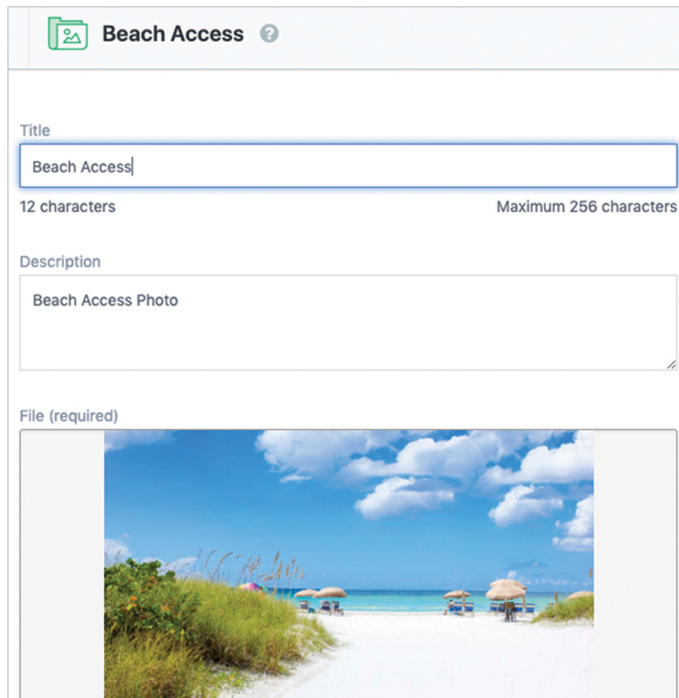
Entry-level localization is not built into Contentful. With entry-level localization, you will need to set up global-level content types that contain fields for common elements and local-level content types that contain fields for localized content.

As shown in the image below, the global-level content type should be linked to the local-level content type via a one-to-many reference field. The reference field in the global-level content type should be localized via the field's validation settings. This means the individual entries don't need to have their fields localized. The front end will query the global-level content type to find the appropriate localized entry.

Entry-level localization is best used for cases where you want your authors to be able to publish content asynchronously. For example, maybe you have a marketing campaign you want to roll out in different locales staggered over several weeks. With entry-level localization, you can have one content author publish content in US-English today and then another author publish content in DE-German next week.

This video shows how to create a content model with [entry-level localization](#).





The screenshot shows the 'Beach Access' asset creation form in the Contentful web app. The form has three main sections: 'Title', 'Description', and 'File (required)'. The 'Title' field contains 'Beach Access' and has a character count of 12/256. The 'Description' field contains 'Beach Access Photo'. The 'File (required)' section shows a preview of a beach photo.

Field	Value	Character Count
Title	Beach Access	12 / 256
Description	Beach Access Photo	-
File (required)	Beach Photo	-

An example of an asset uploaded to Contentful

Assets have only three fields

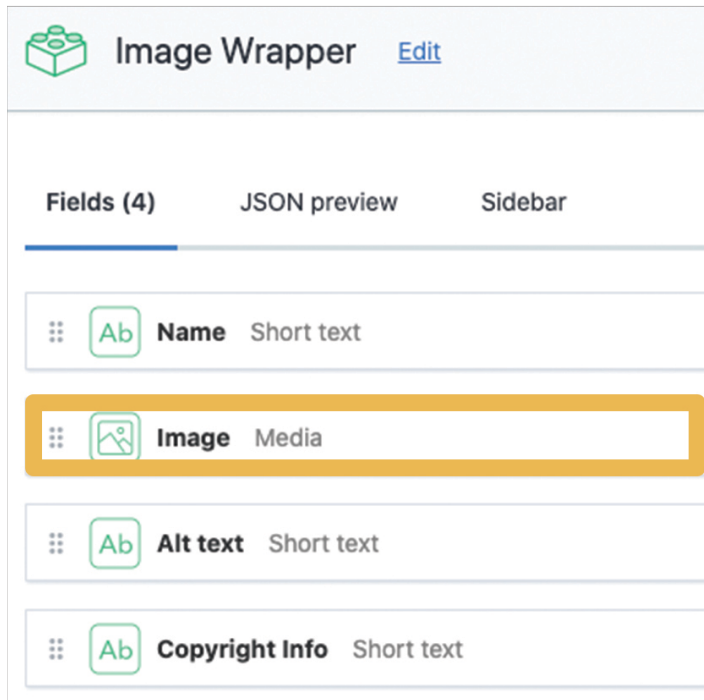
ASSETS

In Contentful we call any type of media file an asset. Assets include images, videos, audio files, slide decks, .pdfs, and more. There are two ways you can upload and create new assets: from inside the Contentful web app or via our Content Management API. While any digital file can be uploaded to Contentful as an asset, only JPEG, GIF, PNG and WebP can be accessed with Contentful's Images API.

DEFAULT MEDIA HANDLING

One way of handling media is to use Contentful's default media tab in the web app. When content authors upload media this way, it will be turned into an asset and they will have three basic fields to complete: title, description, and file. These fields cannot be changed.

This [video](#) shows how media is uploaded and turned into assets via the media tab in the Contentful web app.



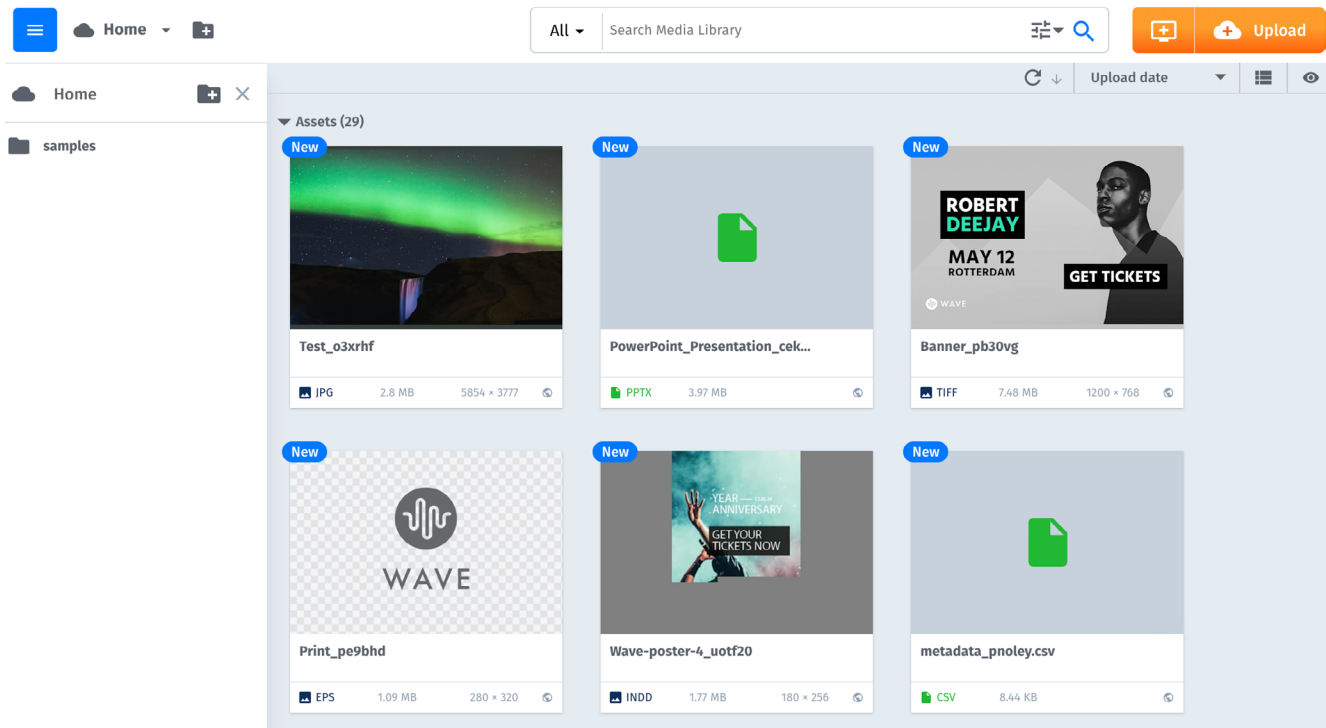
A content type called “Image Wrapper” that links to an asset via a media field called “Image”

MEDIA WRAPPER CONTENT TYPES

In many cases your content authors will need more fields for their media files than what is provided for an asset. For example, they might need fields for custom metadata such as alt-text or copyright. They might also want additional functionality for editing their media files.

When this is the case, we recommend creating a content type for your media that includes those additional fields and links to an asset. Think of this as a media wrapper that holds the asset but also holds additional information. If using this approach, you might want to check out the [Image Uploader UI Extension](#) for flattening and simplifying the editor experience.

This [video](#) shows how to create media wrappers in the Contentful web app.



