Customizing Contentful

A DEVELOPER'S GUIDE

contentful

Table of Contents

Authors

As a solutions architect with more than 16 years of experience, **Arjun Ganesh** designs cutting-edge solutions that help customers increase their speed and agility when bringing digital experiences to market.

In engineering roles and as a solutions architect, **Edwin Maldonado** prioritizes customers' needs and helps them learn the best practices of content modeling and software developer kits.

Sia Rad, a global IT director and technical architect, has more than 20 years of experience in consulting, designing and delivering secure and distributed enterprise solutions across industries.

Jim Sam is a solution architect who is passionate about helping customers achieve their content goals. For nearly a decade, he has helped companies and nonprofits in their digital transformation by analyzing and designing systems that enable seamless access to content.

As a technical architect, **Daniel Schley** helps customers to succeed in transforming their digital solutions toward a modern content management infrastructure.

Kiersten Thamm is a technical writer and the managing editor of the Contentful blog. She's a Contentful power user and collaborates with engineers to write articles that help users build dream projects with Contentful.

Introduction

The world's largest companies run on Contentful. Companies whose scale, scope and legal obligations often produce complicated challenges.

That's where Contentful's solution architects come in. Architects design solutions for the most complex situations, helping customers built secure and inventive digital experiences under any circumstance.

A successful content model sets everyone up for success.

This book begins with Jim Sam's article "Content modeling process." In it, he helps readers develop content models that cater to specific business goals. His advice benefits all Contentful users that want to think more strategically about the process of content modeling.

The following six articles address advanced topics related to Contentful integrations. Our architects wrote each in response to specific customer requests, but their solutions have wider applications.

UI extensions can enhance Contentful's web app in almost any way expressible in HTML and Javascript — including the integration of external APIs in the web app. In "Media metadata patterns," Jim Sam explains how to add fields to a media asset by creating custom metadata fields with a content-type wrapper and UI extension. Daniel Schley clarifies other complex integrations with Contentful in "Integrations with Contentful."

Edwin Maldonado and Arjun Ganesh provide data backup and ownership solutions. Maldonado demonstrates how to set up automatic backups for Contentful spaces using AWS Lambda Function and CloudWatch in "Configuring automatic backups." In "Data ownership," Ganesh takes data a step further and explains how an enterprise can use AWS S3 to back up data on personal infrastructure and maintain an audit log.

Sia Rad and Ganesh address seamless experience and personalization in user experience. In "Blue-Green deployment," Rad explains how to use Contentful spaces to publish updates without interrupting a customer's experience. Ganesh shows how to create a moderately personalized web application using Contentful, Optimizely and Google Analytics.

Some of these solutions depend on features only available to enterprise-level customers. If you aren't already working with Contentful, you can request a free enterprise trial.

We hope this collection helps you use Contentful to build your most ambitious project.

Content modeling process

Jim Sam

How you structure your content determines what business outcomes the content produces. Having project owners, content strategists, developers, and editors understand this is crucial for the project's success.

The content model is at the heart of a team's Contentful experience. Why? It's the technical way you structure content that determines business outcomes. Well-structured, strategic content enables digital teams to quickly ship their products across multiple channels.

How do you go about creating a successful content model? What does that process look like? We've hinted at this before with content modeling basics, composable entries, and topics and assemblies, but let's discuss it on a more thorough level.

Prerequisite concepts

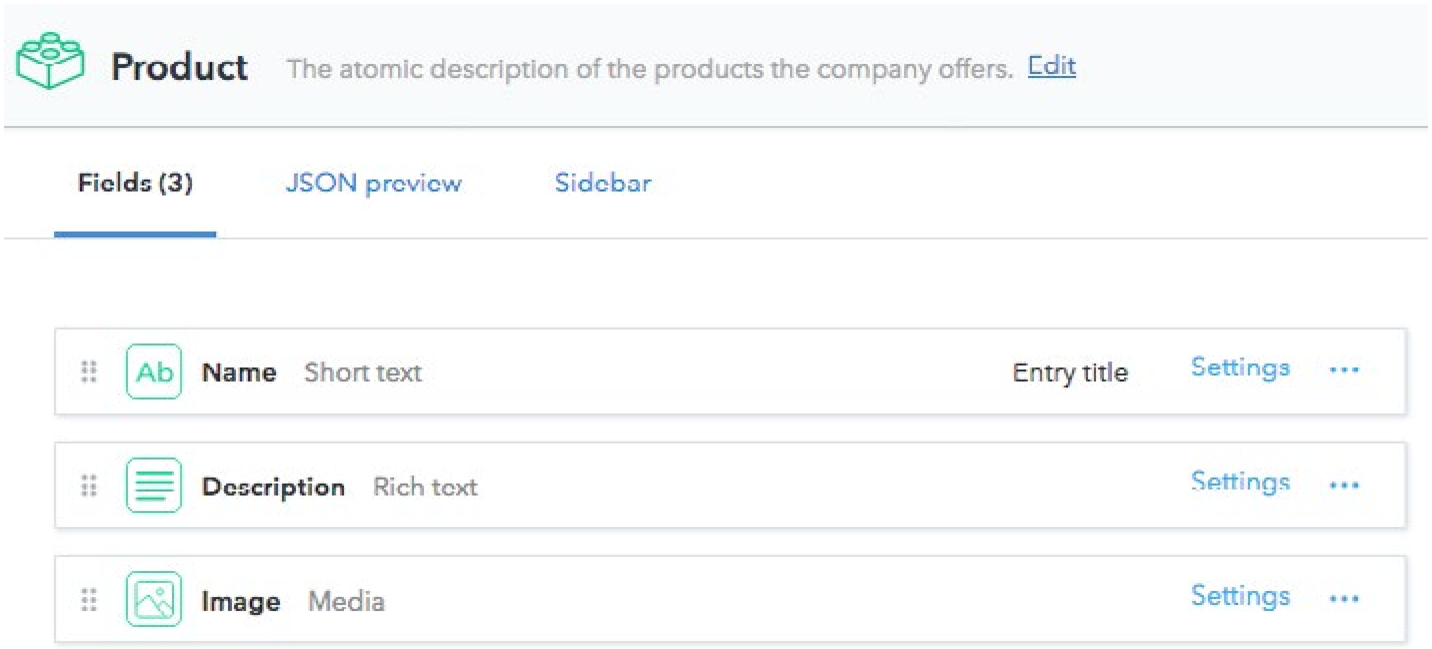
This article builds on the information in content modeling basics, composable entries and topics and assemblies. Familiarity with creating databases or metadata schemas is helpful but not required. Familiarity with JavaScript is also helpful for some of the associated tooling.

Major themes

TOPICS AND ASSEMBLIES

Content types and their entries can describe content (topics) or context (assemblies), which hard content flows into. Assemblies can aggregate topics or other assemblies. Any individual assembly can be medium-specific (web, mobile, etc.) or work across all media. The fields that go in a particular assembly have profound implications on content re-use, content security, personalization, workflows, and automation strategy.

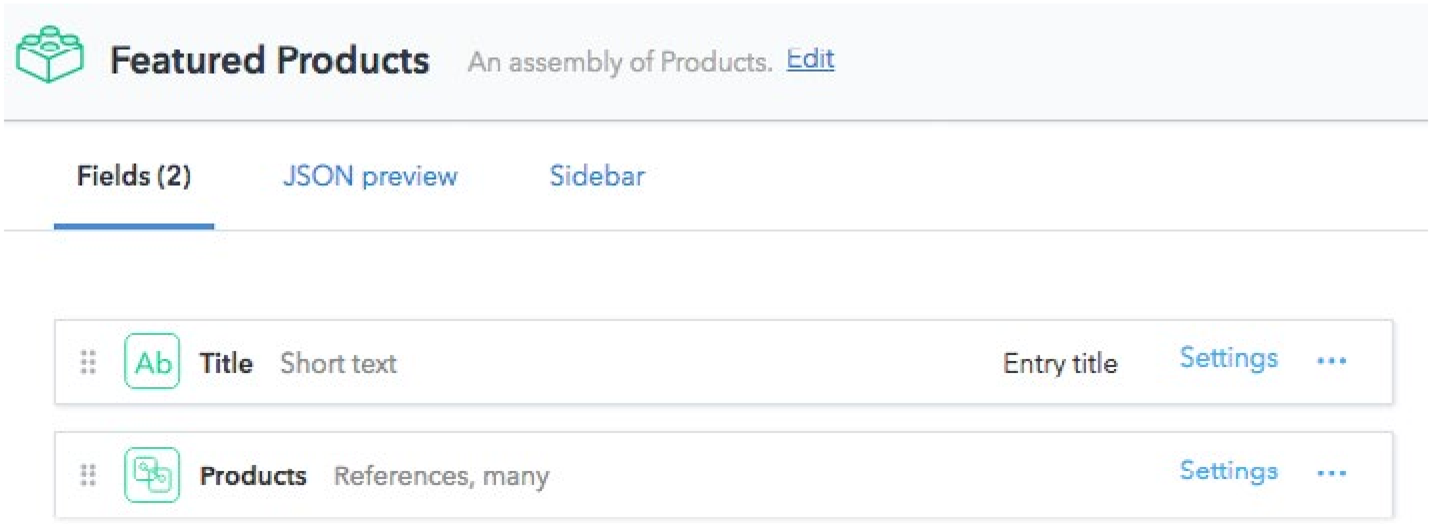
A content type as a blueprint for content (a topic):


Product The atomic description of the products the company offers. Edit

Fields (3) | JSON preview | Sidebar

Name	Short text	Entry title	Settings
Description	Rich text		Settings
Image	Media		Settings

A content type as a blueprint for context, a grouping of featured products (an assembly). Implicit in this assembly is that it will have a parent assembly:


Featured Products An assembly of Products. Edit

Fields (2) | JSON preview | Sidebar

Title	Short text	Entry title	Settings
Products	References, many		Settings

Note that this assembly does not imply a specific medium and could be used for any.

FUNCTIONALITY CHOICES

Your content model depends on what functionality you want to optimize for. Like with all software architecture decisions, there are trade-offs. You need to have in mind what quality attributes you are optimizing for: content reusability, authoring/editing usability, content security, automation, modifiability, app speed, etc. that can be in conflict with each other. There is no singular, definitively-best content model.

WHO SHOULD DEFINE A CONTENT MODEL?

We strongly suggest that a cross-functional team is involved early in the process. A successful content model drives successful business outcomes, and the best content model reflects the needs and expectations of everyone involved.

What may seem obvious to a developer may not be so evident to a content editor, and what an editor wants might be opposed to the needs of content strategy. For example, WYSIWYG editors in other content management systems do not allow you to enforce a content strategy between content creators, which leads to unique pieces of content that do not conform to a larger strategy.

A team can iterate on its content model more quickly when diverse roles discuss their challenges, and everyone maintains appropriate expectations since they understand the project as a whole.

Optimizing for specific outcomes

The mechanics of defining what fields belong in what content types gets to the heart of the functionality question.

