PUBLIC
2026-07-02

Application Content Deployer Buildpack User Guide

Content

1 Application Content Deployer Buildpack. 3

2 Prepare Your System for Deployment. 5

3 Supported Content Runtimes. 7

4 Deployment Scenarios. 8

4.1 Deploy CAP Application with Portal Service. 8

Build Content Deployment Image. 8

Configure Helm Chart. 9

4.2 Deploy CAP Application with SAP Application Frontend Service. 11

4.3 Deploy CAP Application with SAP Build Work Zone. 13

Configure the Helm Chart. 14

4.4 Deploy CAP Application with SAP Build Work Zone Using the Common Data Model. 16

Build the Content Deployment Image. 17

Configure the Helm Chart. 18

4.5 Deploy Standalone UI Application with Application Frontend Service. 19

4.6 Deploy Single Target Application with ZIP Archives. 23

5 Troubleshooting. 28

1 Application Content Deployer Buildpack

The Application Content Deployer Buildpack packages application content, such as HTML5 apps and Application Frontend service, into an OCI image for deployment on SAP BTP, Kyma runtime.

Overview

The application content deployer buildpack is a Cloud Native Buildpack (CNB) that packages application content archives together with the deployer into an Open Container Initiative (OCI) image. Running this OCI image in a container deploys the application content to SAP BTP, Kyma runtime.

Key Capabilities

- Automatic detection of the application content archives and its content types.
- Packages application content into an OCI container image
- Supports deployment to SAP BTP, Kyma runtime as a Kubernetes Job
- No Dockerfile requirement
- No manual configuration of a separate deployer module
- Supports the Common Data Model (CDM) approach by detecting the `cdm.json` file

Supported Scenarios

CAP Applications with SAP Build Work Zone:

Deploys CAP applications integrated with SAP Build Work Zone.

CAP Application with Portal Service:

Deploys a CAP application integrated with Portal Service.

CAP Application with SAP Build Work Zone Using the CDM:

Deploys CAP application integrated with SAP Build Work Zone following the common datal model approach.

CAP Application with Application Front-end :

Deploys CAP application with Application Frontend service to SAP BTP, Kyma runtime

Standalone UI Application:

Deploys a standalone UI application to SAP BTP, Kyma runtime.

Accessing Application Content Deployer Buildpack

To access the buildpack, see: Application Content Deployer Buildpack

2 Prepare Your System for Deployment

The following table describes the tools, services, and configurations required before deploying with the application content deployer buildpack.

Tools	Purpose
SAP BTP Setup	• Create or access an SAP BTP subaccount with Kyma runtime enabled. • For Application Frontend deployments, subscribe to the Application Frontend Service (`build-default` plan) and assign the Application Frontend Developer and Viewer roles.
Install the Kubernetes CLI (`kubectl`)	Install the Kubernetes CLI (`kubectl`) to interact with the Kyma cluster. Also, to manage Kubernetes resources, inspect pods, and view logs during deployment and troubleshooting.
Install kubelogin	Install the `kubelogin` credential plugin for Kubernetes authentication. This enables secure OIDC-based login to SAP BTP Kyma clusters.
Log in to your Kyma cluster	Configure your Kubernetes context and authenticate against the target SAP BTP Kyma cluster. By doing so, all subsequent `kubectl` and Helm commands target the correct environment.
Install the Helm CLI (`helm`)	Install the Helm CLI (`helm`). Helm is used to package and deploy the application using Helm charts generated by CAP/CDS tooling.
Install the Cloud Native Buildpacks CLI (`pack`)	Install the Cloud Native Buildpacks CLI (`pack`). This tool builds OCI container images using the application content deployer buildpack.
Install a container management application	Install Docker Desktop or another OCI-compatible container runtime. A running container daemon is required by the `pack` CLI to build and publish images.