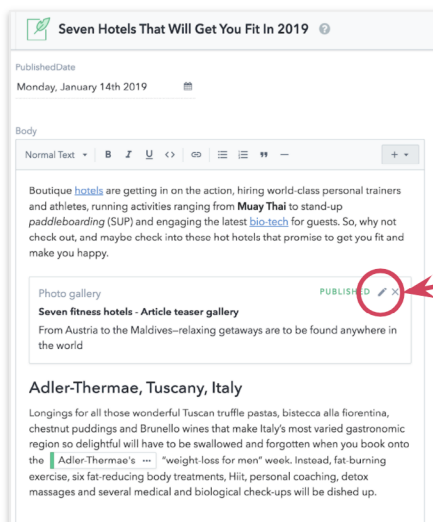
USING A DAM

If you are content modeling for an enterprise, you might want to seriously consider using a digital asset management (DAM) software to help you manage and deliver your digital assets.

DAMs not only provide a single, searchable source of truth for an organization's digital assets but also provide a high degree of governance, have specialized support for asset approval workflows as well as extensive metadata capabilities. Some DAMs, such as Cloudfire (featured in the image above), also support adaptive bit-rate video streaming on top of high performance image serving.

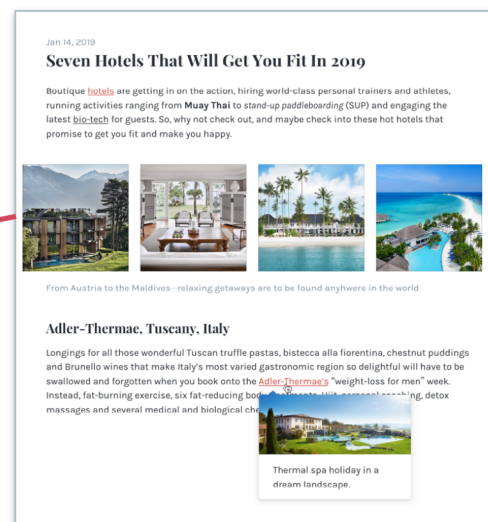
If your content authors are already storing and organizing their media in a centralized DAM, there is no need to abandon that system. Any third party media service can be used with Contentful as long as they offer an API you can query from a custom app. Create a custom app and pull in media from virtually anywhere.

If you are using Cloudfire or Bynder as your DAM, you are in luck! We have pre-built apps for Cloudfire and Bynder integration via in the Contentful marketplace. This [video](#) shows how you can integrate Cloudfire with Contentful.



Rich text editor view

Click the pencil icon
to edit the
embedded entry



Rendered view

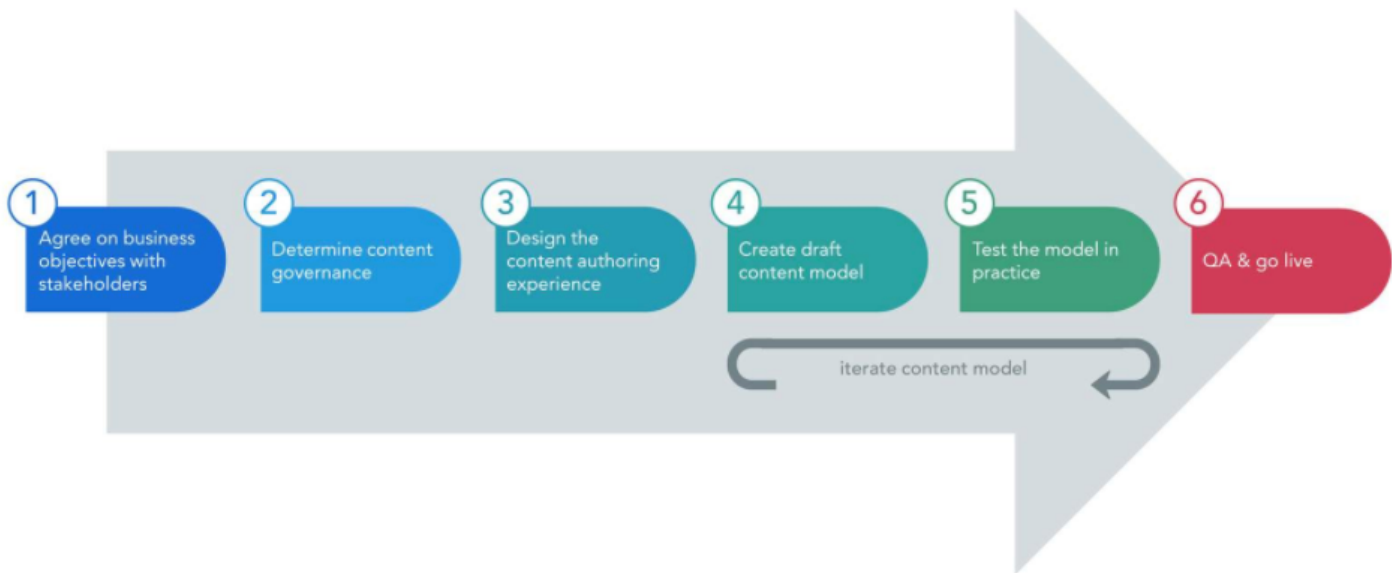
RICH TEXT

Rich Text is a field type that enables content authors to create rich text content, on par with traditional editors. Additionally, it allows entries and assets to be linked dynamically and embedded within the flow of the text.

Rich Text provides these capabilities while maintaining a rich format on the API response. The API response is in JSON format thereby eliminating the empty `<p></p>` tags (associated with an HTML response) or shortcodes.

The rich text field gives content authors more of a WYSIWIG or classic rich text editing experience. What makes it different from Contentful's plain text field is that you can embed pieces of content (entries) within a body of text. The image above shows what it looks like for a content author to edit an embedded entry in the rich text editor.

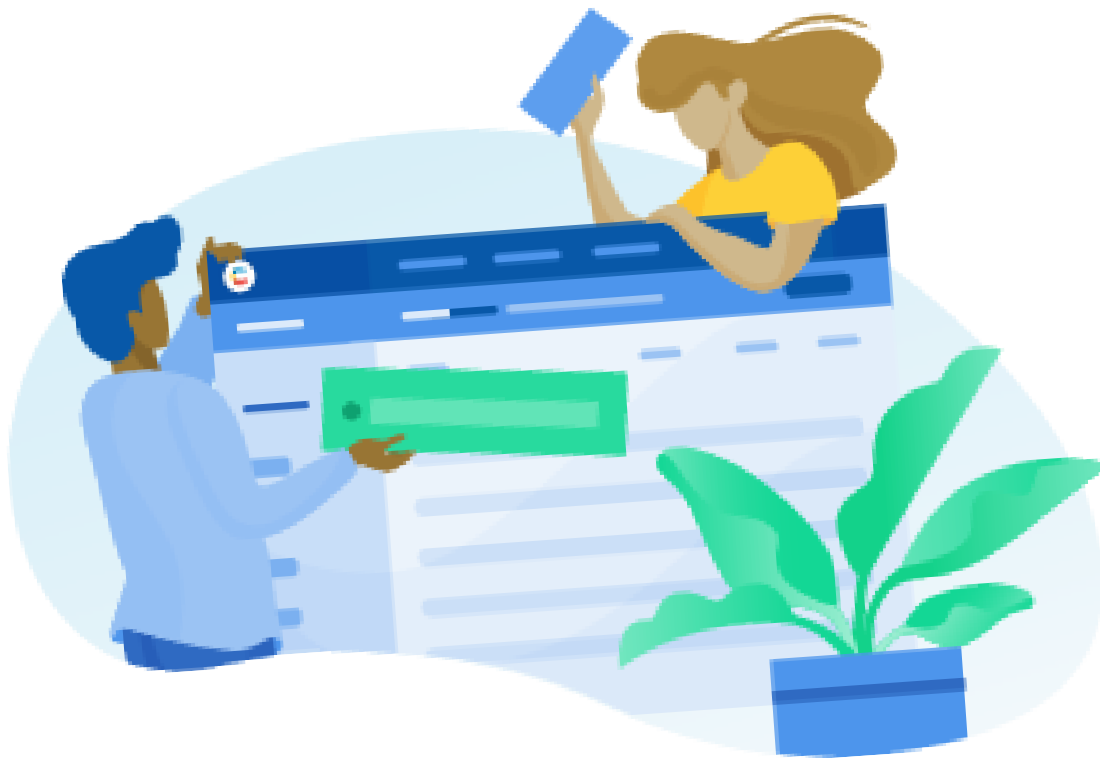
This [video](#) provides a detailed overview of everything you can do with the Rich Text field.



CONTENT MODELING PROCESS

This is the general methodology that we find customers follow to build their content models. Obviously for smaller teams, you can have shortened versions of this methodology or for bigger teams you can have more steps. Some customers leave their content models up to their developers, which is fine. However, the most successful customers understand that content modeling is an exercise in domain mapping. It is beneficial to have a number of stakeholders involved in the process, not just developers. Authors and designers can contribute important perspectives and help to understand how the content is going to be consumed and viewed.

It is hard to develop a content model in a single pass. Take an iterative approach. Create a content model for your content authors to test and make improvements based on what you learned.



Authoring in the Contentful Web App



WHAT'S IN IT FOR CONTENT AUTHORS?