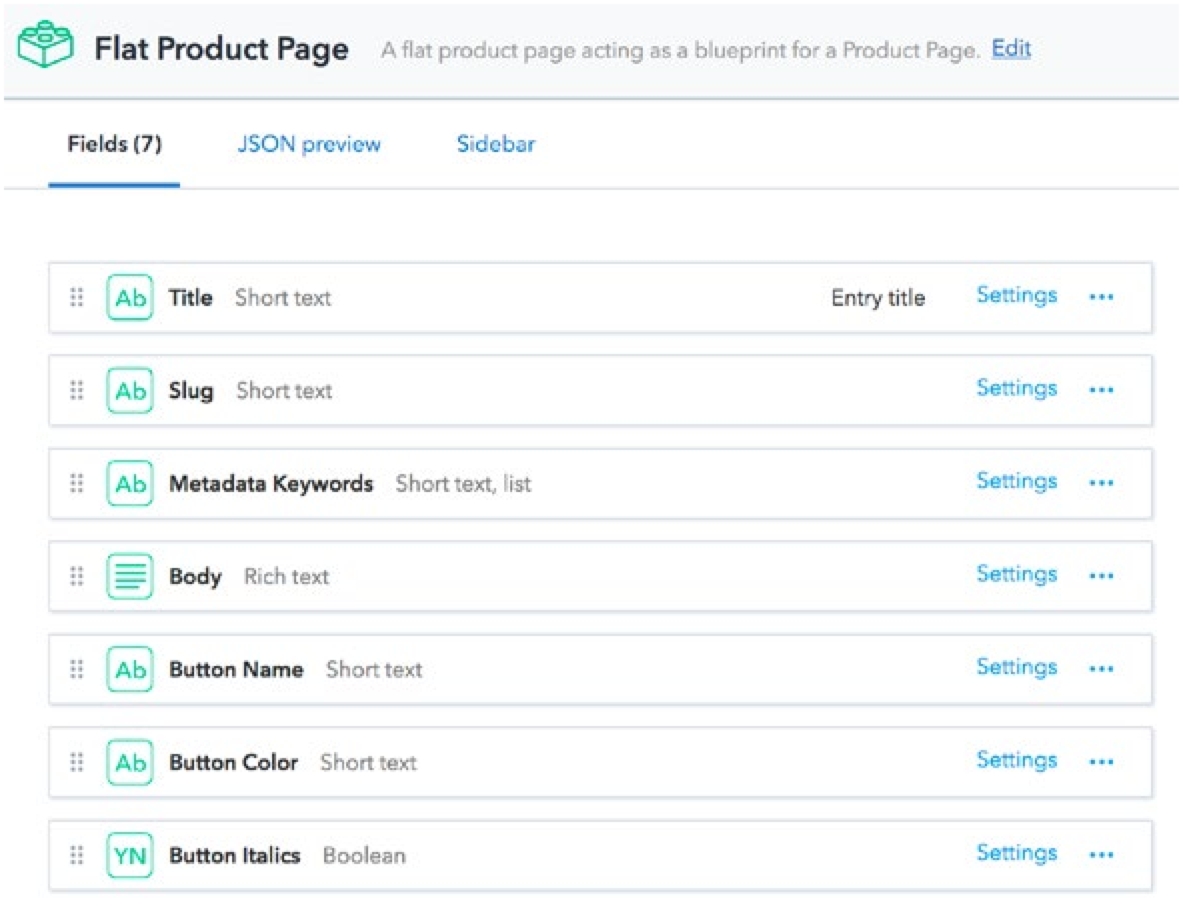
In general, each content type should be a conceptual thing unto itself (i.e. product, row of cards, button, etc.). But it's a key question every time, if an attribute or group of attributes should be a field on that content type or abstracted to a referenced content type. Different optimizations will show different, often unforeseen problems.

To better illustrate this, let's walk through a few common scenarios and outline the pros and cons. This is by no means an exhaustive list of trade-offs or content modeling tactics that you might face but a greatest-hits collection. Where your team falls in each is dependent on your objectives.

Authoring usability vs. content reusability

One of the hardest choices with content modeling is the question of how flat a model should be. Flat means here that a content type does not have many reference fields. Flat content types are easy to understand for an editor, but they do not lend themselves to content reuse. The same content is often manually entered again and again across flat entries. When this content requires an update to change a value across all entries, someone must either write a script to programmatically change the values or change each manually. Abstracting those values into a referenced entry offers an elegant solution.

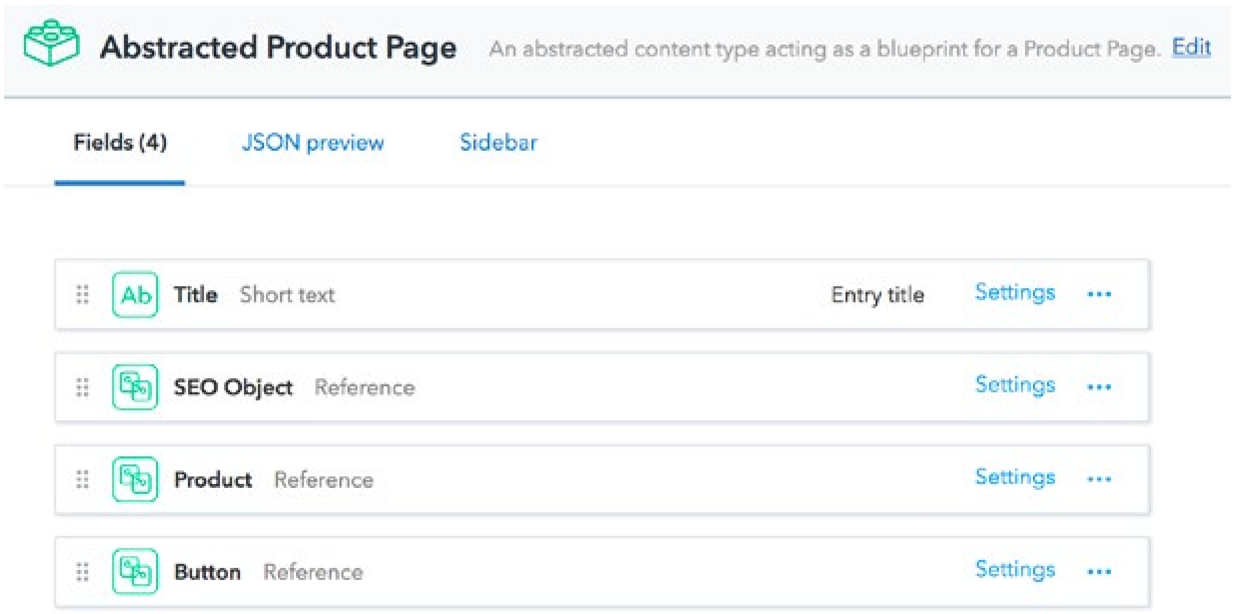
This is a flat content type for a product web page. It's easy to understand what fields will compose a page, but you might find many of the same button designs used repetitively across entries.



Flat Product Page — A flat product page acting as a blueprint for a Product Page. Edit

Title	Short text	Entry title	Settings	•••
Slug	Short text		Settings	•••
Metadata Keywords	Short text, list		Settings	•••
Body	Rich text		Settings	•••
Button Name	Short text		Settings	•••
Button Color	Short text		Settings	•••
Button Italics	Boolean		Settings	•••

This is an example of an abstracted content type for a product web page.



Abstracted Product Page — An abstracted content type acting as a blueprint for a Product Page. Edit

Title	Short text	Entry title	Settings	•••
SEO Object	Reference		Settings	•••
Product	Reference		Settings	•••
Button	Reference		Settings	•••

Understandability vs. omnichannel reusability

Content models are frequently geared towards one medium, which can cause omnichannel problems down the road. For example, we often see content models that mimic a web site's layout with landing page, category page, product page, etc. as content types. It's easy to understand for all stakeholders.

Users can onboard quickly, and it maps nicely to the front-end code for a web site. It suits many customers well, but it has a downside: poor content reusability. The mixing of hard content (a product) with context (a page describing the product) means the wrong content is pulled from the API in a different medium. If you want to reuse the content in a mobile app or digital signage when displaying product info using our REST APIs, you would likely pull an excessive amount of content from a product page entry that doesn't make sense for a non-web medium. Using the GraphQL API could be a solution, but this may create unwanted cognitive overhead when maintaining your app over time.

Extensibility automation vs. authoring usability

Abstracting fields into their own content type offers some benefits to developers while challenging content authors and editors. Webhook automation exemplifies this dynamic. As a content model becomes more structured through assemblies, it is possible to finely tune when a webhook fires to another service. This is a similar strategy to developers including extra divs on a web page for styling reasons. However, like the question of omnichannel reusability, this introduces added complexity for a content creator or editor.

Automation vs. space limits and maintainability

From time-to-time, customers map content types to React components in their design system. On a technical level, this seems like a good idea. Having a one-to-one mapping between Contentful and the front-end makes sense. However, we generally do not endorse this pattern for two reasons. First, it often results in a high content type count, which pushes against the limits of purchase limits. Second, components and content types will change as the app goes through its life cycle. Keeping them in sync may become difficult to maintain, especially if the app is only partially redesigned. Instead, we suggest using higher order components or selectors in redux to do this mapping.

Malleable content models

With Contentful, your content model is not permanent. It may need to change over time due to agile iteration, honest mistakes or changing business needs. A content type as an abstracted blueprint for a product web page. Our easy-to-learn migration tooling allows you to refine or overhaul it as needed.

Content models require modification for any number of reasons. Maybe you need to move some fields into their own content type so that content can be reused and data entry time diminished, such as in this deep dive video.

Combining our content migration DSL and tooling with space environments allows your development team to prepare migration scripts for the next version of your content model, independent of editors, and programmatically apply these changes — CMS as code.

As you adjust your content model, we suggest using an expand/contract process. In this, one migration script will create the necessary new fields, references and new content types. A second script will adjust the corresponding entries so that the new and old schemas temporarily exist together, with old fields disabled for editing and the new fields possibly omitted from the API response. Once all applicable parties have approved the new version of the content model, a final script runs to clean out the old content.

Conclusion

Contentful allows you to define your content model and equips you to ensure your content model promotes intended outcomes, even as your digital products evolve. However, a content model is just another software architecture choice, and there are tradeoffs between functionalities. Keeping this in mind will help you develop your optimal content model.

Media metadata pattern

Jim Sam

How custom metadata fields on an asset can be achieved with a content type wrapper and UI extension.

Customers often want more fields on a media asset beyond the default title and description. Handling this is a three-step process. This can be as simple as adding an explicit alt text field or supporting standards like IPTC photo metadata or Dublin Core.

How it's done

STEP 1: CREATE A WRAPPER CONTENT TYPE

Create a new content type named image (or video, etc.). Because a content type is custom-defined by default, you now define the fields you'd like. Included in this content type is a media field, likely with the media field validation set to the type of file required by this type.