Tools	Purpose
Download and set up the project in your local environment	Clone the CAP sample application repository and prepare the local workspace by installing Node.js dependencies and verifying CAP tooling. This provides the project structure used in later deployment steps.
Install the SAP CAP CDS Command Line Interface (CLI) (`@sap/cds-dk`)	Install the SAP CAP CDS command line interface (CLI) to use CDS commands in your deployment workflow.
Container Registry Access	Ensure access to an OCI-compatible container registry. For example Docker Hub or any container registry, where deployment images can be pushed and stored.

3 Supported Content Runtimes

The application content deployer buildpack supports the following content runtimes:

- SAP API Management
- SAP Cloud Integration
- SAP Workflow Management
- Workflow
- SAP Business Rules Management
- Mobile Applications
- Process Visiblity
- HTML5 Repository
- SAP Destination Service
- Role Collections
- Job Service
- Business Logging
- Portal
- Application Frontend Service

4 Deployment Scenarios

4.1 Deploy CAP Application with Portal Service

Prerequisites

- You have completed the steps in the Prepare Your System for Deployment [page 5] section.
- You've integrated the portal service using CDS add portal. See: Portal and Launchpad Configurations

Context

The application content deployer buildpack detects the `resources/` folder containing HTML5 application ZIP archives and the `portal/` folder or `portal*.zip` file. It then builds the content deployment image. This replaces the need for a separate `portal-deployer` section module in `values.yaml` file of helm charts.

Build Content Deployment Image

Procedure

1. Add `portal/` folder to the `html5-deployer/resource/` directory where the HTML5 application ZIP archives are stored:

   Sample Code

   ```
   html5-deployer/
   resources/
       app-content.zip
   portal/
       portal-site/
       i18n/
       CommonDataModel.json
   ```

2. Build the content deployment image using one of the following options:
   a. Using `pack build` command.

Run the following command to build the content deployment image:

Sample Code

```
pack build <your-container-registry>/app-html5-deployer:<image-version>
\
--path html5-deployer \
--builder paketobuildpacks/builder-jammy-base \
--buildpack sapse/application-content-deployer-buildpack:latest \
--publish
```

b. Using `containerize.yaml` command.

Run the `cds add kyma` command to generate the `containerize.yaml` file. Adapt the module definition for content deployer image to use the application content deployer buildpack:

Sample Code

```
_schema-version: '1.0'
repository: <your-container-registry>
tag: latest
modules:
- name: app-html5-deployer
    build-parameters:
    buildpack:
        type: sapse/application-content-deployer-buildpack:latest
        builder: builder-jammy-base
        path: app/html5-deployer
```

Configure Helm Chart

Procedure

1. Run the following command to generate the Helm chart folder containing the `Chart.yaml`, `values.yaml`, `values.schema.json` files:

   ```
   cds add kyma
   ```

   Note

   If you have already run this command, skip this step.

2. Remove the `portal-deployer` module section from the `values.yaml` file:

   Sample Code

   ```
   # Remove if present
   # portal-deployer:
   #   image:
   #     repository: incident-management-portalcontent
   #   resources:
   #     limits:
   #       ephemeral-storage: 1G
   #       memory: 500M
   #     requests:
   ```

```
#       ephemeral-storage: 1G
#       cpu: 500m
#       memory: 500M
#   bindings:
#     auth:
#       serviceInstanceName: xsuaa
#     html5-apps-repo-host:
#       serviceInstanceName: html5-apps-repo-host
#     portal:
#       serviceInstanceName: portal
#   env:
#     SERVICE_BINDING_ROOT: /bindings
#     BPL_VCAP_SERVICES_ENABLED: "true"
#     XSUAA_CREDENTIALS:
#       secretKeyRef:
#         name: "{{ .Release.Name }}-approuter-auth"
#         key: credentials
#     REPOHOST_CREDENTIALS:
#       secretKeyRef:
#         name: "{{ .Release.Name }}-html5-apps-deployer-html5-apps-repo"
#         key: credentials
#     PORTAL_CREDENTIALS:
#       secretKeyRef:
#         name: "{{ .Release.Name }}-portal-deployer-portal"
#         key: credentials
```