Contentful is not just for developers. It's great for content authors as well. Contentful empowers content authors to publish copy wherever and whenever needed and to update content quickly and independently. There's no need for content authors to bug developers every time they need to update content or make a change.

Note that Contentful is not a "one-size-fits-all" platform. It's built to be extraordinarily customizable to each organization and even different use cases in the same organization. That means that content authors will need to work with you to define workflows, enable content previews, and take advantage of other features to optimize their content authoring experience.

Here are some reasons Contentful is great for content authors:

- Reusable content - change content in one place and it changes everywhere
- Customizable workflow - easily streamline processes and organize roles and permissions
- Independence - add or change content instantly without the assistance of technical teams
- Version control - see earlier versions of content and roll back instantly

- Flexible localization - easily manage country-specific or audience-specific text and images

THE CONTENTFUL WEB APP

The [Contentful web app](#) is the main workspace for content authors. Content authors can log in to the web app via email address and password, third-party login providers, or your company's single sign-on (SSO). This [video](#) provides a tour of the Contentful web app.

The Contentful web app is organized into the following tabs:

- Organization - This shows users the organizations they belong to. Typically users will only belong to one organization, but they might belong to multiple spaces
- Space Home - This is a launching tab that has some exercises, documentation, etc.
- Content - This is where content authors will create, edit, and update entries
- Media - This is where content authors will upload media files
- User - This is where users can update their profile information or contact Contentful for support

THE AUTHORIZING EXPERIENCE

When content authors log into the web app, they will mainly be working under the "Content" tab, where they will be creating entries based on the various content types that were modeled by their development team. This [video](#) shows how authors create and publish a basic entry.

In addition to creating entries, content authors will also be uploading media files (images, videos, .pdfs, etc.) to create assets. We have built-in tools that allow for basic image editing, such as scaling and cropping. This [video](#) shows how content authors create and publish assets.

Finally, we know that content authors are not just creating content. On small teams, content authors are often also responsible for content operations, which are the processes, workflows, and technologies that are required to deliver content.

Contentful offers content authors several features to streamline content operations:

- Scheduled publishing - allows content authors to schedule entries to be published or unpublished sometime in the future. This [video](#) goes over scheduled publishing in detail.
- Tasks - allows content authors to add, assign and edit tasks in the web app sidebar. This is an enterprise-only feature at the moment.
- Comments - allows content authors to communicate directly in the web app about the status of a project, like requesting a proofread or asking about image selections.

Content Type "Hotel Landing Page"

Paradise Beach Ocean Resort

Title: Paradise Beach Ocean Resort (27 characters, Requires less than 256 characters)

Slug (required): twd-paradise-beach (18 characters, Requires less than 256 characters)

SEO Metadata: Paradise Beach Ocean Resort (PUBLISHED)

Sections:

- Header: TWD Hotels // Header (PUBLISHED)
- Hero Image: Paradise Beach Hero Image (PUBLISHED)
- Paragraph with Headline: Introduction Experience your bliss at Paradise Beach in Cancun!!! (PUBLISHED)
- Set of Three: Hotel Overview / Set of Three (PUBLISHED)
- Call to Action: Book Now! Ready for relaxation? (PUBLISHED)
- Headline: Rooms and Suites Headline Rooms and Suites (PUBLISHED)

Content Type "Set of Three"

Hotel Overview / Set of Three

Name: Hotel Overview / Set of Three (29 characters, Requires less than 256 characters)

Items:

- Image with Caption: Outside2 (PUBLISHED)
- Image with Caption: Pool (PUBLISHED)
- Image with Caption: Lobby (PUBLISHED)

+ Create new entry and link | Link existing entries

Content Type "Image with Caption"

Outside2

Name: Outside2 (8 characters, Requires less than 256 characters)

Image: (PUBLISHED)

Caption: (0 characters, Requires less than 256 characters)

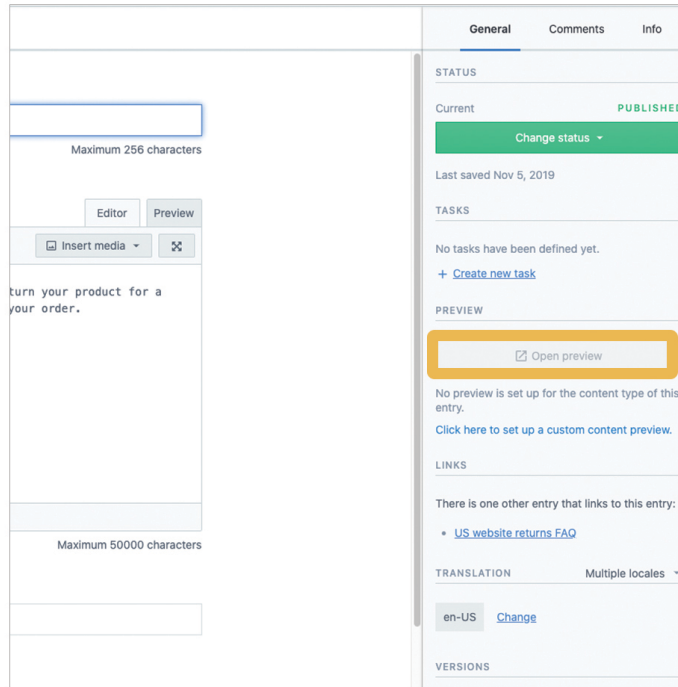
NESTED ENTRIES

As we've discussed before, structured content depends on creating relationships between different content types. This is achieved via references (links) between content types.

As a result of these references in your content model, your content authors will be spending a lot of time creating nested entries. The image above is an example of what this might look like from the point of view of a content author editing entries in the Contentful web app. The entry for content type "Image with Caption" is nested within an entry for content type "Set of Three," which is nested within an entry for content type "Hotel Landing Page."

As you can imagine, too many nested entries can become burdensome for your content authors to navigate. As a result, we don't recommend setting references in your content model more than five levels deep.

To help content authors navigate these nested entries, Contentful offers a handy feature called the slide-in editor. The slide-in editor stacks entries on top of one another so content authors can keep track of where they are. This [video](#) shows the slide-in-editor in action.



Content authors can use this button to preview draft content if preview environments have been configured

PREVIEWS FOR CONTENT AUTHORS

As developers, you can set up one or more content preview links in a space. This generates links in the web app's entry editor, allowing draft and changed content to be previewed in a live environment before it is published and made public.

This is made possible by Contentful's Preview API, which maintains the same behavior and parameters as the Content Delivery API, but for the latest drafts of entries and assets instead of published content. You can set up content preview links by following this [guide](#).

This functionality isn't exclusive to previewing content in a web browser – it also allows you to preview your content across mobile apps, large monitors, digital kiosks, etc.

Because the Content Preview API uses an endpoint and authorization token distinct from the Content Delivery API, there is a low risk of accidentally exposing draft content to the public.

Maximum 256 characters

Editor Preview

Insert media

turn your product for a your order.

Maximum 50000 characters

General Comments Info

STATUS

Current PUBLISHED

Change status

Last saved Nov 5, 2019

TASKS

No tasks have been defined yet.

+ Create new task

PREVIEW

Open preview

No preview is set up for the content type of this entry.

Click here to set up a custom content preview.

LINKS

There is one other entry that links to this entry:

- US website returns FAQ

TRANSLATION Multiple locales

en-US Change

VERSIONS

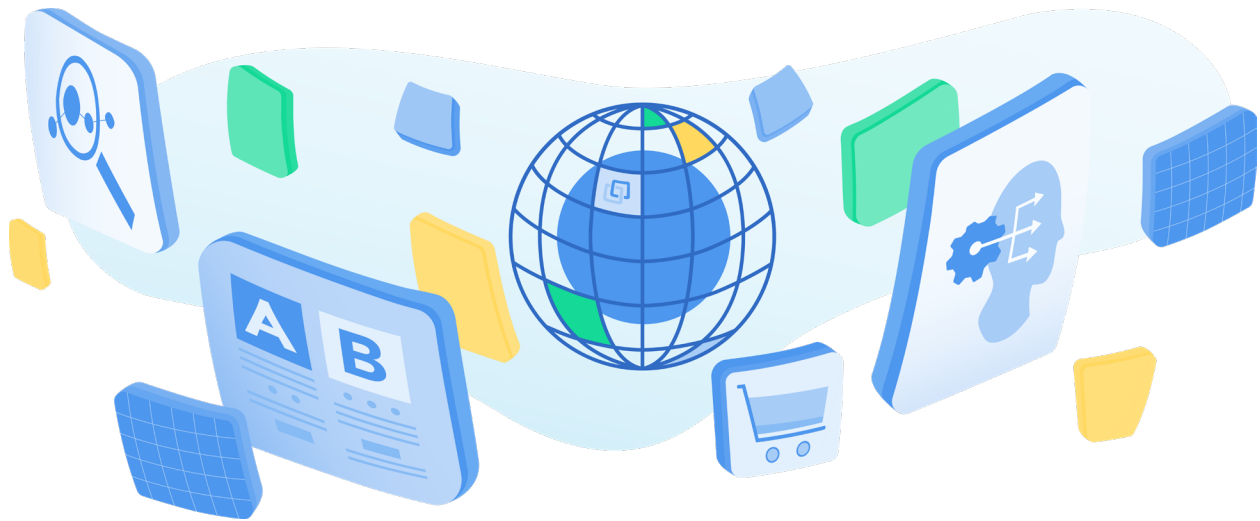
Content authors can localize entries and assets under “translation” in the right-hand sidebar

LOCALIZATION FOR CONTENT AUTHORS

The way your content authors localize content will depend on the way you’ve set up localization for your content model.