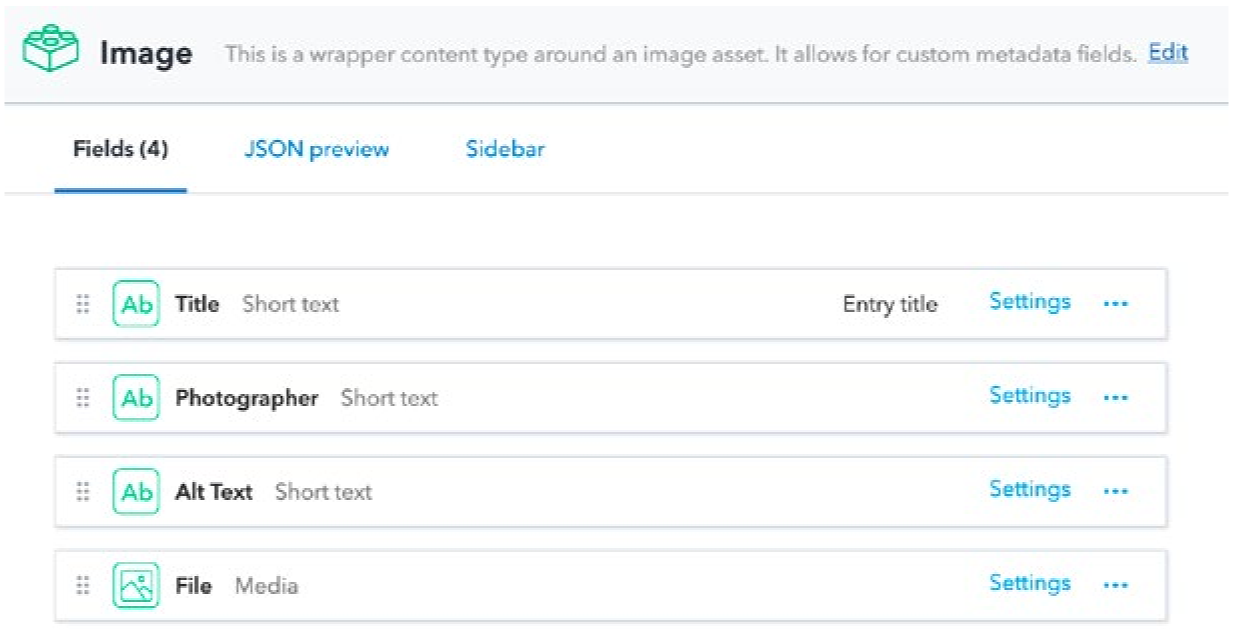
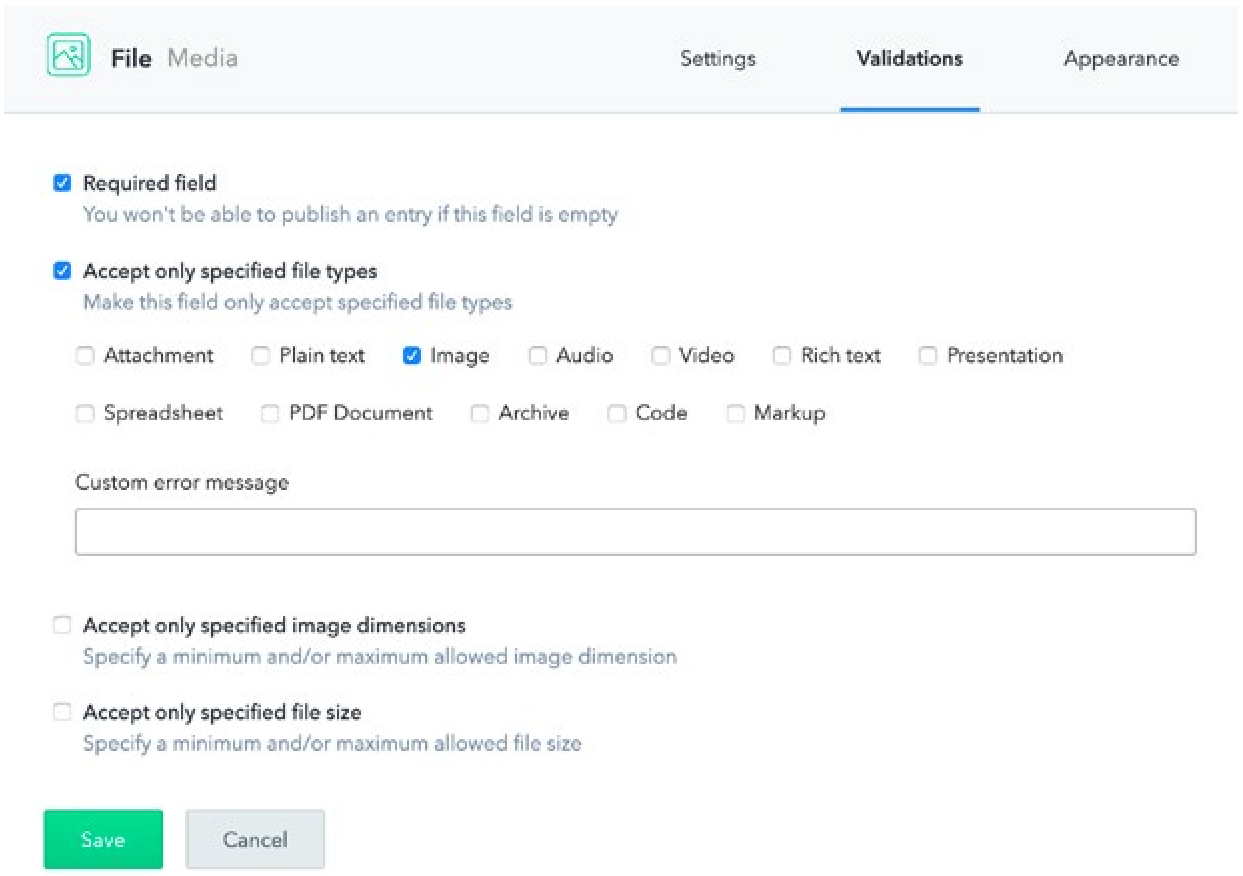


Image This is a wrapper content type around an image asset. It allows for custom metadata fields. Edit

Title	Short text	Entry title	Settings	•••
Photographer	Short text		Settings	•••
Alt Text	Short text		Settings	•••
File	Media		Settings	•••


File Media

Settings | **Validations** | Appearance

[✓] **Required field**
You won't be able to publish an entry if this field is empty

[✓] **Accept only specified file types**
Make this field only accept specified file types

[ ] Attachment [ ] Plain text [✓] Image [ ] Audio [ ] Video [ ] Rich text [ ] Presentation

[ ] Spreadsheet [ ] PDF Document [ ] Archive [ ] Code [ ] Markup

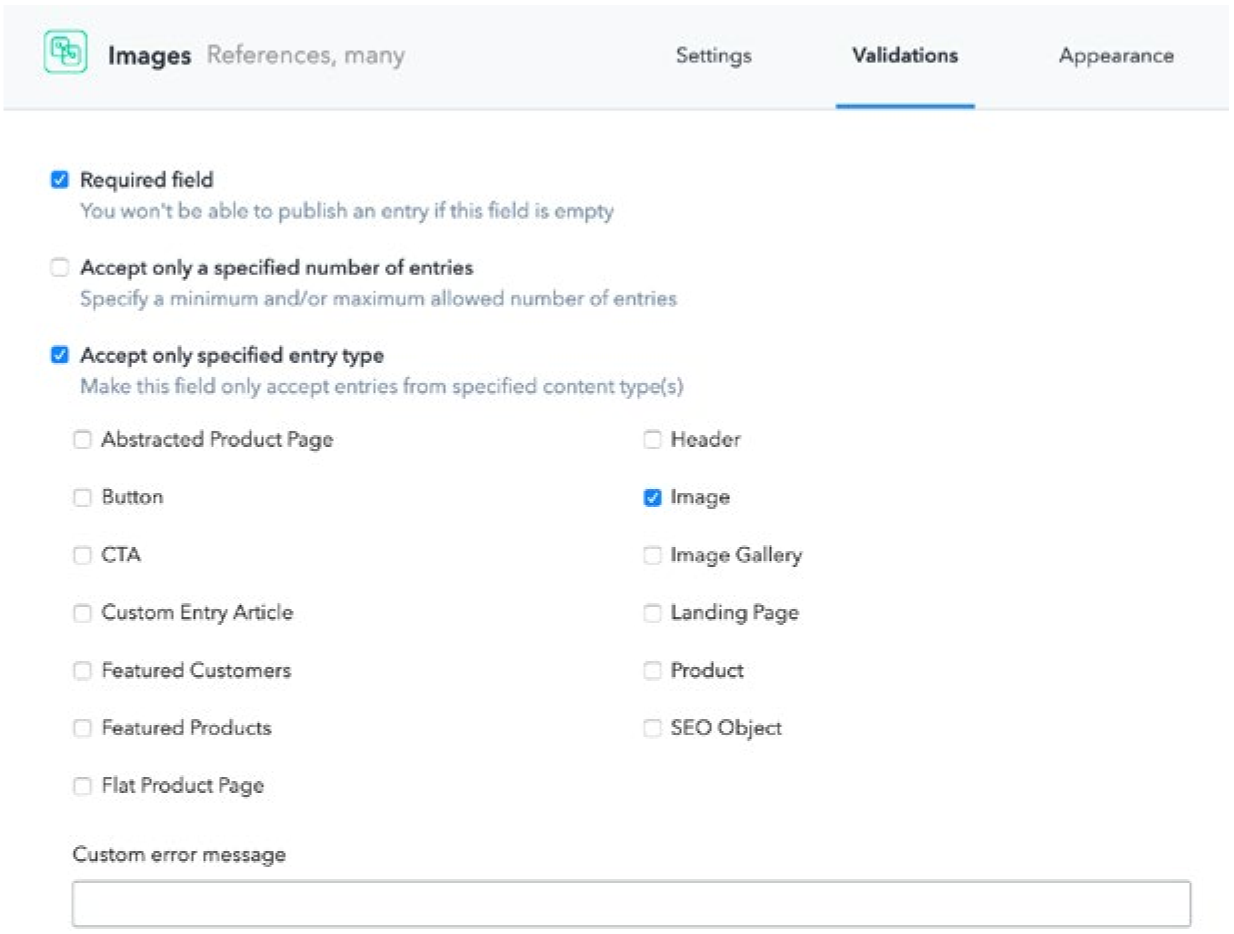
Custom error message

[ ] **Accept only specified image dimensions**
Specify a minimum and/or maximum allowed image dimension

[ ] **Accept only specified file size**
Specify a minimum and/or maximum allowed file size

STEP 2: VALIDATIONS ON THE PARENT CONTENT TYPE

Set a validation on any field in a parent content type that would need to reference this new content type. In this example, an Images carousel needs to reference this Image content type and nothing else.



[✓] Required field
You won't be able to publish an entry if this field is empty

[ ] Accept only a specified number of entries
Specify a minimum and/or maximum allowed number of entries

[✓] Accept only specified entry type
Make this field only accept entries from specified content type(s)

[ ] Abstracted Product Page	[ ] Header
[ ] Button	[✓] Image
[ ] CTA	[ ] Image Gallery
[ ] Custom Entry Article	[ ] Landing Page
[ ] Featured Customers	[ ] Product
[ ] Featured Products	[ ] SEO Object
[ ] Flat Product Page	

Custom error message

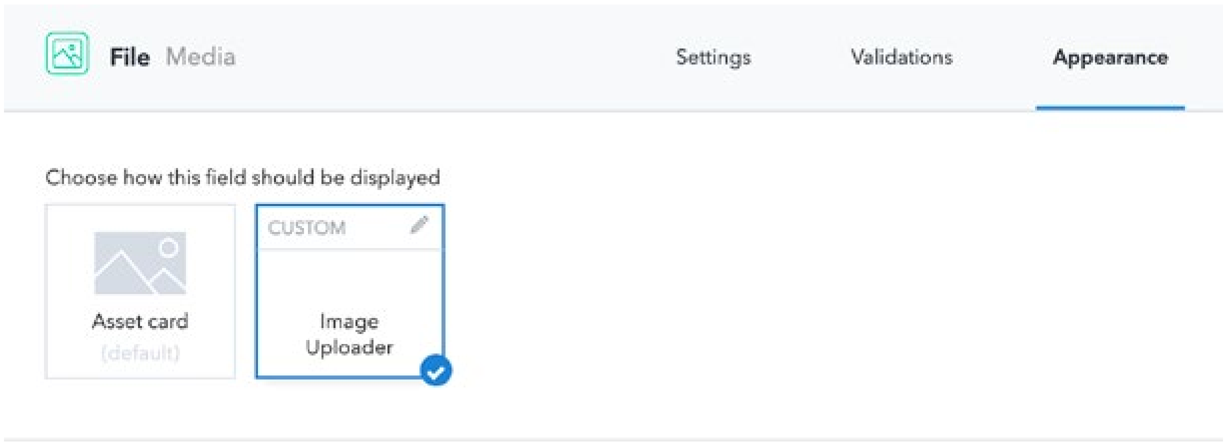
At this point, you could be ready to go, though we do not recommend stopping here. Stopping here would require your editors to click into and publish the media when creating content, which can cause problems.