3. Add the `content-agent-service` instance definition to the `values.yaml` file:

Sample Code

```
content-agent-service:
  serviceOfferingName: content-agent
  servicePlanName: application
```

4. Add the `content-agent-service` and `portal service` bindings to the `html5-apps-deployer` section in the `values.yaml` file:

Sample Code

```
bindings:
content-agent-service:
    serviceInstanceName: content-agent-service
portal:
    serviceInstanceName: portal
```

5. Add the `content-agent-service` dependency entry to the `Chart.yaml` file:

Sample Code

```
dependencies:
- name: service-instance
    alias: content-agent-service
    version: ">0.0.0"
```

6. Run the following command to rebuild the Helm chart dependencies:

```
cds build --production
```

7. Install the Helm chart into the target Kyma namespace using the `cds up -2 k8s -n <namespace>` command.

4.2 Deploy CAP Application with SAP Application Frontend Service

Prerequisites

- You have completed the steps in the Prepare Your System for Deployment [page 5] section.
- You've completed all the steps required to create a sample CAP application using CDS CLI. See: Capire documentation.

Context

Application content deployer buildpack enables deployment of application content by accepting all referenced ZIP archives. The buildpack creates a container image using this folder, streamlining the deployment process for your application content.

Procedure

1. Initiate the CAP project.

   Sample Code

   ```
   cds init bookshop --add nodejs,tiny-sample && code bookshop
   ```

2. Run one of the following commands depending on your application type:

   `cds add vue --into catalog` (or) `cds add react --into catalog`
3. Add the app-frontend facet to the application using the following command:

   ```
   cds add app-frontend
   ```
4. Run the appropriate command based on your authentication type (IAS or XSUAA):

   `cds add ias --for production` (or) `cds add xsuaa --for production`
5. Add HANA as the production database for deployment:

   ```
   cds add hana --for production
   ```
6. Add Helm charts and `containerize.yaml` to your project for deployment, run the following command:

   ```
   cds add kyma
   ```

   This command creates the `chart` folder and the `containerize.yaml` file.
7. Modify the `chart` folder files and `containerize.yaml` as described in the following steps:

a. In `containerize.yaml`, update the html5-deployer configuration as follows:

Sample Code

```
name: <appName>-html5-deployer
   build-parameters:
     buildpack:
       type: sapse/application-content-deployer-buildpack:1.4.0
       builder: builder-jammy-base
       path: app/html5-deployer
```

In `containerize.yaml`, set the `type` field to the application content deployer buildpack.

b. In `chart/values.yaml` , add the following content-agent service configuration:

Sample Code

```
content-agent-service:
  serviceOfferingName: content-agent
  servicePlanName: application
```

8. In the `html5-apps-deployer` section of `chart/values.yaml`, add the following:

a. Add the content-agent service binding in `bindings`.

Sample Code

```
bindings:
  content-agent-service:
    serviceInstanceName: content-agent-service
```

b. Add the destination environment variable in `env` :

Sample Code

```
env:
  destinations: '[{"forwardAuthToken":true,"name":"srv-
api","url":"https://{{ .Release.Name }}-srv-{{ .Release.Namespace }}.
{{ .Values.global.domain }}"}]'
```

Note

If IAS is configured as the authentication mechanism, use the following instead:

Sample Code

```
env:
  IAS_DEPENDENCY_NAME: <appName>-ias-api
  destinations: '[{"forwardAuthToken":true,"name":"srv-
api","url":"https://{{ .Release.Name }}-srv-{{ .Release.Namespace }}.
{{ .Values.global.domain }}","IASDependencyName":"<appName>-ias-
api"}]'
```

9. In `chart/Chart.yaml`, add the following dependency:

Sample Code

```
- name: service-instance
  alias: content-agent-service
  version: ">0.0.0"
```

10. Deploy the application to your target namespace using the following command:

    `cds up -2 k8s -n <namespace>`

    This command automatically runs `cds build --production` to build production-ready artifacts and calls `helm install` to deploy the application to the target namespace.