The most straightforward way to allow content authors to localize content is through field-level localization, which we covered on [page 35](#).

These two videos demonstrate how content authors can localize [entries](#) and [assets](#) using field localization.



Extensibility



WHY EXTENSIBILITY?

A key requirement for any modern content management platform is it has to be easy to extend. You're going to want to do integrations with other API-based web services. It should be really easy to hook many web-based services with webhooks through simple configuration, without the need for writing custom code. You are also going to want to customize the authoring experience for your content authors.

CONTENTFUL'S EXTENSIBILITY JOURNEY

We started out by creating an SDK for UI extensions and a mechanism to create webhooks. With UI extensions you could customize the Contentful web app with a small amount of JavaScript, HTML and CSS. We then added a design system so these UI extensions would have the same look and feel as the web app. We discovered that we needed to add additional capabilities such as governance to allow org and space admins manage these UI extensions. From all of that we evolved our UI extensions into what we call the App Framework. Webhooks remain a stand-alone extensibility component, but we will be merging them into a future version of the App Framework.

WHAT IS AN APP?

Apps allow developers to extend the basic functionality of the Contentful web app. They can be used to update an existing field, like creating a different interface for editing JSON fields, or building something completely new, like integrating third-party data in Contentful. For example, adding a custom field to Contentful that allows users to search and select Shopify products.

Apps are (essentially) a small HTML5 (including CSS and JavaScript) application that exists in a sandboxed iFrame and interacts with the Contentful web app through the Apps SDK. This SDK is a proxy to the Content Management API and acts on behalf of the logged-on user. The Apps code is completely customizable.

Apps can start out with something as simple as customizing the appearance of a JSON field in a content type, and grow to provide a completely custom editorial interface for content authors.

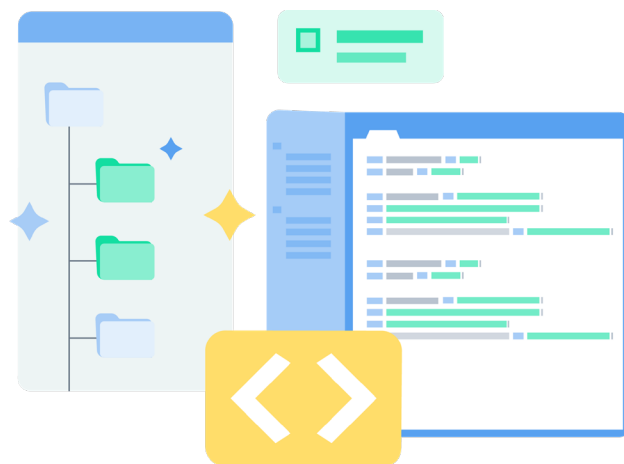
We have built governance into the app model such that you can manage the apps you use, public or private, at the organization and space level.

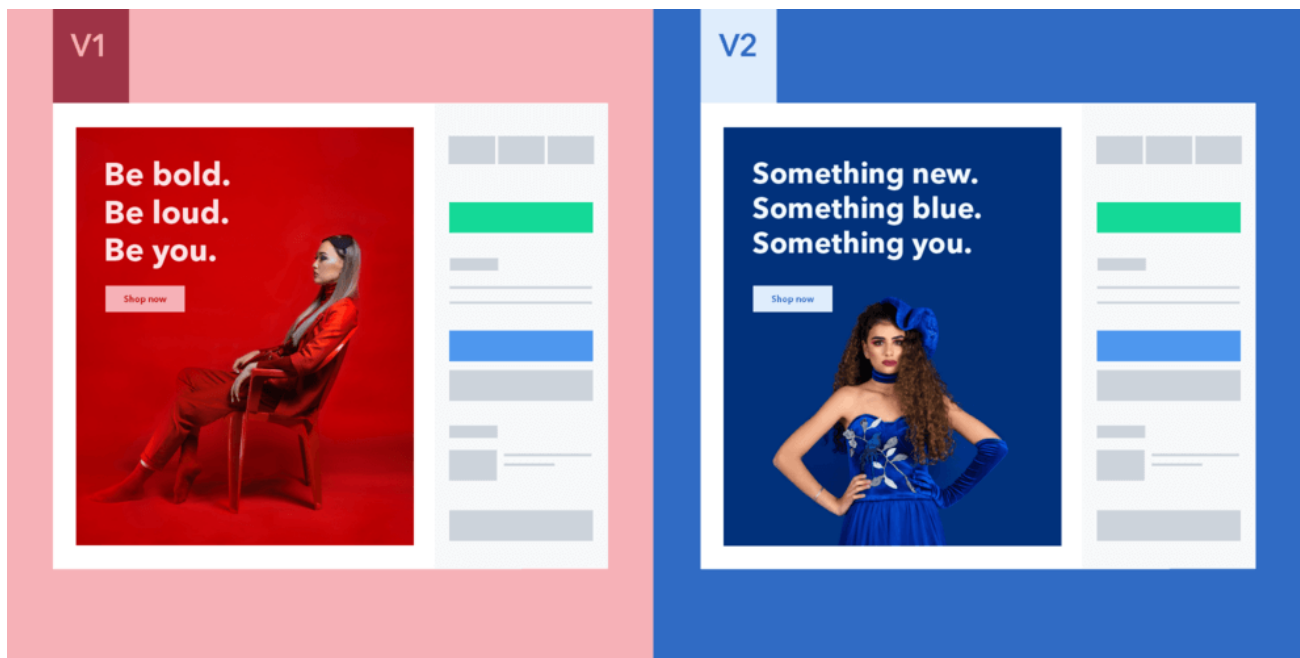
WHAT IS THE APP FRAMEWORK?

There are three key components to Contentful's App Framework:

1. Public apps that are available in our [marketplace](#), all open-sourced with code available on Github.
2. Private apps that you can build in-house. Many of our enterprise customers develop private apps. Some apps are not more than 100 lines of JavaScript and can be built in a few hours, but generate a large UI by improving the user experience for their editorial team. Digital agencies can also build private apps and deploy them to their customers. So when looking at digital agencies you want to work with you should ask them what they've built to date.
3. A framework with an SDK and design system to make apps really easy to build and maintain the same UI as the core Contentful web app.

If you have the need for an app that doesn't exist yet - let's say you want to integrate with a DAM other than Cloudinary or Bynder for example, you can use many of the existing public apps as blueprints to [build your own app](#).





APPS VS UI EXTENSIONS

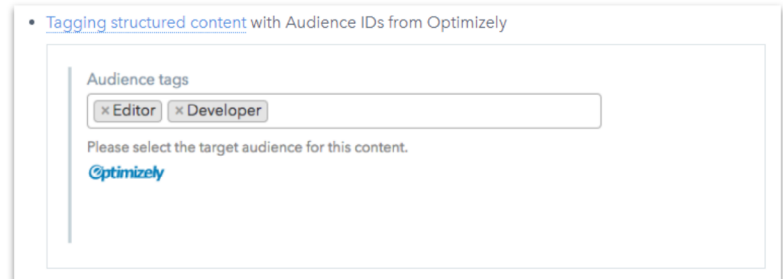
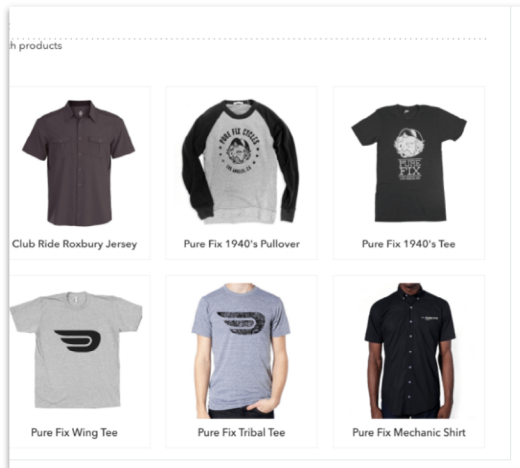
Apps come with capabilities such as an installation screen, configuration and state management. They can be shared across your whole Contentful organization and across spaces, which simplifies maintenance and upgrades.

While UI extensions are building blocks for customizing a single part of the Contentful web app, their application and ability to replicate integration across multiple spaces is limited. Taking this into consideration, we highly recommend customers build and use apps going forward, as they will scale even as you integrate more services with Contentful. In addition, we will continue to release new features to apps.