STEP 3: INSTALL THE IMAGE UPLOADER UI EXTENSION (RECOMMENDED)

The Image Uploader UI extension changes the presentation of the web app to flatten the content creation flow. For a content creator or editor, the flow is a drag and drop to the Image content type entry and the UI extension field (which looks like a regular asset field).

The extension creates a new asset with that file and connects it to the entry in question. Nothing structurally changes about how the image exists as far as the APIs are concerned, but it makes it easier on your content creators and editors.

To take advantage of this feature, install the UI extension from the marketplace. Then, in the image content type, change the appearance of the media field to use the UI extension.



To a content creator, the flow now looks simpler.

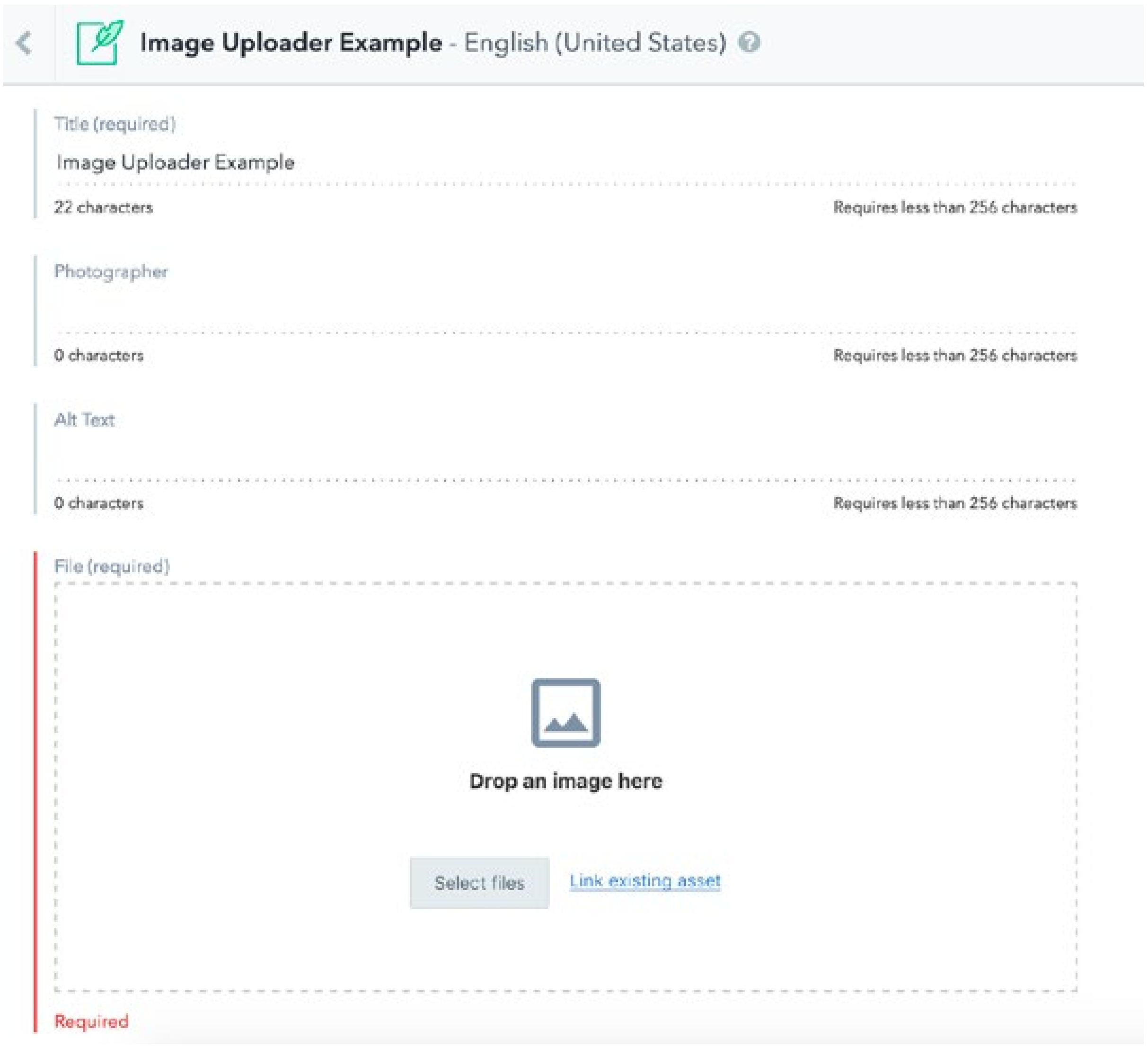


Image Uploader Example - English (United States)

Title (required)

Image Uploader Example

22 characters — Requires less than 256 characters

Photographer

0 characters — Requires less than 256 characters

Alt Text

0 characters — Requires less than 256 characters

File (required)

Drop an image here

Select files — Link existing asset

Required

Conclusion

In Contentful, adding custom fields to an image asset is done by creating a wrapper content type around a media asset. One defines these custom fields as they would define any content type. With a helper UI extension, it's a seamless experience for content creators and editors.

Configuring automatic backups

Edwin Maldonado

How to set up automatic backups for Contentful spaces using AWS Lambda Function and CloudWatch for scheduling.

Contentful maintains a command line interface (CLI) which offers a `space export` command that lets you download a backup of your Space. If you are interested in reading about the different options that you can pass to the space export command, go to the readme on Github.

```
        contentful space export — space-id {SPACE-ID} — environment-id {SPACE-ENV} — management-
    Token {CMA-TOKEN} — exportDir {YOUR-PATH}
```

This will create a backup that is a JSON file with the whole space structure (content model, entries, assets, roles & permissions and webhooks).

The CLI `space export` command works great but we have bigger plans. We want to automate this process by scheduling a routine to do it for us.

Thankfully, there is yet another tool called the Contentful export tool, which is a Javascript library that helps you to backup your space (as the `space export` CLI) in a programmatic way.

Process

1. Generate a backup of your space. By writing a NodeJS script that can collect the export options, download the backup and save it as a JSON file locally.
2. Upload the backup to AWS S3. Once you have the JSON dump locally, you can upload it to a S3 Bucket where you can collect the backups.
3. Wrap your backup script in a Lambda function so that you have the advantage of triggering it in a variety of ways. One good option is AWS CloudWatch, which can be configured to run as a Cron Job.
4. For the purpose of this tutorial, I will schedule my script on a daily basis. You are free to adapt it to your needs.

Implementation

Create a new AWS Lambda function from the AWS Console. You will quickly notice that there are three ways to create a lambda deployment package:

1. Use the built in editor
2. Upload a .zip file
3. Upload a file from Amazon S3

Use the `Upload a .zip file` option, which means that you will need to upload a zip file that contains our function `.js` file and the `node_modules` required for our function.

Create a `package.json` file with the below dependencies:

PACKAGE.JSON

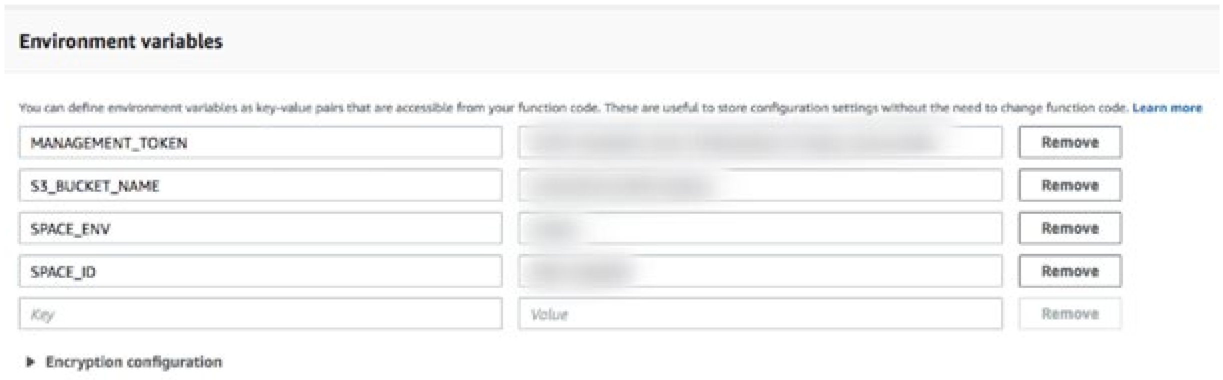
```
{
  “dependencies”: {
  “aws-sdk”: “2.450.0”,
  “file-system”: “”,
  “contentful-export”: “”
  }
}
```

Create another file called `index.js` This file will contain your main function — now you're only downloading our backup file to our local `tmp` folder:

INDEX.JS

```
 'use strict'
const AWS = require('aws-sdk');
const fs = require('fs');
const contentfulExport = require('contentful-export');
exports.handler = async (event) => {
 const local_backup_path = '/tmp/'
 const file_name = 'contentful_backup.json'
 const options = {
   spaceId: process.env.SPACE_ID,
   environmentId: process.env.SPACE_ENV,
   managementToken: process.env.MANAGEMENT_TOKEN,
   contentFile: file_name,
   exportDir: local_backup_path,
   useVerboseRenderer: false,
   saveFile: true
 }
 try {
   const result = await contentfulExport(options);
   return sendResponse(200, 'File downloaded');
 } catch (err) {
   console.log('Oh no! Some errors occurred!', err);
 }

 return sendResponse(200,'End of the Function');
};
```

Note that we are using environment variables on the script, you can set these `ENV` variables on the AWS Lambda function page.


Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more

MANAGEMENT_TOKEN		Remove
S3_BUCKET_NAME		Remove
SPACE_ENV		Remove
SPACE_ID		Remove
Key	Value	Remove

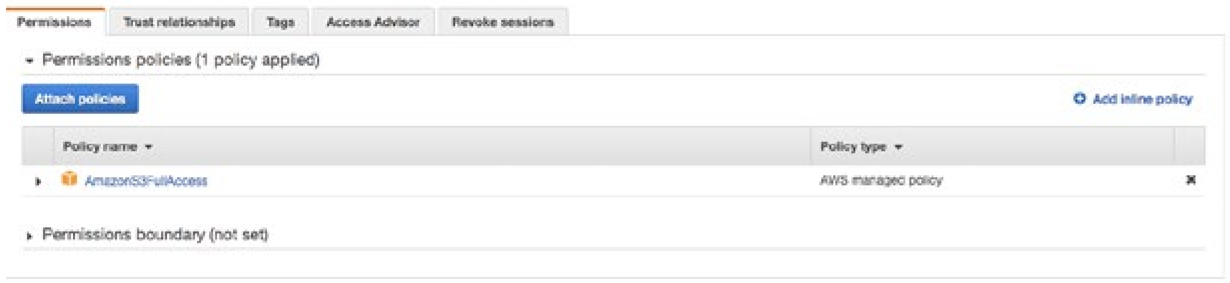
▸ Encryption configuration

You still need to upload the dump file to a S3 bucket. In the AWS Console, create a new bucket (or pick an existing one). Once you have your bucket in place, you need to set up the communication between our lambda function and the S3 bucket. In order to do that we have two options:

1. Use your API credentials in the lambda function to talk to S3.
2. Or set up a new IAM Role.

I lean towards the IAM Role option to keep things clean in our function. So again from the AWS Console, go to the `Security, Identity, & Compliance` section and choose IAM.

Create the new IAM role and make sure to attach the AmazonS3 policy to it.