4.3 Deploy CAP Application with SAP Build Work Zone

Prerequisites

You have completed the steps in the Prepare Your System for Deployment [page 5] section.

Context

Deploy applications built with the SAP Cloud Application Programming Model (CAP) and integrated with SAP Build Work Zone using the application content deployer buildpack. The buildpack automatically detects the `resources/` folder containing the HTML5 application ZIP archives and builds the container image for content deployment.

Procedure

Build the Content Deployment Image

1. Prepare the deployer folder.
   a. The `@sap/cds-dk` generates a folder named `html5-deployer`. Whereas the older version generates `ui-resources` name.

      Note

      Earlier versions of `@sap/cds-dk` generate a folder named `ui-resources`. Adjust the folder path accordingly.

   b. The `resources/` subfolder contains the HTML5 application ZIP archives.

c. Optionally, remove the folder-level `package.json` file.

2. Build the OCI image using one of the following options:

Using the `pack` CLI:

a. Run the following command to build the container image:

Sample Code

```
pack build <your-container-registry>/app-html5-deployer:<image-version>
\
  --path html5-deployer \
  --builder paketobuildpacks/builder-jammy-base \
  --buildpack sapse/application-content-deployer-buildpack:latest \
  --publish
```

> ### Note
>
> In the latest `@sap/cds-dk` versions, the folder is named `html5-deployer`. Adjust the `--path` accordingly.

Using `containerize.yaml`

a. Run the `cds add kyma` command to generate the `containerize.yaml` file. Adapt the module definition for the content deployer image to use the application content deployer buildpack.

Sample Code

```
schema-version: '1.0'
repository: <your-container-registry>
tag: latest
modules:
- name: app-html5-deployer
  build-parameters:
    buildpack:
      type: sapse/application-content-deployer-buildpack:latest
      builder: builder-jammy-base
      path: app/html5-deployer
```

Configure the Helm Chart

Procedure

1. Generate the Helm chart folder.

   Run the following command to generate the Helm chart files,`Chart.yaml`, `values.yaml`, and `values.schema.json`:

   ```
   cds add kyma
   ```

> ### Note
>
> If you have already run this command, skip this step.

2. Add the `content-agent-service` instance definition to `values.yaml`:

   Sample Code

   ```
   content-agent-service:
     serviceOfferingName: content-agent
     servicePlanName: application
   ```

3. Add the `content-agent-servicec` binding to the `html5-apps-deployer` section in `values.yaml`:

   Sample Code

   ```
   bindings:
   content-agent-service:
       serviceInstanceName: content-agent-service
   ```

   Note

   The `content-agent-service` binding is required for The application content deployer buildpack to deploy content to the target content runtimes

4. Add the `content-agent-service` dependency entry to `Chart.yaml` file:

   Sample Code

   ```
   dependencies:
   - name: service-instance
       alias: content-agent-service
       version: ">0.0.0"
   ```

5. Rebuild the Helm artifacts for production:

   ```
   cds build --production
   ```

6. Deploy to the Kyma namespace.

   Run the following command to install the Helm chart into the target Kyma namespace:

   Sample Code

   ```
   cds up -2 k8s -n <namespace>
   ```

   The application content deployer buildpack automatically creates destinations for the following service instances:

   - HTML5 Application Repository service (mandatory):
     Creates a destination for the HTML5 Application Repository service instance.
   - SAP Authorization and Trust Management (XSUAA) service (optional):
     Creates a destination for the XSUAA service instance if both the XSUAA and Destination service instances are configured in `values.yaml`.
   - Business services (optional):
     Creates a destination for the business service instance if the service instance is configured in `values.yaml`.
   - Backend destinations defined using `BACKEND_DESTINATIONS` (optional):

Creates destinations for cloud or on-premise backend applications defined using `BACKEND_DESTINATIONS`.