APPS USE CASES

Here are three common use cases for apps:

1. Experimentation: Apps that allow marketers to run A/B experiments and obtain better insights (e.g. [Optimizely](#) as seen in the image above)
2. Digital Asset Management: Apps that allow editors to access, select or edit media (e.g. [Cloudinary](#))
3. Translation: Apps that enable you to automate, manage, and professionally translate content (e.g. [Smartling](#))



APPS LOCATIONS

As a developer, we want you to easily be able to provide rich experiences for your users. Apps Locations enhance the user experience by adding functionality to critical touch-points of the Contentful web wpp.

Apps currently support six locations:

1. App Configuration
2. Page
3. Dialog
4. Entry Editor
5. Entry Field
6. Entry Sidebar

The image above is an example of how you can use Apps to add extra functionality to entry fields. The screenshot on the left shows a custom UI field that allows users to search and select Shopify products. The screenshot on the right shows an app customization that allows users to tag structured content in Contentful with audience IDs loaded from a project in Optimizely.



Webhooks



WHAT IS A WEBHOOK?

Webhooks are HTTP callbacks which can be used to send notifications when data in Contentful is changed. Contentful has 16 [pre-built webhooks](#).

Common examples of integrations of Contentful with third party systems are:

- Running a test suite on a CI service when the content model of a development environment was changed
- Deploying a production build when content was published or unpublished
- Deploying a preview build when the draft state of content has changed
- Re-building a search index when content was updated
- Feeding assets into an image detection service when they are uploaded
- Triggering email or chat notifications due to editing activity

The screenshot shows the 'Webhooks (5)' page in the Contentful web app. The top navigation bar includes 'Space home', 'Content model', 'Content', 'Media', 'Apps', and 'Settings'. The sidebar on the right lists 'ENVIRONMENT SETTINGS' (Locales, Extensions) and 'SPACE SETTINGS' (General settings, Users, Teams, Roles & permissions, Environments, API keys, Webhooks, Content preview, Usage). The 'Webhooks' item is highlighted.

Webhook name	URL	% of successful calls
jbodenhausen_dev events	POST http://webhook.site/2e5d10d7-a747-4bf8-9e5b-fb0e5bd9ff54	No data collected yet
Smartling-Asset-Updated	POST https://api.smartling.com/contentful-connector-api/v2/webhook/media	No data collected yet
Smartling-Entry-Updated	POST https://api.smartling.com/contentful-connector-api/v2/webhook/entry	No data collected yet
Logging	POST http://logs-01.loggly.com/inputs/72bec389-762d-49cf-b12f-4ffce70b5026/tag/http/	No data collected yet
Approval workflow	POST https://6ajny5opo2.execute-api.us-east-1.amazonaws.com/production/logging	No data collected yet

CONFIGURING WEBHOOKS

Webhooks are called as the result of an action against assets, entries or content types. Whenever a matching event occurs, Contentful sends a request to the URL defined in the webhook definition.

It's easy to configure the triggers of webhooks inside of the Contentful web app. In the top navigation bar, open *Settings > Webhooks*. Click *Add Webhook*, configure the remote host, and click *Save*.

Although Contentful's pre-built webhooks create default JSON payloads, you can easily modify the payload. In the image above we are changing the default payload of the Slack webhook to generate a message that shows the editorial status of a piece of content that has been modified.

This [video](#) illustrates a custom content moderation workflow using content types, custom roles and permissions, and webhooks. This [followup video](#) shows how the previous content moderation workflow was created in Contentful.



Roles and Permissions



WHAT ARE ROLES AND PERMISSIONS?

Roles and permissions is an important feature that allows you to customize access to content for different groups of users. The ability to assign different roles to users in various spaces provides a great deal of flexibility in how companies can enforce governance of their content creation process.

Let's say that a large, multi-national company has one space for global content and multiple spaces for regional content. With Contentful you could assign editors in the global content creation team the ability to both create and publish content in the global space, but then limit their ability to only draft content in the regional spaces.

Nikoo's Organization
Tld2
master

Space home

Content model

Content

Media

Apps

Settings

Roles (5)

Your space is using 4 out of 7 available

ENVIRONMENT SETTINGS

Locales

Extensions

SPACE SETTINGS

General settings

Users

Teams

Roles & permissions

Environments

API keys

Webhooks

Content preview

Usage

Role	Description
Administrator (included in space)	Members of this role have full access to everything in this space. This role is automatically part of your space and does not count against your role limit.
Author	Allows editing of content
Editor	Allows editing, publishing and archiving of content
Freelancer	Allows only editing of content they created themselves
Translator	Allows editing of localized fields in the specified language


CONFIGURING ROLES AND PERMISSIONS

Contentful comes with four out-of-the-box roles:

1. Admin - Can do everything, including work with entries, create and update content types, configure space settings and work with API keys.
2. Editor - Can work with entries, has read-only permission to content types and space settings, does not have access to API keys.
3. Author - Can do everything an editor can do except for deleting, (un)publishing, (un)archiving entries.
4. Translator - Can only work in the assigned language, can't create, delete, (un)publish, (un)archive any entries.

Developer and Freelance roles have been deprecated in the current product version and are only available on select legacy plans.

Admins assign roles and permissions to users at the space level and users can have different roles in different spaces. To configure roles and permissions in the Contentful web app, go to the master environment for your space and from the menu bar choose *Settings > Roles and Permissions*.

 Editor*

Role details

Name (required)

Editor

Description

Allows editing, publishing and archiving of content

Content

Users with this role can:

All actions

Any entry

All content types

× Delete rule

[Add another rule](#)

Users with this role can **not**:

Edit

Any entry

Navigation Menu

All fields

All locales

× Delete rule

[Add another exception](#)

CUSTOM ROLES AND PERMISSIONS

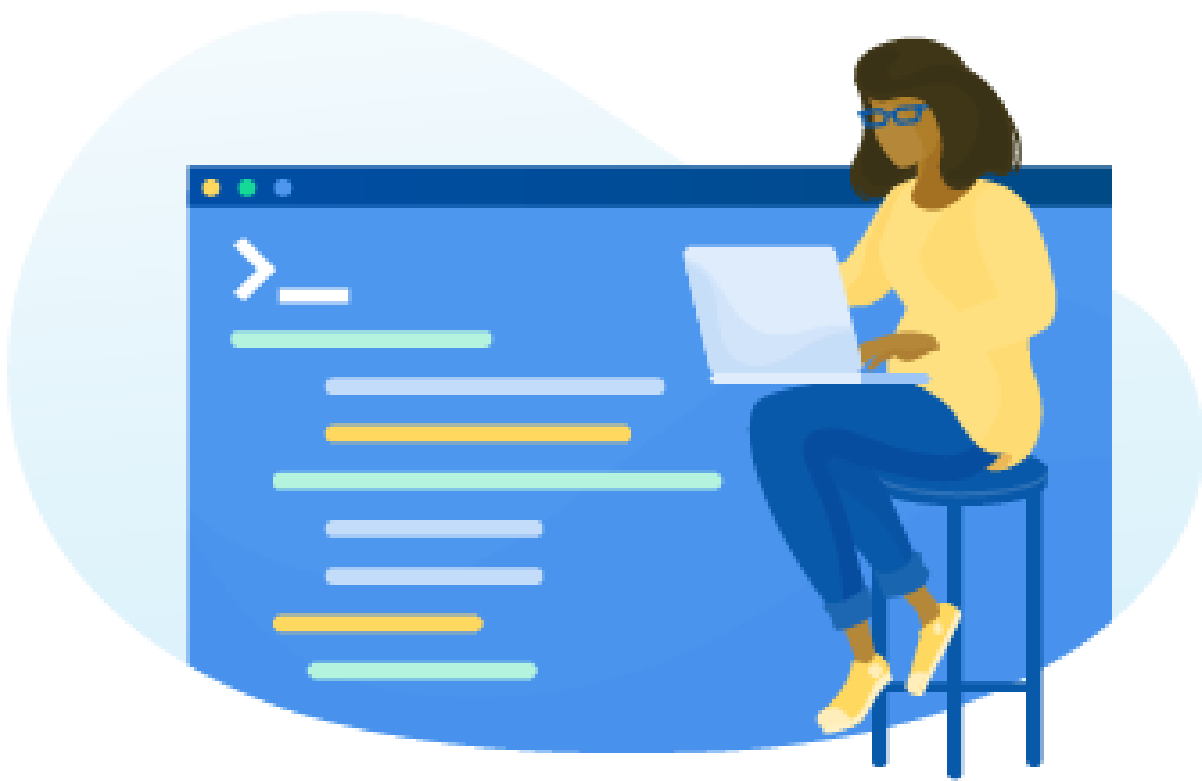
Depending on your subscription level for Contentful, you might be creating custom roles and permissions. In this case, it's important to remember the following high-level principle: anything that is not explicitly allowed is denied. For example, let's say a role can edit entries but not read them. This will result in a user who won't be able to open any entries at all.

You can give a role a permission but then add specific exceptions. The image above shows how the editor role can perform any kind of operation on an entry except for editing any entries belonging to the content type "Navigation Menu."

TEAMS

Contentful also offers some enterprise-level customers the option of teams.