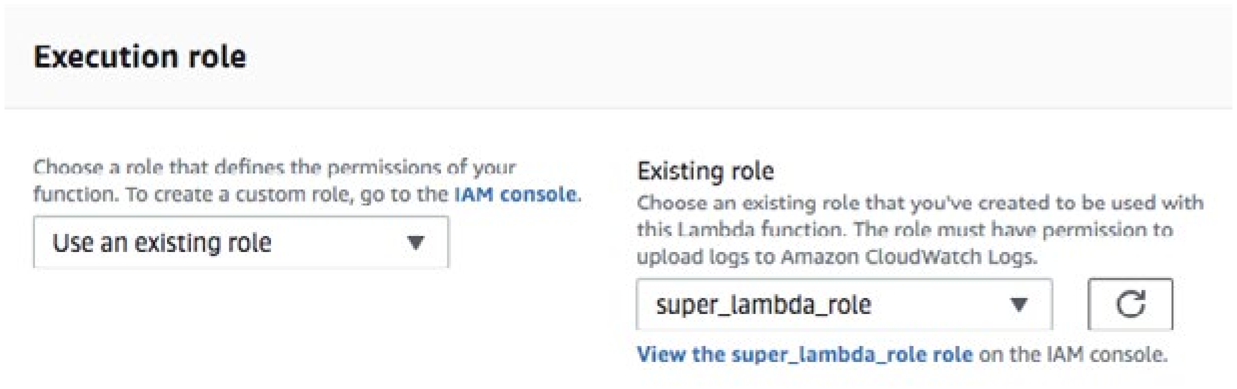


▾ Permissions policies (1 policy applied)

Policy name	Policy type
AmazonS3FullAccess	AWS managed policy

▸ Permissions boundary (not set)

Link the new IAM Role to your lambda execution option:



Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

Use an existing role

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

super_lambda_role

View the super_lambda_role role on the IAM console.

This allowed our lambda function to talk directly with our S3 bucket. Let's write the uploader part then, and this is how the final function looks like:

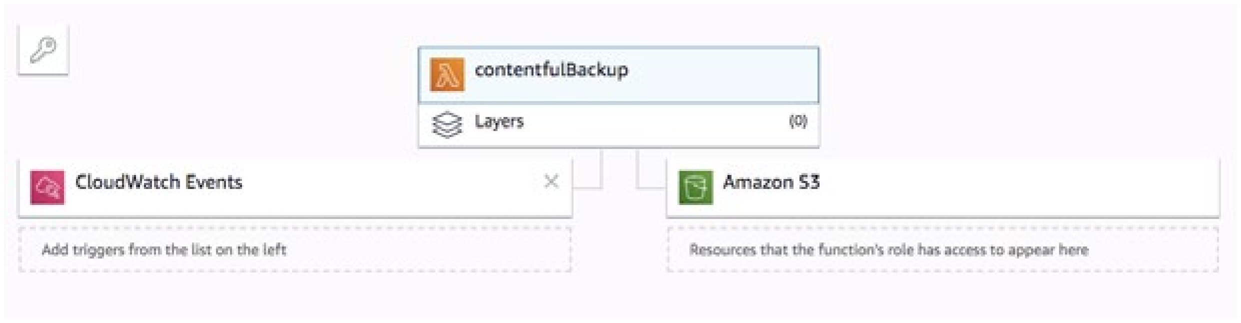
```
'use strict'

const AWS = require('aws-sdk');
const s3 = new AWS.S3();
const fs = require('fs');
const contentfulExport = require('contentful-export');

exports.handler = async (event) => {
  const local_backup_path = '/tmp/'
  const file_name = 'contentful_backup.json'

  const options = {
  spaceId: process.env.SPACE_ID,                                  // Replace it
  environmentId: process.env.SPACE_ENV,                           // Replace it
  managementToken: process.env.MANAGEMENT_TOKEN,                  // Replace it

  contentFile: file_name,
  exportDir: local_backup_path,
  useVerboseRenderer: false,
  saveFile: true
  }
  try {
  const result = await contentfulExport(options);
  console.log('Your space data has been downloaded!');
// Save file to AWS S3
console.log('Preparing file for AWS S3!');
const outputFile = local_backup_path + file_name;
let fileBuffer = new Buffer(fs.readFileSync(outputFile));
fs.unlinkSync(outputFile);

await uploadFile(fileBuffer);

return sendResponse(200, outputFile);
  } catch (err) {
  console.log('Oh no! Some errors occurred!', err);
  }

  return sendResponse(200,'End of the Function');
};

const sendResponse = (status, body) => {
  var response = {
  statusCode: status,
  body: body
  };

  return response;
};

const uploadFile = async (buffer) => {
  let params = {
  Bucket: process.env.S3_BUCKET_NAME,
  Key: 'backups/' + Date.now().toString() + '.json',
  Body: buffer
  };

  return await s3.putObject(params).promise();
}
```

Scheduling the function

Pick a trigger for our lambda function, specifically a CloudWatch Event. Create the trigger and schedule it as you wish, I used the daily option:

```
`Schedule expression: rate(1 day)`
```

Make sure to compress your deployment package (`index.js` and `node_modules`), upload the zip file to the lambda’s function code section and test it!


contentfulBackup
Layers
(0)
CloudWatch Events
Amazon S3
Add triggers from the list on the left
Resources that the function's role has access to appear here

This is how the final solution looks like on the AWS lambda page.

Data ownership with Contentful

Arjun Ganesh

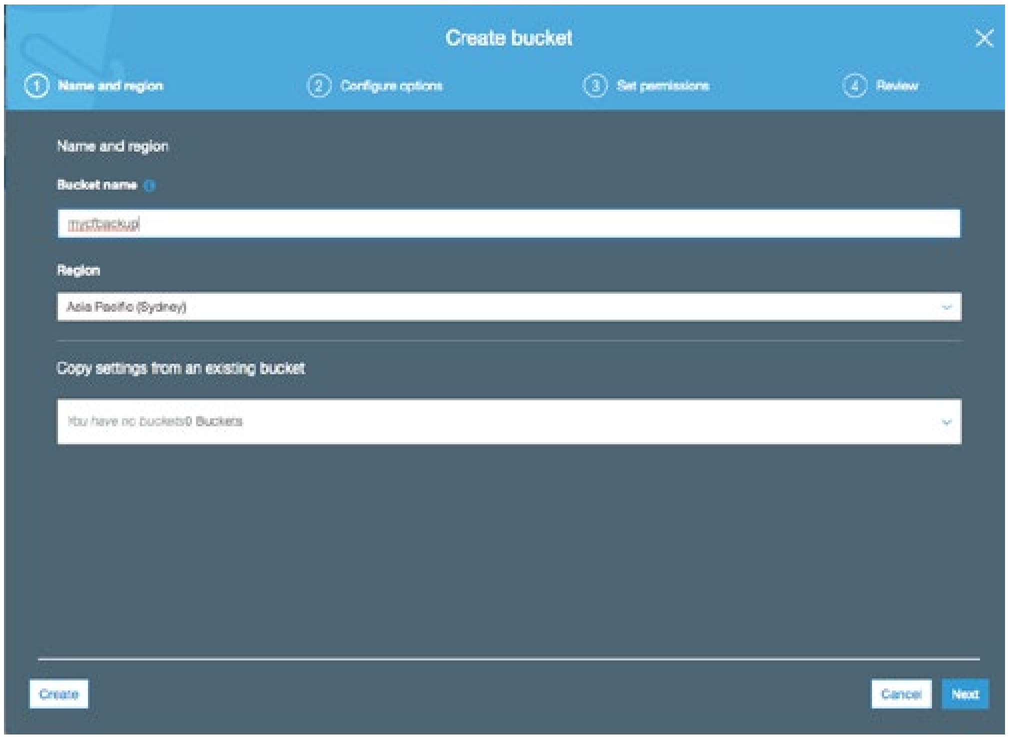
How to own your data and make sure it is always available no matter what.

Contentful provides 99.96 to 99.99 availability depending on your enterprise SLA. This is sufficient in most cases, but sometimes you need even more reliability. Enterprises that store critical data in Contentful may need to satisfy legal or compliance requirements associated with data ownership and business continuity. Regulated and public enterprises might need to have backup protection in the unlikely event that Contentful ceases to operate in the future.

Contentful and AWS S3 offer a reliable, secure and out-of-the-box integration that solves these demands. It backs up data on your own infrastructure and allows you to maintain an audit log. You could also build a custom HA/DR/BPC procedure with Contentful.

Solution

Integrating AWS and Contentful begins by setting up a S3 bucket on AWS. Below are screenshots that show how a S3 bucket being created via the AWS console:



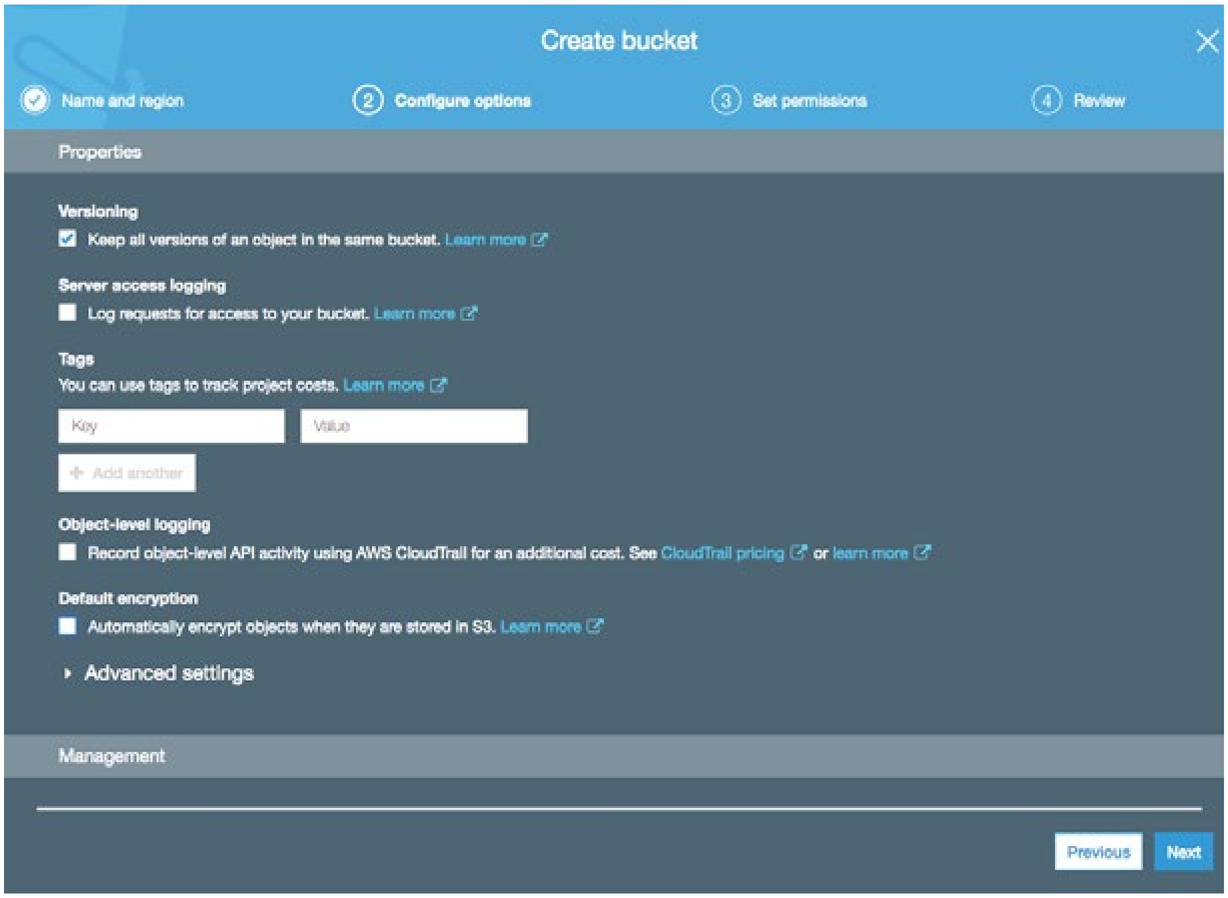
Name and region

Bucket name: mycfbackup

Region: Asia Pacific (Sydney)