Additional Configuration

The following environment variables are also supported in the `html5-apps-deployer` section of `values.yaml`:

- `sap.cloud.service`:
  Set to the `sap.cloud.service` value from the HTML5 application `manifest.json` file.
  The application content deployer buildpack automatically creates subaccount-level destination configurations for the HTML5 Application Repository service, XSUAA, business services, and backend destinations.
- `destinations`:
  Uploads destination configurations to the SAP HTML5 Application Repository service.
- `IAS_DEPENDENCY_NAME`:
  Enables app-to-app navigation using Identity Authentication token exchange.
- `HTML5Runtime_enabled`:
  Enables consumption through the SAP-Managed Application Router without XSUAA or IAS token exchange.

4.4 Deploy CAP Application with SAP Build Work Zone Using the Common Data Model

Use the Common Data Model (CDM) approach to integrate CAP application with SAP Build Work Zone.

Prerequisites

- You've created the CDM configuration.
  For example, follow steps as described in step 3 of the Common Data Model section in Integrate with SAP BTP, Kymaruntime
- You have completed all the Prepare Your System for Deployment [page 5] section.
- A `cdm.json` file containing the required CDM definition has been created and added to the `resources/` folder.

> **Note**
>
> To integrate a business solution with SAP Build Work Zone, Standard Edition or SAP Build Work Zone, Advanced Edition, define the business solution as a content provider by including a `cdm.json` file in the `resources/` folder. The `cdm.json` file contains the Common Data Model (CDM) definition for the business solution.

Context

The Application Content Deployer Buildpack detects the `resources/` folder containing HTML5 application ZIP archives and the `cdm.json` file, and builds the content deployment image. The `cdm.json` file is included during the content deployment build process and configures the business solution as a content provider.

Build the Content Deployment Image

Procedure

1. Add the `cdm.json` file to the `html5-deployer/resources/` folder where the HTML5 application ZIP archives are stored.
2. Build the OCI image using one of the following options:
   a. Using the `pack` CLI:

      Run the following command to build the content deployment image:

      Sample Code

      ```
      pack build <your-container-registry>/app-html5-deployer:<image-version>
      \
        --path html5-deployer \
        --builder paketobuildpacks/builder-jammy-base \
        --buildpack sapse/application-content-deployer-buildpack:latest \
        --publish
      ```

   b. Using `containerize.yaml` to create the image

      Run the `cds add kyma` command to generate the `containerize.yaml` file, and then adapt the module definition for the content deployer image to use the Application Content Deployer Buildpack:

      Sample Code

      ```
      _schema-version: '1.0'
      repository: <your-container-registry>
      tag: latest
      modules:
      - name: app-html5-deployer
          build-parameters:
          buildpack:
              type: sapse/application-content-deployer-buildpack: latest
              builder: builder-jammy-base
              path: app/html5-deployer
      ```

Configure the Helm Chart

Procedure

1. Generate Helm chart folder:

   ```
   cds add kyma
   ```

   > ⓘ Note
   >
   > If you have already run this command, skip this step.

2. Add the `content-agent-service` instance definition in `values.yaml`:

   > {·} Sample Code

   ```
   content-agent-service:
     serviceOffering Name: content-agent
     servicePlan Name: application
   ```

3. Bind the `content-agent-service` in the html5-apps-deployer bindings section in `values.yaml` file:

   > {·} Sample Code

   ```
   bindings:
   content-agent-service:
       serviceInstanceName: content-agent-service
   ```

   > ⓘ Note
   >
   > The `content-agent-service` binding is required for the application content deployer to deploy content to target content runtimes.

4. Configure backend destinations in the `values.yaml` File.

   Add the backend destination configuration to the `env` section of the `html5-apps-deployer`:

   > {·} Sample Code

   ```
   env:
     destinations: '[{"forwardAuthToken":true,"name":"srv-api","url":"https://
   <deployment-name>-srv-<namespace>.<cluster-domain>"}]'
   ```

5. Add `content-agent-service` dependency in `Chart.yaml` file:

   > {·} Sample Code

   ```
   dependencies:
   - name: service-instance
       alias: content-agent-service
       version: ">0.0.0"
   ```

6. Rebuild Helm artifacts for production:

   ```
   cds build --production
   ```

7. Install the Helm chart into the target Kyma namespace.

   Run the following command to install the Helm chart into the target Kyma namespace:

   Sample Code

   ```
   cds up -2 k8s -n <namespace>
   ```

4.5 Deploy Standalone UI Application with Application Frontend Service

Prerequisites

You've already completed all the steps in Prepare Your System for Deployment [page 5].