Space Admins can use the teams feature to organize and streamline the administration of groups of users and roles/permissions. With teams, there is no need to add users individually to different spaces. Add the user to the team and then give the entire team access to different spaces with just a few clicks. Every user within the team will inherit the role (and corresponding permissions assigned to that role) that the team has been granted for each assigned space.



Managing Content at Scale



WHY CONTENT AS CODE?

Content models are always evolving, creating new content types, adding additional fields to existing content types, and refactoring models to meet new requirements. Developers can efficiently work with code at scale - using multiple Git branches, webhooks to automate CI/CD workflows triggered by actions such as committing a new branch, and tools to make large scale refactoring changes. Contentful allows you to work with content at scale, treating content as code.

Developers who are evolving an application's content model can use Contentful's CLI tools built on top of the Content Management REST API. Developers spin up one or more development environments, similar to how they create feature branches in Git, to work in an isolated environment from content authors.

Content authors can use the web app to publish new content in draft mode. Draft content can then be put through an approval process by having their colleagues preview their changes using the Contentful Content Preview API, and use the web app to publish their changes.

Once content is published, websites and applications can consume the content using Contentful's REST APIs and SDKs.

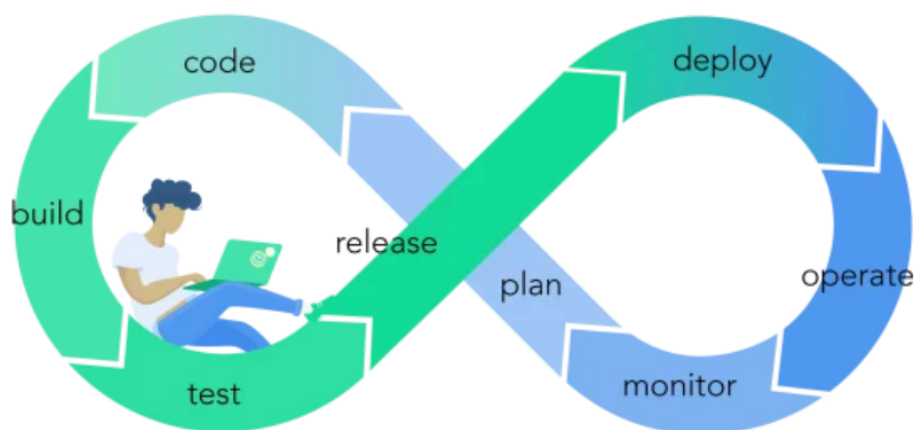
Developers can prototype content model changes in the Contentful web app, and then create small code snippets using the Migration SDK to apply those changes programmatically. The combination of the Migration SDK, multiple development environments, and webhooks enables developers to tap into their existing CI/CD workflow tools to make large scale content model changes. After thorough testing, the changes can be safely deployed to the master environment used by the editorial team.

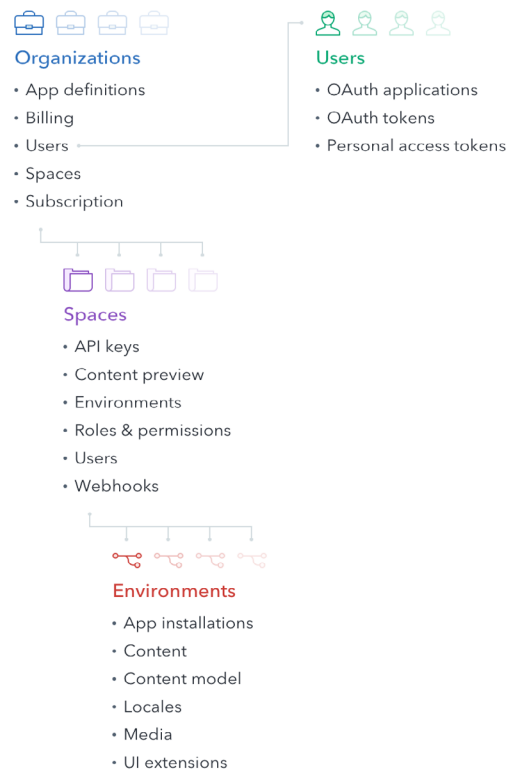
CLI TOOLS

Contentful's [command line interface tools](#) let you use Contentful features straight from your CLI, providing the basis for DevOps automation.

Available CLI commands include:

- Securely login and logout with our OAuth service
- Manage spaces - List, create, ...
- Export your space to a JSON file
- Import your space from a JSON file
- Execute migration scripts written in the Contentful Migration DSL
- Generate migration scripts for the Contentful Migration DSL from existing spaces
- Seed your space with example data
- Create a new UI extension from a template
- Manage installation of extensions in a space
- Run a guide which introduces you to the Contentful basics





CONTENTFUL'S DOMAIN MODEL

The image above illustrates Contentful's domain model:

- Organizations - the top most parent object, which connects users to a company.
- Users - members of an organization who have access to specific spaces based on roles & permissions.
- Spaces - a child entity of an organization. Spaces contain content, assets and webhooks.
- Environments - entities belonging to a space. Spaces contain an isolated copy of content and assets.

Each space always has at least one environment named "master." Depending on their subscription, customers might have additional environments as well. When you create a new environment inside of a space, the content model, content, locales and webhooks will by default be copied from the master environment. However, there might be instances where you will want to choose any other environment to be the basis for your new environment inside of that space.

Content models live inside of environments, enabling development teams to evolve a content model in parallel for testing. They can then apply their content model changes back to the master environment. You can manage org, space settings, users, access tokens and api keys, snapshots with the Contentful web app or the Content Management API.

SPACE MODELING PATTERNS

Space modeling is the concept of utilizing spaces and environments as well as other objects within a space to design the best architecture for your project.

We use spaces to separate content based on different teams or use cases unless we find that they have a lot in common and can share content or content types between them.

Here are three tips to get the most out of your spaces:

1. Use spaces to separate content that has different teams or different use cases unless they need to share content
2. For large scale implementations, consider putting global shared content, team shared content and project-specific content in separate spaces
3. Roles are specific to a space—use separate spaces if the same users need different permissions

As we covered on the previous page, Contentful's spaces are like repositories. They are a child entity of an organization. Spaces contain content, assets, and webhooks.

In order to determine which space pattern is the best fit for your organization, consider the following questions:

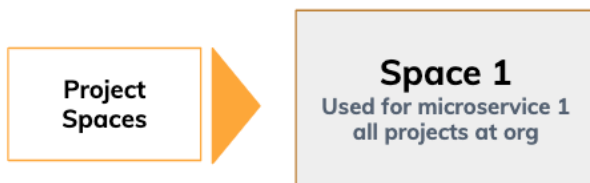
1. Will governance rules (i.e. user roles and access permissions) be the same or different among the teams and projects?
2. Will content types be the same and/or shared across projects?
3. Will projects have the same localization requirements? Same or different set of locales?
4. Is there a legal or regulatory requirement for projects?

ONE SPACE TO RULE THEM ALL

One space to rule them all is a very common space modeling pattern we see in Contentful. Many customers start off with a single space model.

This is a good pattern to use if all of these conditions are true:

- Content types and/or content entries are reusable across projects
- Projects all have the same localization needs
- Roles & Permissions provide sufficient access controls across all projects
- Content is intended to be used for multi-channel delivery
- Multiple teams or business units want to collaborate together across projects



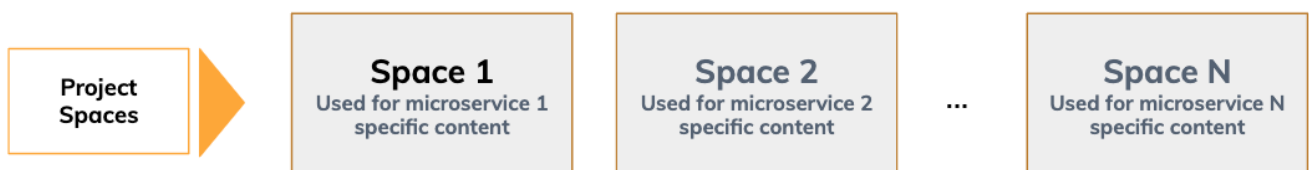
Content consists of Assemblies, Topics, and Media