Copy settings from an existing bucket: You have no buckets0 Buckets

You will need to enable versioning, as seen below. This stores multiple versions of the same file as one:



Properties

Versioning
[✓] Keep all versions of an object in the same bucket. Learn more

Server access logging
[ ] Log requests for access to your bucket. Learn more

Tags
You can use tags to track project costs. Learn more

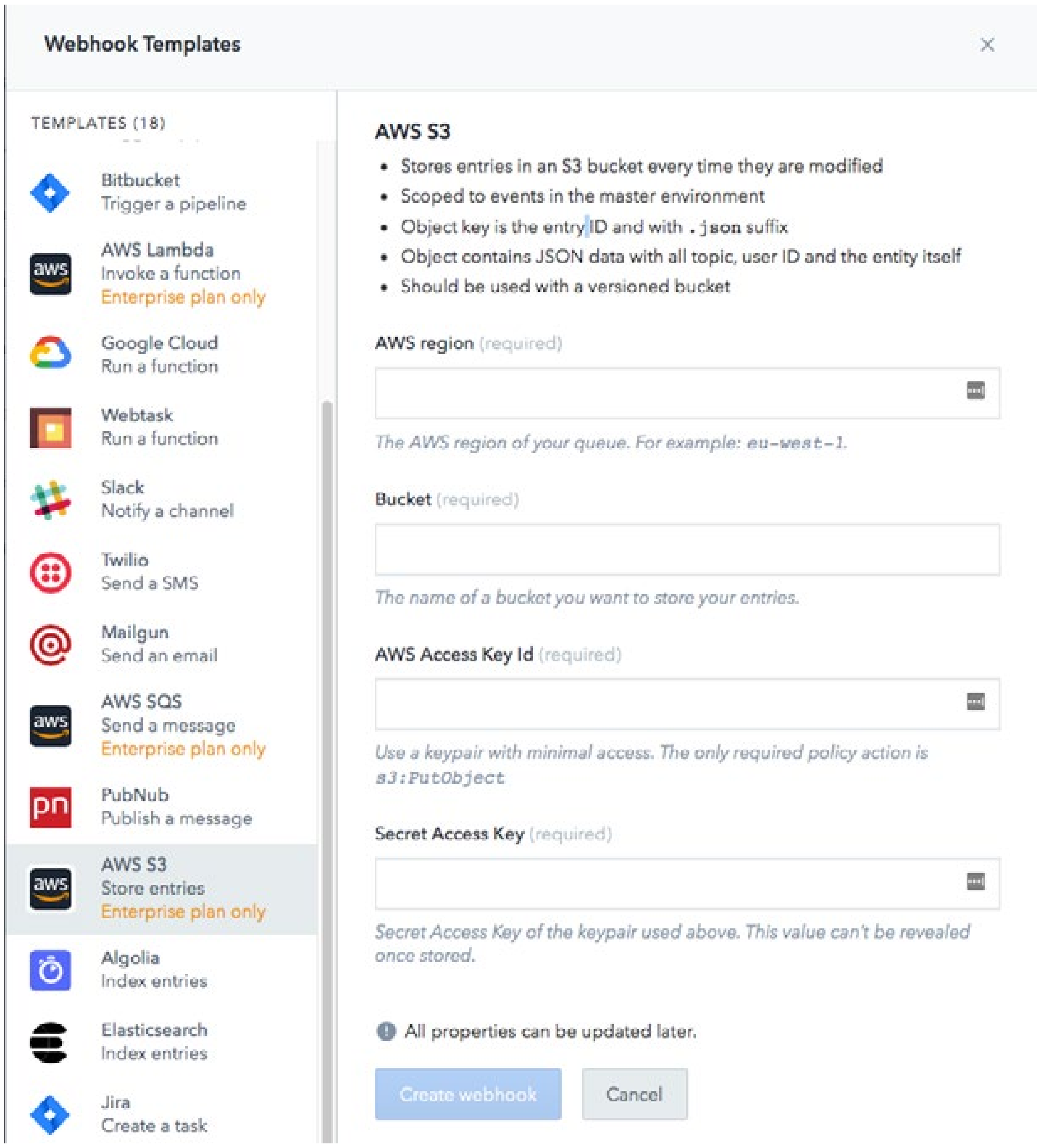
Object-level logging
[ ] Record object-level API activity using AWS CloudTrail for an additional cost. See CloudTrail pricing or learn more

Default encryption
[ ] Automatically encrypt objects when they are stored in S3. Learn more

Use an existing user with the right permissions, or generate a new Access ID and Access Key that can be used from Contentful to write in this folder. The latter is the better option.

That's all that is required from AWS. The remaining configuration can be done from the Contentful web app. You need to be an admin or an owner of the space that you're using to do so.

Create a new webhook within Contentful using the AWS S3 template.



AWS S3

- Stores entries in an S3 bucket every time they are modified
- Scoped to events in the master environment
- Object key is the entry ID and with `.json` suffix
- Object contains JSON data with all topic, user ID and the entity itself
- Should be used with a versioned bucket

AWS region (required)

The AWS region of your queue. For example: `eu-west-1`.

Bucket (required)

The name of a bucket you want to store your entries.

AWS Access Key Id (required)

Use a keypair with minimal access. The only required policy action is `s3:PutObject`

Secret Access Key (required)

Secret Access Key of the keypair used above. This value can't be revealed once stored.

All properties can be updated later.

Enter the region and the name of the bucket, along with the access credentials.

When an entry is created or updated, a JSON file is stored in this S3 bucket with all the info of the entry. The JSON file is versioned — you always see the latest version. If you want to publish content and aren't concerned with draft content, then update the events when the webhook is triggered under the webhook settings.

This covers most of the use cases:

1. Backup of contentful data in your AWS S3
   a. You own your data, it is on your cloud.
   b. You're ensured compliance and data availability in the unlikely event that Contentful does not exist in the future
3. Create an audit log, because the data contains user information for each entry. Notice each JSON is associated with the user ID who made the change.

We still need to do one more thing to satisfy HA, DR and BCP requirements. If you are accessing Contentful directly from a client:

1. Create a lambda or similar function that can be called from the client
2. In the lambda make the Contentful call
3. If this fails, make the call to the S3 bucket to get the corresponding JSON
4. Massage the JSON from the S3 and remove unwanted elements
5. Return it to the client

If you have a service that merges contentful content and some form of React/Mustache templates, update this service to call the S3 bucket (similar to above) if Contentful is down.

Now — even if Contentful is down — your digital experience will still function with the latest published data from the S3 bucket.

Conclusion

This simple, out-of-the-box integration between Contentful and AWS S3 is reliable and solves complex enterprise requirements. It demonstrates just how flexible and reliable SaaS and API based integrations are.

Useful links

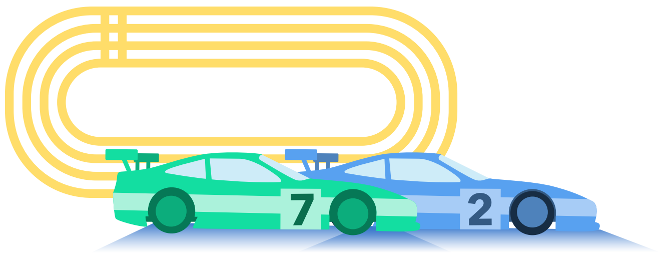
Webhooks

Contentful Delivery APIs

AWS S3 bucket (create, read)

Blue-green deployment

Sia Rad



User experience plays a crucial role in the success of every business. Displaying the latest content to customers without interrupting their experience is a win-win for both businesses and customers.

Some businesses struggle to make their latest changes available to customers with the agility that the market demands due to the redundant and inherent dependencies in their developer operations (DevOps) process. Time to market increases when platforms cannot eliminate the dependencies between making and deploying iterative changes.

Business leaders and marketing teams need an agile approach to let customers know about products and services just in time. Developers need a platform that enables them to deploy changes while future changes are being developed without impacting customer experience. And consistent and reliable access to the latest content and features keeps customers happy.

Solution summary

You can leverage Contentful's out-of-the-box capabilities to build the solution. Refer to CMS as Code to learn how to use spaces and space environments if your DevOps is not already doing so.

Two spaces can be dedicated to your agile DevOps (Blue and Green) and a config file (self-hosted) to switch from the Blue to Green or vice versa. Doing so enables you to serve your front-end with the content from one space (Blue acting as the active space serving as production) while your dev team works on new changes in the other space (Green acting as the standby space serving pre-production) without impacting production. Once the changes are ready to go live, a config change would promote the pre-production to production and vise versa. As a result, Blue becomes pre-production and Green becomes production until the next round of continuous deployment.

Prerequisite concepts

Take some time to review and learn about the following concepts before you continue:

1. CMS as Code, also known as Infrastructure/Content as Code
2. Managing multiple environments
3. Contentful CLI and CMA SDKs

Use case

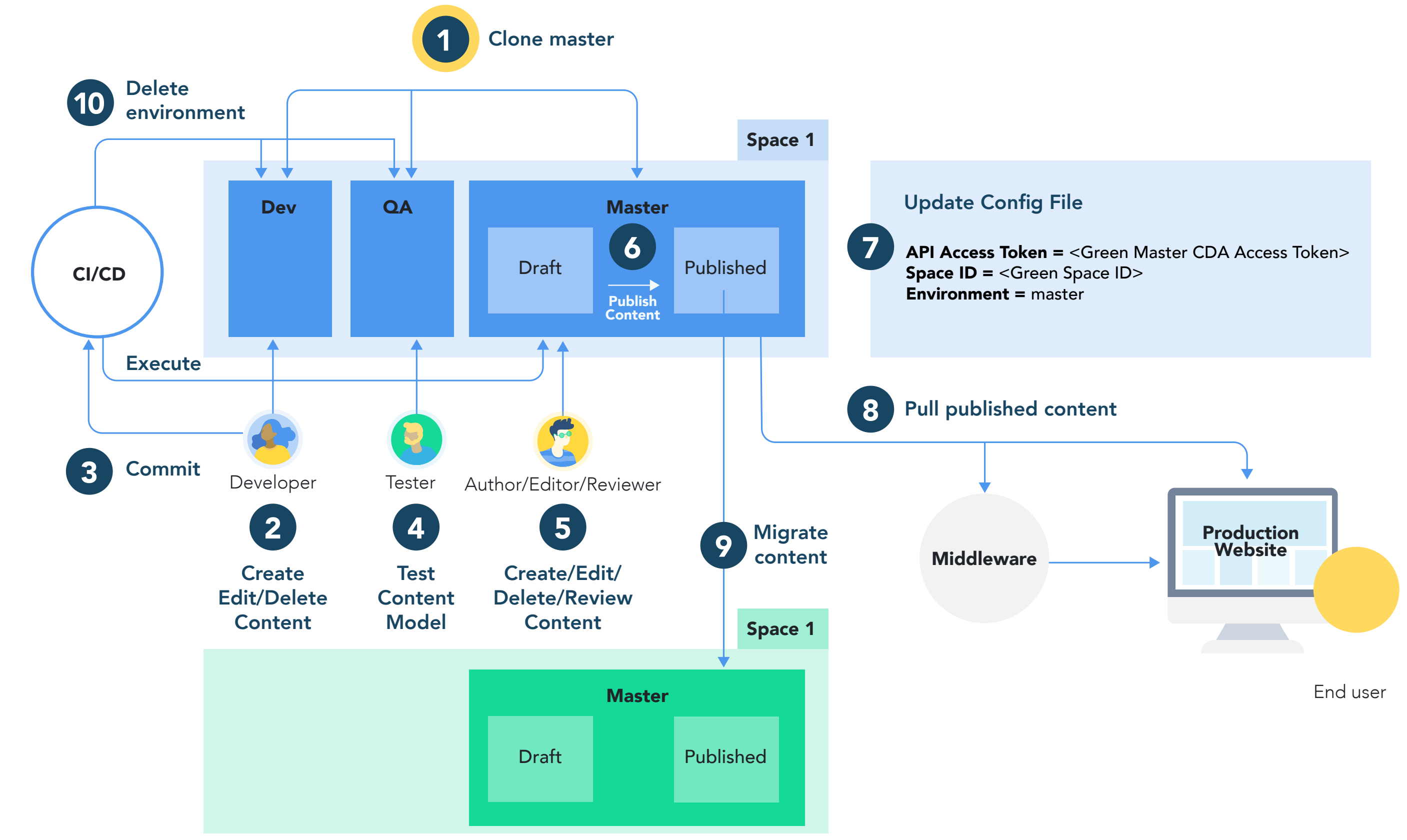
A Healthcare Provider company in North America with more than 75 million members wanted to build a platform that had a high level of availability for customers. They also wanted their IT team to build and deploy content models, and revamp, review and publish content without interrupting the customer experience. Previously, they made iterative releases and pushed changes from development to test and production. But each deployment of a new release caused downtime. Some of the deployments took hours and needed to happen overnight to minimize the impact on the customer experience.