Context

Deploy a standalone UI application to SAP BTP, Kyma runtime, using the Application Frontend service for UI hosting and the application content deployer buildpack for content deployment.

This is a Single Target Application (STA), a standalone UI frontend with no CAP backend or HANA database. The UI application is hosted on the Application Frontend service.

Procedure

1. **Create the Helm Chart Directory**

   ```
   mkdir chart
   ```

   Note

   This scenario is not a CAP project. Do not use `cds add helm`, this either fails or produce an incomplete chart that assumes a CAP backend exists. Create the chart manually instead.

2. Configure chart/Chart.yaml file.

   Add the following dependencies to `chart/Chart.yaml`:

   Sample Code

   ```
   dependencies:
   ```

```
  - name: service-instance
    alias: app-front
    version: ">0.0.0"
    repository: "@unified-runtime"
  - name: content-deployment
    alias: html5-apps-deployer
    version: ">0.0.0"
    repository: "@unified-runtime"
  - name: service-instance
    alias: xsuaa
    version: ">0.0.0"
    repository: "@unified-runtime"
  - name: service-instance
    alias: content-agent
    version: ">0.0.0"
    repository: "@unified-runtime"
```

> **Note**
>
> `repository: "@unified-runtime"` is required on every entry. The `helm dep update` command uses it to fetch the subcharts.

> **Note**
>
> xsuaa is optional. If your application does not require authentication, remove the xsuaa entry from `Chart.yaml`.

3. Configure chart/values.yaml file.

   Replace your entire chart/values.yaml with the following. Replace the <your-*> placeholders with your values:

Sample Code

```
global:
  domain: <your-kyma-domain>
  imagePullSecret:
    name: docker-registry
  image:
    registry: <your-registry>
    tag: latest
xsuaa:
  serviceOfferingName: xsuaa
  servicePlanName: application
  parameters:
    tenant-mode: dedicated
    oauth2-configuration:
      redirect-uris:
        - https://*.{{ tpl .Values.global.domain . }}/**
    xsappname: <your-app-name>-{{ .Release.Namespace }}
app-front:
  serviceOfferingName: app-front
  servicePlanName: developer
content-agent:
  serviceOfferingName: content-agent
  servicePlanName: application
html5-apps-deployer:
  image:
    repository: <your-app-name>-html5-deployer
  env:
    destinations: '[{"name":"ui5","url":"https://ui5.sap.com"}]'
  bindings:
    xsuaa:  # Optional – remove if xsuaa is not used
```

```
      serviceInstanceName: xsuaa
    app-front:
      serviceInstanceName: app-front
    content-agent:
      serviceInstanceName: content-agent
  resources:
    limits:
      cpu: 2000m
      memory: 1Gi
    requests:
      cpu: 1000m
      memory: 1Gi
```

> **Note**
>
> xsuaa is optional. If your application does not require authentication, remove it.

> **Note**
>
> The `destinations` entry is required only for UI5-based applications. Remove it if your application does not use UI5.

4. Configure xs-app.json

   Add the following route to the routes array in your xs-app.json:

> **Sample Code**

```
{
  "welcomeFile": "/index.html",
  "authenticationMethod": "route",
  "routes": [
    {
      "source": "^/resources/(.*)$",
      "target": "/resources/$1",
      "authenticationType": "none",
      "destination": "ui5"
    },
    {
      "source": "^/test-resources/(.*)$",
      "target": "/test-resources/$1",
      "authenticationType": "none",
      "destination": "ui5"
    },
    {
      "source": "^(.*)$",
      "target": "$1",
      "service": "app-front",
      "authenticationType": "xsuaa"
    }
  ]
}
```

> **Note**
>
> The `resources` and `test-resources` routes are required only for UI5-based applications. They forward UI5 framework requests to `https://ui5.sap.com` via the `ui5` destination. If your application does not use UI5, remove these routes.