SEPARATE SPACE PER PROJECT

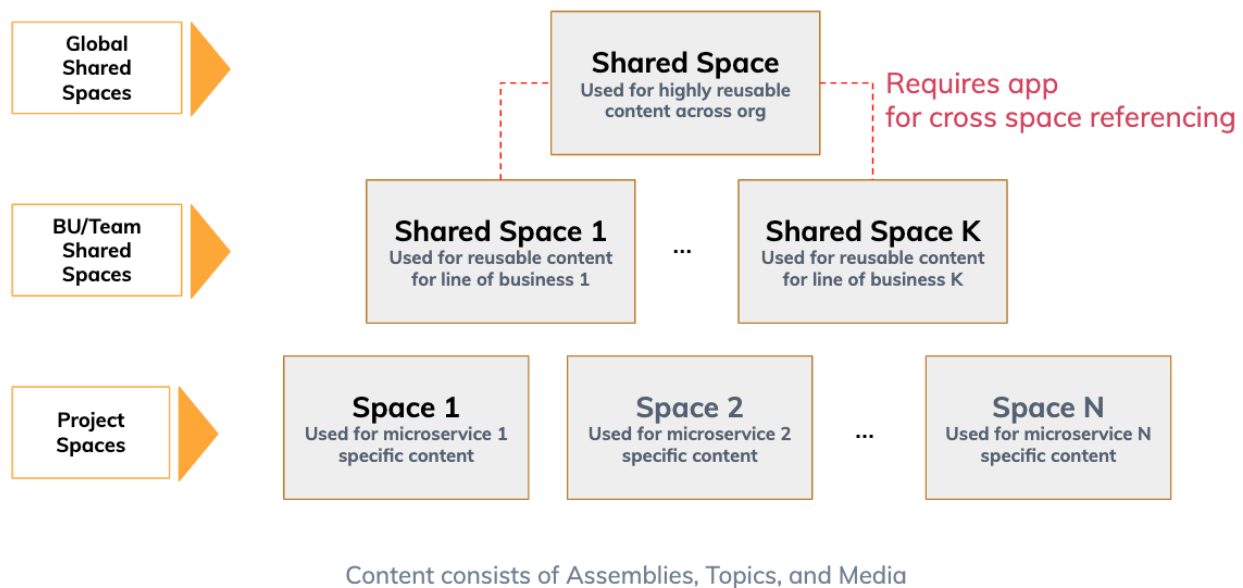
Another space modeling pattern to use is separate spaces per project. This pattern lets multiple development teams work independently of each other. In this model each team/space can provide its own microservice.

This is a good pattern to use if one or more of the following conditions are true:

- Content types are not reusable across projects.
- Projects have different localization needs.
- Roles & permissions cannot be effectively used to create granular access controls.
- Business units have a regulatory or legal requirement to keep their project and content separate.



Content consists of Assemblies, Topics, and Media



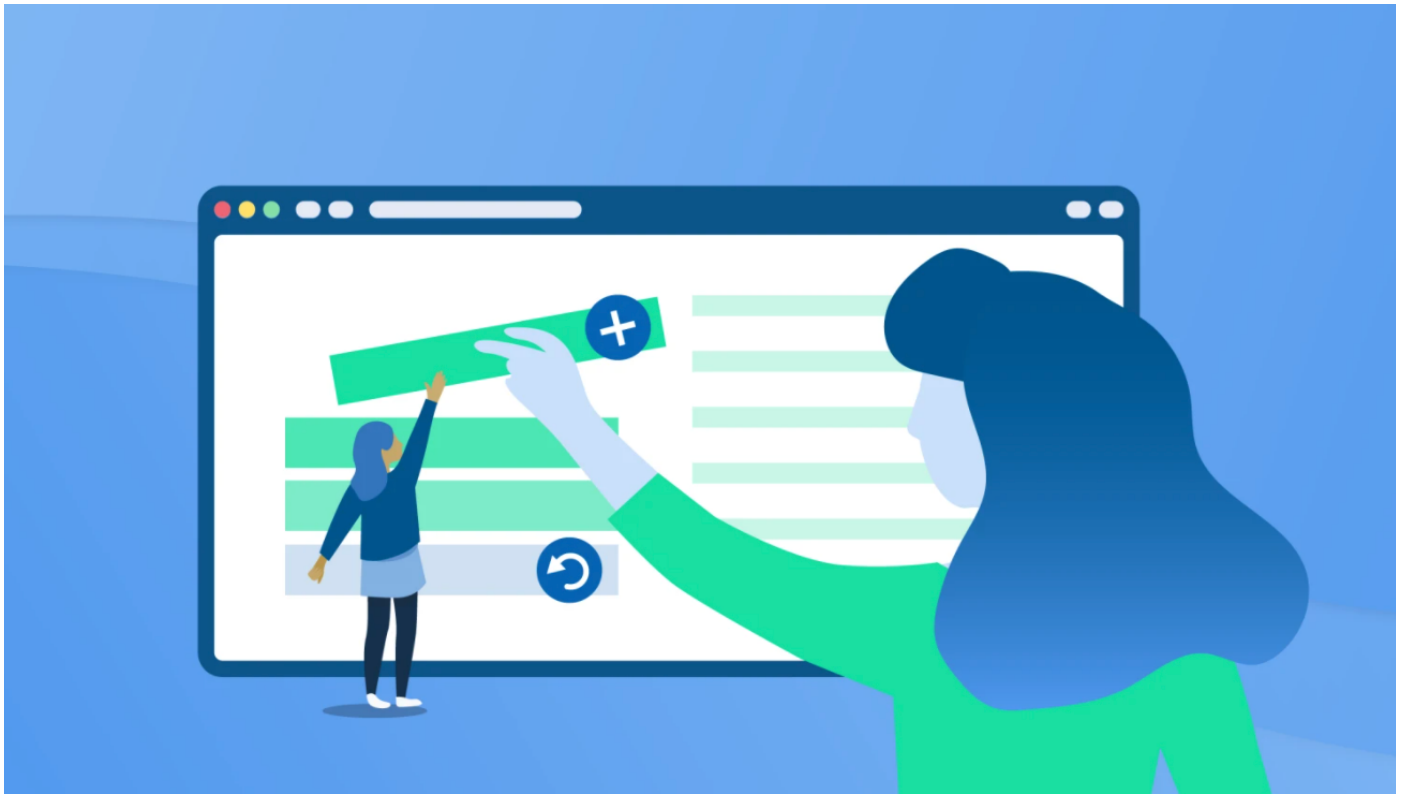
MULTI-TIER SPACES

For large, multi-national, multi-brand, multilingual companies, we often see the need to mix global content with more regional or local content. This is very difficult to do with traditional server-based, legacy CMSes. Using multiple levels of spaces provides a very flexible solution to these complex content requirements.

In this pattern, you create one or more spaces that are designated as a global or shared space. This allows some of the content to be shared across an organization or across specific departments while other content can be project specific.

Any front-end or any application that is used for your project can access this global shared space using an API token and retrieve and show those assets among those projects. You then segregate your content across business units or teams. As a result, each team or business unit might have a space designated for their specific line of business or projects that they don't share with other units or other teams and can manage in an isolated way.

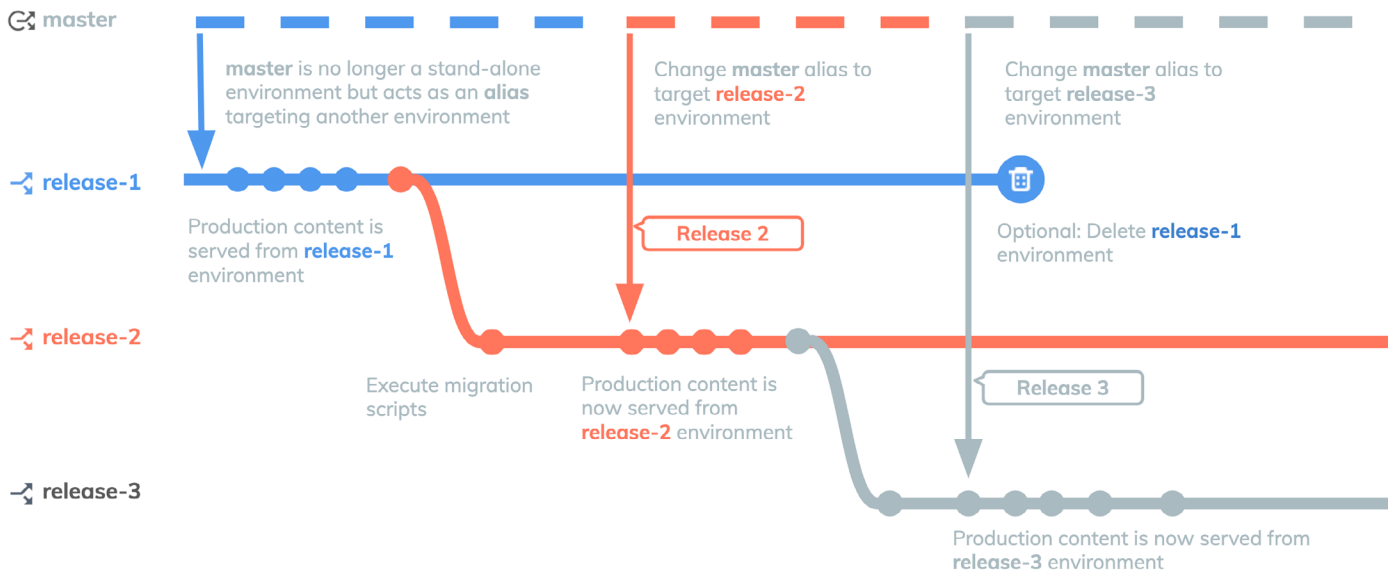
If you want your content authors to view that content across their business and project spaces, your team will need to build a UI extension so you can do cross-space referencing.