The requested platform required a DevOps process that uses the following components:

1. Two spaces (known as Blue and Green)
2. Space environments (e.g. Dev, QA, Master)
   Note: The number of environments and the naming convention could vary for different businesses.
3. Contentful CLI/CMA scripts to apply changes from Dev, QA, Master
4. Code repository to commit the scripts and check out and run as needed
5. Script to publish the draft content, if applies per release
6. A self-hosted config file containing:
   a. Prod Space API Access Token
   b. Prod Space ID
   c. Prod Environment Name (this is currently optional, since it isn't possible to rename the "master" environment at this point. Let's keep it here for forward compatibility.)

Blue-green deployment process

Now that you have a way to build the Blue-Green architecture, let's look at what process should be used to properly build and deploy changes in this model.



1. Let's assume that the Blue space (production) is serving the production front-end and the Dev and QA envs are created as clones of the Master on the Green Space (Pre-Prod).
2. Script the changes, run it on the Dev environment, and do unit testing
3. Commit the script into your CI/CD code repository
4. Deploy the changes on the QA env by running the same script and review and QA the changes
5. Deploy the changes on the Master by running the same script and preview and validate the changes
6. Publish any content that needs to be part of this release. Note: At this point, you are ready to serve production front-end by the latest changes from the Green space.
7. Update the config file with the Green space properties (i.e. Space ID, CDA Access Token, and master environment name)
8. Start serving the front-end by pulling the latest changes from the Green space
9. Assuming you have already built a script to refresh everything on the master environment of the Blue space with a copy of the master environment from the Green space, run the script and validate the results.
10. Dispose of the environments on the Green space if your normal practice is to recreate those environments for every sprint.
11. Repeat the same process for the next release by switching Blue and Green space roles

Expected outcome

Implementing this solution will enable businesses and their IT teams to streamline their DevOps process and deploy changes to production without having production downtime and interrupting their customer experience. It will improve time to market and increase customer satisfaction and sales.

Conclusion

A Blue-Green deployment offers a solution for IT teams unable to deploy changes to production without degrading customer experience. A Blue-Green deployment provides the architecture to automate the process, eliminate dependencies, and deploy changes without scheduled production outages or unscheduled interruptions to service.

Useful links

CMS as Code

Managing multiple environments

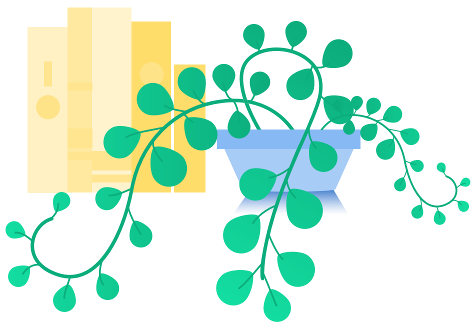
Contentful CLI

Contentful Delivery APIs

Contentful Management APIs

Integrations with Contentful

Daniel Schley



One of the most powerful aspects of Contentful is its extensibility. UI extensions can enhance Contentful's web app in almost any way expressible in HTML and Javascript—including the integration of external APIs in the web app.

The high-level principles for these integrations can seem simple, but complicated questions often arise during implementation. Does this field need to be in Contentful? Content is already stored in another system, so should we sync it to Contentful or integrate it as a reference? How can we combine data from multiple source systems for common use in one product, such as a website?

Almost all use cases built with Contentful entail publication of content to a publicly available information system, such as a website or an app. Most editorial content is managed in Contentful, which means change happens in Contentful and potentially should trigger follow up actions with Contentful as the leading system and event source. More complex use cases require the integration of data hosted and/or managed in another system like a product information management.

Use case

ACME Corp is a telecommunications company that frequently publishes product pages for the mobile phones that they sell through their website. They decided to use Contentful to managing their marketing websites. They already had a product information system (PIS), and they wanted to use an enhanced version of the content in their marketing website. They respected the separation of concerns and avoid redundancy; they had learned how important this was when they lost product information due to last-minute changes during a product launch phase. Recovering that information required significant effort since the same data needed to be changed in several places in separate systems, such as CMS and PIM.

Data

The following content was relevant to their product website, in addition to what is available in the PIM.

Simple content model for product description page:

- Product description
- Product images

The PIM already contains product information, including:

- Title/product name
- Manufacturer
- Colors
- Configurations

ACME needed an integration that respected a loose coupling among integrated components so that their content creators could see relevant information and avoid duplicating information in the CMS. They also needed to present their products in a seamless and unified way.

API integration

ACME needed to move quickly, so they decided to use the easiest integration possible. Their architecture required a BFF for additional reasons, so they integrated via PIM reference and enrich product information in Contentful.

1. Contentful Date: Contentful data contain the product ID as productReference.
2. PIM Data: Data from the PIM can be retrieved via product ID subsequently from the backend.
3. Architecture: The backend fetches product data first from Contentful, and can then use the referenced product ID to resolve data in the PIM and provide a unified response for the frontend.

Web app integration

Web app integrations allow editors to reliable link PIM data and additional content in Contentful without manually creating a reference.

UI extensions to the rescue

In ACME's case, the answer is a single UI extension. The product entry in Contentful shows a drop-down menu that contains the product selection where the Content Creator selects the referenced product. The UI extension stores the corresponding product ID and fills the internalTitle field automatically, which is validated for uniqueness. The additional title field serves web app search purposes and makes it easier to find products. The entry also contains the description and the corresponding images.

Consistency

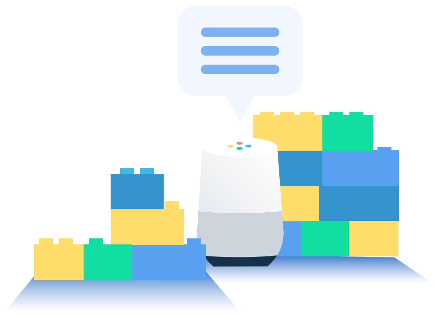
Because there might be changes in the PIM products name, there is a chance for inconsistency. ACME responded to this challenge by implementing a serverless function that uses the Content Management API to update the title in corresponding change situations triggered via outgoing PIM webhooks.

Future expansion

ACME decided to adopt GraphQL in the future, which will allow even easier integrations via schema stitching and a unified ACME GraphQL API for all systems managing external data.

Smart web applications

Arjun Ganesh



Your web application should have a top-down approach from the view of your customers, putting your customers at the top. It can be as simple as a sleek, user-friendly layout of interactive images sprinkled with personalised data. The latter is what adds intelligence to your site. Customers want to see more of the things they like in an enjoyable, immersive experience.

Various cloud based services make this possible. Using Contentful, Optimizely and Google Analytics, you can achieve simple, mildly personalised web applications that can then be moderated based on performance statistics. It's like lego pieces for building a site.

Where to start?

Every web application needs content. Let's start with that. You need somewhere to securely store the content that allows you to retrieve and edit it quickly without developers. Get Contentful. Your editors and authors can manage content within Contentful, and your developers can retrieve content via APIs and integrate it with their front-end code of choice.

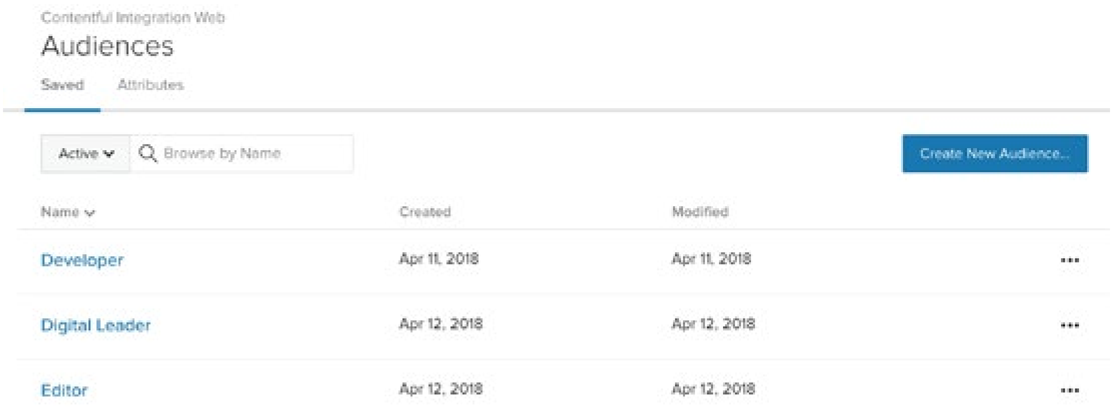
Content creators are always trying to figure out what content to display when, on which page and to whom. An experimentation platform enables them to do this without the guesswork. Optimizely seamlessly integrates with Contentful and equips you with the capability of targeting audiences and the APIs to access them. Google Analytics provides the data that content creators need to make strategic decisions. Data about what performs well, what makes the highest sale and generates the most traffic.

Build build build

Create a content model inside Contentful that can power the data for your user experience and front-end code. Get content creators to start populating content. Numerous articles cover this topic in detail, and I have listed some at the end of this article.

To use the Optimizely feature in Contentful, the fields that content creators would like to experiment on need to be references. For example, if a call to action button is required, make it a child reference type to you home page parent type.

Next, setup your experimentation platform. Create variations and audiences. Define targeting criteria such as cookies or other context parameters such as device or location).

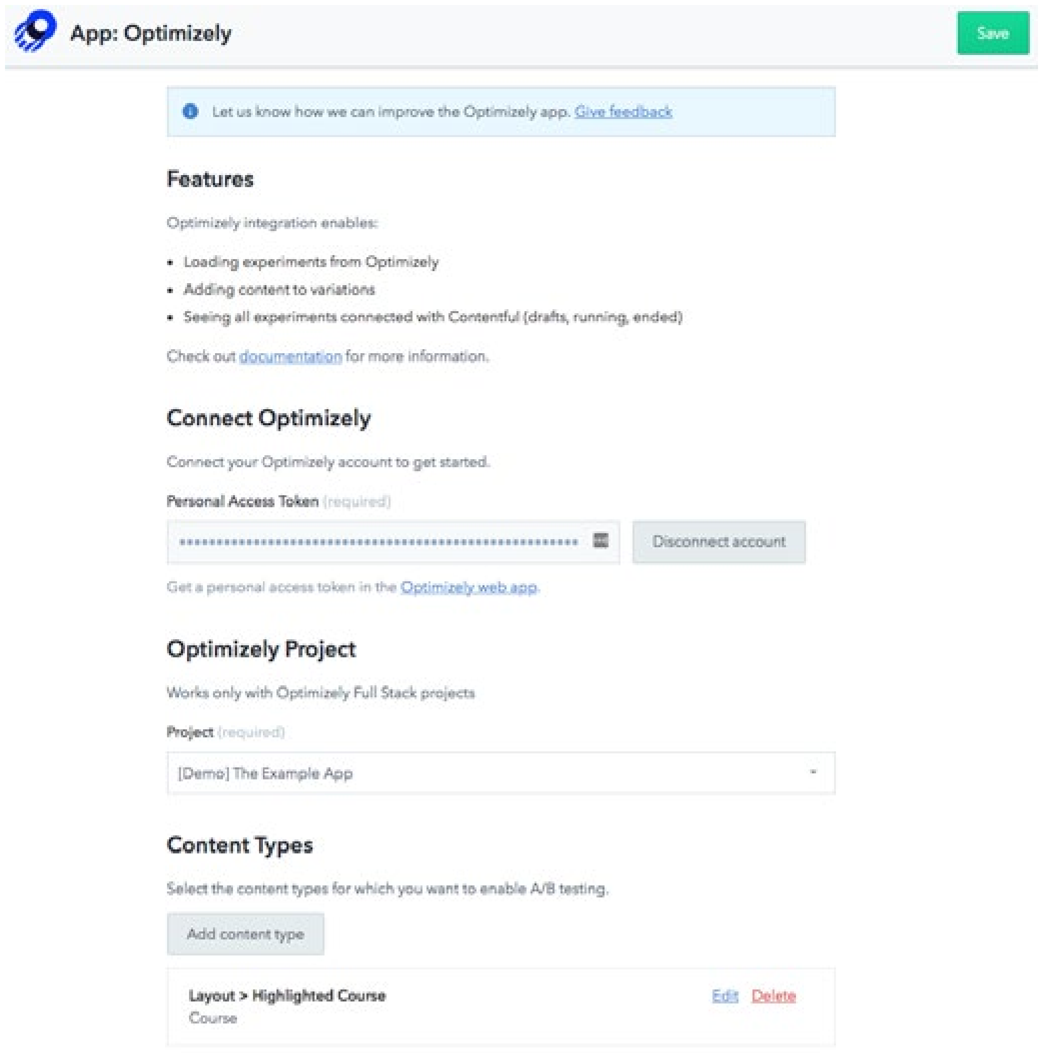


Contentful Integration Web

Audiences

Name	Created	Modified
Developer	Apr 11, 2018	Apr 11, 2018
Digital Leader	Apr 12, 2018	Apr 12, 2018
Editor	Apr 12, 2018	Apr 12, 2018

To connect Optimizely to Contentful, go to apps in the Contentful web app and enable the Optimizely app. You should see a screen similar to below:



App: Optimizely

Let us know how we can improve the Optimizely app. Give feedback

Features

Optimizely integration enables:

- Loading experiments from Optimizely
- Adding content to variations
- Seeing all experiments connected with Contentful (drafts, running, ended)

Check out documentation for more information.

Connect Optimizely

Connect your Optimizely account to get started.

Personal Access Token (required)

••

Get a personal access token in the Optimizely web app.

Optimizely Project

Works only with Optimizely Full Stack projects

Project (required)

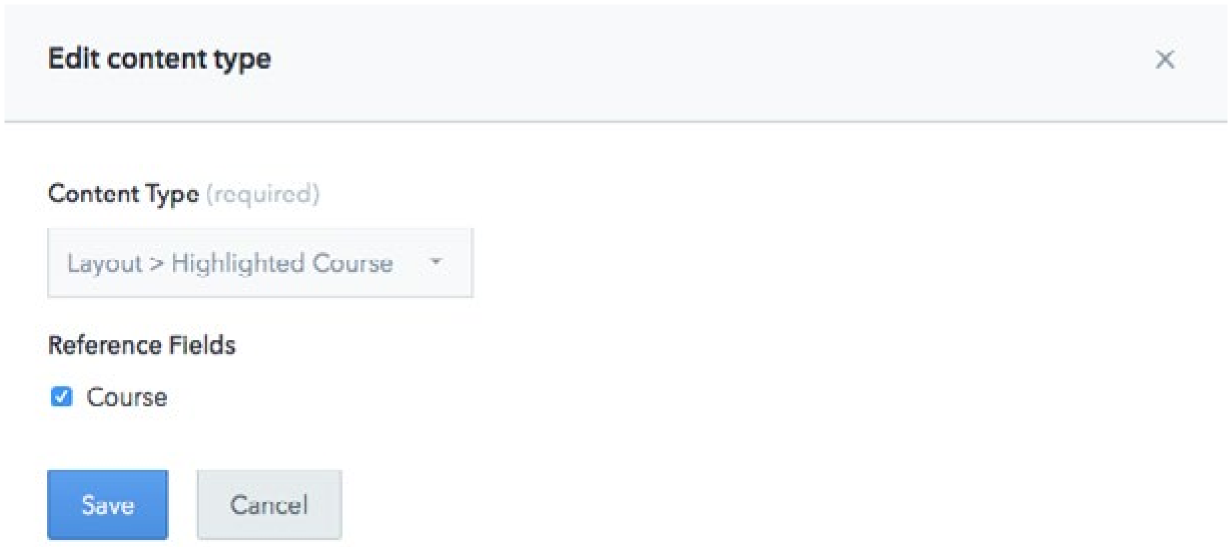
[Demo] The Example App

Content Types

Select the content types for which you want to enable A/B testing.

Layout > Highlighted Course
Course

Enter your Optimizely Personal Access Token and connect your account. This will bring a list of your projects. Select the one you created. Add the relevant Contentful content types that you want to run this experiment against. If you select the parent type, you should see the referenced child that you want to experiment with.


Edit content type

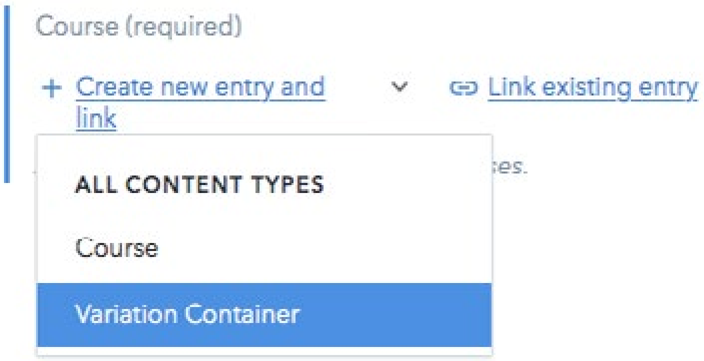
Content Type (required)

Layout > Highlighted Course

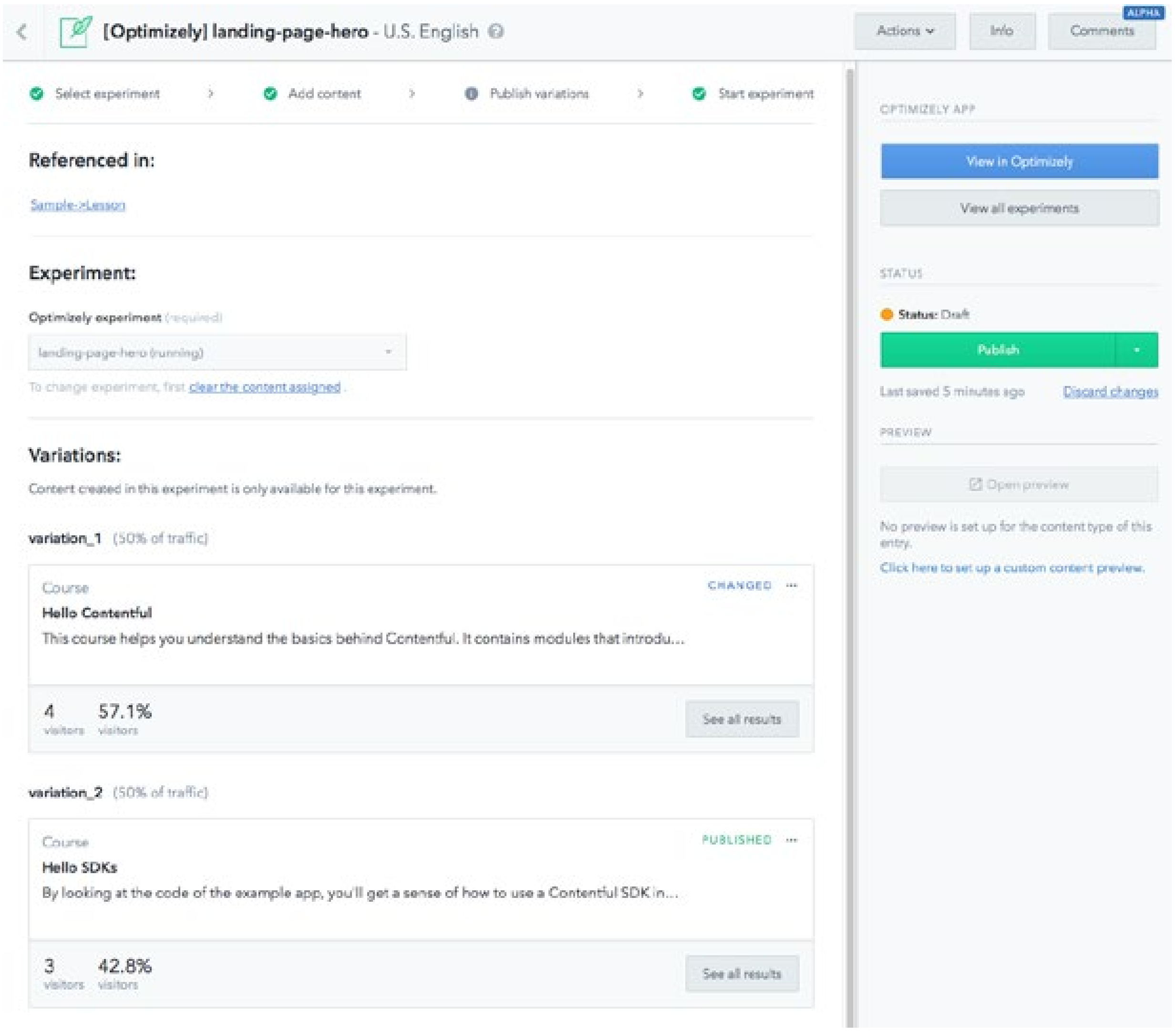
Reference Fields

☑ Course

When you create an entry, you will see a drop-down menu that gives you the option to select a variation container.


Course (required)

A variation container brings up the Optimizely variations and allows you to select the content you would like to display for each variation.



Referenced in:

Sample->Lesson

Experiment:

Optimizely experiment (required)

landing-page-hero (running)

To change experiment, first clear the content assigned.

Variations:

Content created in this experiment is only available for this experiment.

variation_1 (50% of traffic)

Course
Hello Contentful
This course helps you understand the basics behind Contentful. It contains modules that introdu...
CHANGED

4 visitors 57.1% visitors

variation_2 (50% of traffic)

Course
Hello SDKs
By looking at the code of the example app, you'll get a sense of how to use a Contentful SDK in...
PUBLISHED

3 visitors 42.8% visitors

Status: Draft

Last saved 5 minutes ago

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

Click here to set up a custom content preview.

Publish the entry and let the front-end code work with the Optimizely API.

Surface your google analytics data inside the Contentful web app as a custom sidebar UI extension. You need to build a JS page that surfaces stats from google analytics and install it as a sidebar UI extension in the web app. I have included additional resources under useful links.

Conclusion

Contentful provides a single platform for practitioners to create content, run Optimizely experiments and make decisions based on data derived from Google Analytics. This reduces complexity and increases go-to-market speed. It empowers practitioners to create content based on numerous experiments — resulting in an awesome customer experience.

Useful links

Content Modeling Basics

Contentful Delivery APIs

Contentful Management APIs

Sample Google Analytics Integration

Telus Personalisation Case Study