USING ENVIRONMENTS FOR AGILE DEVELOPMENT

Now that we've discussed space architecture, you need to think about how to use environments to implement an agile deployment pipeline that adheres to CI/CD best practices.

Environment aliases are our solution for what we called in the past "environment promotion" feature. Environment promotion is the ability for you to "promote" a new environment to be served as your default environment (aka to be served from master or from the unscoped route). It does not include git-like merging/diffing between environments (for now). Think of promotion as "flipping a switch"- now your master/unscoped route serves content from a different environment. We decided to achieve this by turning master into an alias. In the future, we may support user-defined aliases, but for now, master is the only one that will exist.



RISK-FREE RELEASES AND INSTANT ROLLBACKS

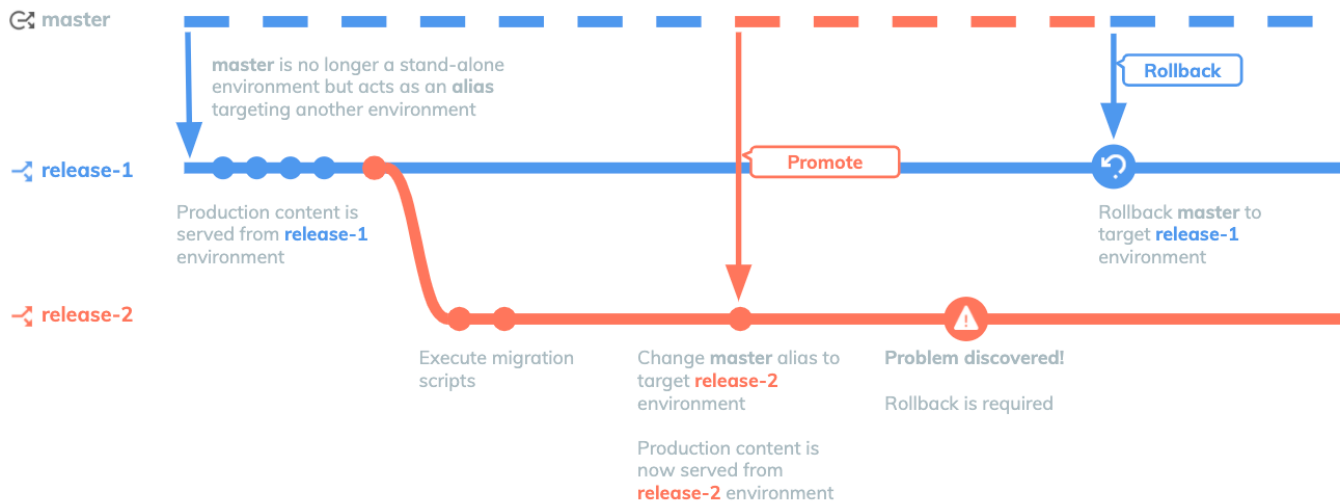
When you activate environment aliases within a Contentful space, “master” is no longer a space. It’s an alias for a different space. In this example master initially is pointing to the release-1 space, so you’re serving production content off of release-1. The development team is working on a new release using an environment named release-2. When the team has completed the final testing of release-2 they switch master from pointing to release-1 to release-2. You do this in the Contentful web app and it happens immediately. You can think of this as a blue/green deployment if you are familiar with that concept.

You no longer need the release-1 environment and can delete it, although you generally want to keep it around for a while in case you need to do an emergency rollback.

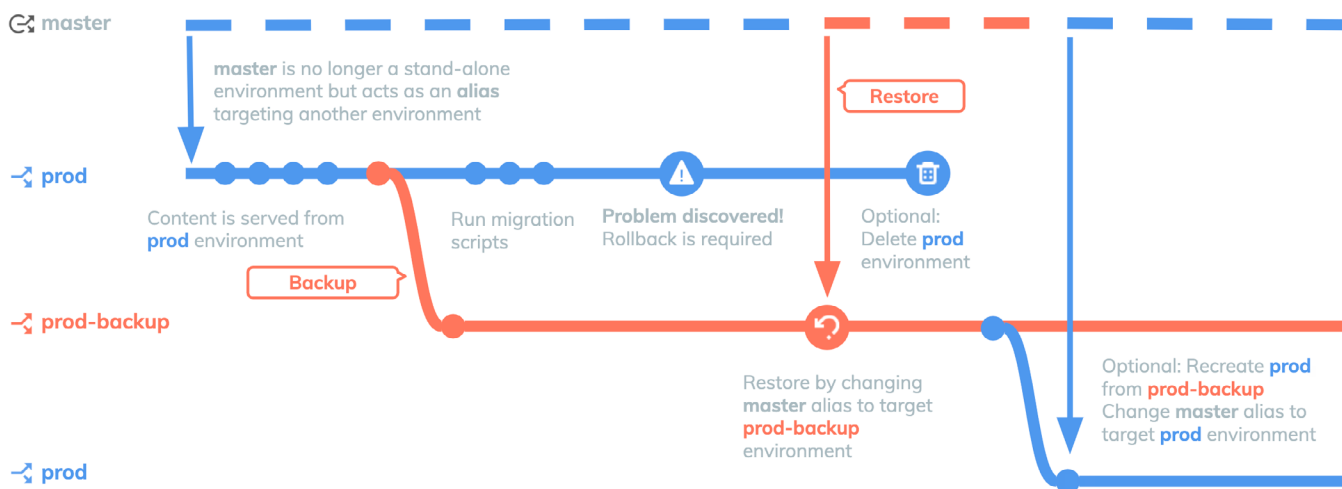
You can now spin up a new environment based upon release-2, and follow the same approach and switch master to it when you’re ready.

USE CASES FOR ENVIRONMENT ALIASES

In the previous example we switched master from release-1 to release-2. What if, despite your best testing efforts, you discover a serious bug after the switch? With environment aliases you can immediately revert to release-1. That's what's great about zero downtime deployments.



With environment aliases it is very easy to create multiple backups of development environments. New environments can be spun up in seconds through either the Contentful web app or the Content Management API. In the event that switching the master alias to a new environment results in problems not found during testing, you can immediately revert master to a previous working backup environment.



```

1  module.exports = function (migration) {
2
3      const blogPost = migration.editContentType('blogPost');
4
5      blogPost.createField('blogAuthor')
6          .name('Blog author')
7          .type('Link')
8          .linkType('Entry')
9          .validations([
10             {
11                 "linkContentType": ['author']
12             }
13         ]);
14
15      blogPost.moveField('blogAuthor').afterField('title');
16

```

MANAGING CONTENT WITH CODE

In this chapter we've discussed how you can manage content at scale by managing content as you manage code. So how do you do that?

There are five basic steps of managing your content as code:

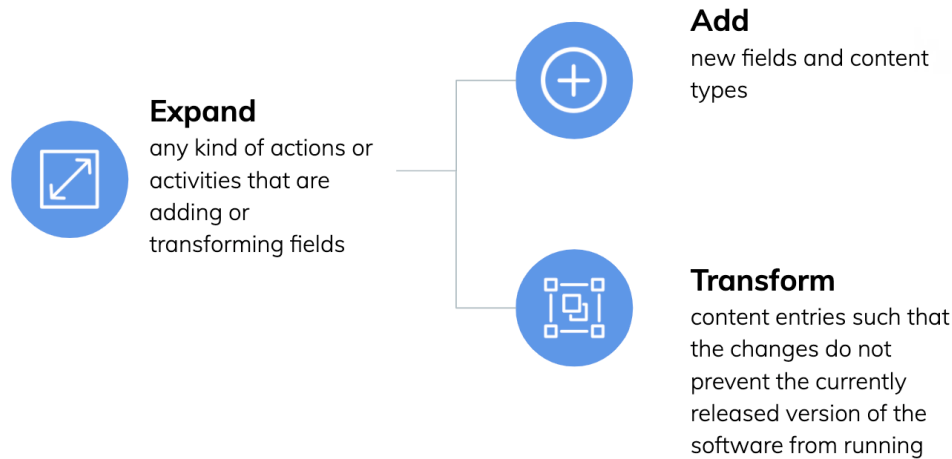
1. Create the environments you need for CI/CD
2. Experiment with a content model in the web app where you can see JSON output
3. Create a script to programmatically build the full content model
4. Create migration scripts to evolve your content model and content
5. After testing, propagate your changes into the production environment by switching the master environment alias

This [video](#) shows how you can manage your content as code.

REFACTORING USING EXPAND/CONTRACT PATTERN

Content migrations can be very complex. For example, adding/deleting fields, editing existing fields, creating new relationships between content types and fields, etc. In this case, we want to have resilient content migrations that mitigate the risk of data loss or corruption. As a best practice, we refactor our code and our content model using the expand/contract pattern.

The “expand” phase refers to any change that adds or transform fields. Anything in this phase is additive and will not affect the current version of your application from working properly.



The “contract” phase activities are destructive, such as deleting fields, deleting content types, or deleting links between content types. If you have an application that has a dependency on these fields, you want to avoid accidentally breaking that application by making sure that application is updated to match the new fields and content types (i.e. the “expand” phase) before you run the migration in the “contract” phase.