> ### Note
>
> For the catch-all route, set *authenticationType* based on your project setup. Use *xsuaa* if your app requires authentication, or use *none* if it is publicly accessible.

5. Create the resources/ Folder

   The buildpack looks for a resources/ folder containing your app ZIP file. Create it at the project root:

   ```
   mkdir resources
   ```

6. Build your UI5 application using your build tool and place the ZIP file in the `resources/` folder at the project root. The procedure varies depending on whether your project uses a build tool:

   Using a build tool, for example, UI5-tooling

   > ### Sample Code
   >
   > ```
   > npm install && npm run build
   > cd dist && zip -r ../resources/<your-app>.zip . && cd ..
   > ```

   If your application has no build tool, zip the source directly:

   > ### Sample Code
   >
   > ```
   > cd webapp && zip -r ../resources/<your-app>.zip . && cd ..
   > ```

   > ### Note
   >
   > `xs-app.json` must be at the root of the ZIP file. Verify before proceeding.

7. Build the Deployer OCI Image

   The buildpack detects the ZIP file in `resources/` folder, bundles it with the content-agent deployer binary, and produces a single OCI image. When this image runs as a Kubernetes Job during `helm install`, it uploads the ZIP file to the Application Frontend service via the content agent.

   > ### Sample Code
   >
   > ```
   > pack build <your-registry>/<your-app-name>-html5-deployer:latest \
   >   --buildpack sapse/application-content-deployer-buildpack:latest \
   >   --builder proxy-unified-runtime-dmz.int.repositories.cloud.sap/builder/
   > noble:latest \
   >   --publish
   > ```

8. Fetch Helm Subchart Dependencies

   Run this command after `chart/Chart.yaml` and `chart/values.yaml` are configured. It downloads the required subcharts into `chart/charts/` based on the dependencies in `Chart.yaml`.

   ```
   helm dep update chart/
   ```

9. Deploy to SAP BTP, Kyma Runtime

   ```
   helm upgrade --install <release-name> ./chart -n <your-namespace>
         --wait
   ```

The standalone UI application is deployed to SAP BTP, Kyma runtime, and hosted on the Application Frontend service.

4.6 Deploy Single Target Application with ZIP Archives

Deploy application content to a single target content runtime using the application content deployer buildpack with pre-packaged ZIP archives and a JSON descriptor.

Context

The application content deployer buildpack enables deployment of application content by accepting a folder containing a JSON descriptor (`metadata.json`) and all referenced ZIP archives. The buildpack creates a container image using this folder, streamlining the deployment process for your application content.

Procedure

1. Build the content folder

   Prepare a content folder containing your build artifacts that are to be deployed to respective content runtimes.

   Example content folder layout:

   ```
   app-deployer/
   ├── resources/
   │   ├── ui-bundle.zip
   │   └── cdm.json
   ├── portal-site.zip
   └── metadata.json
   ```

2. Create the metadata.json descriptor file

   Create a descriptor (`metadata.json`) file that defines your content deployments. This file includes:

   - identifier (optional): A unique identifier for the content deployment. Required if `dependsOn` is used.
   - provider: The content provider (for example, `html5-apps-repo`, `portal`, `destination`, `app-front`).
   - content: The content to be deployed (type and data path).
   - requiredServices (optional): Array of required services.
   - dependsOn (optional): Array of deployment identifiers this deployment depends on.

   The descriptor file is an array of deployments objects.

Example metadata.json:

For the content folder layout shown above, the corresponding `metadata.json` is:

```
{
  "deployments": [
    {
      "identifier": "html5-content",
      "provider": "html5-apps-repo",
      "content": { "type": "file", "data": "./resources" }
    },
    {
      "provider": "portal",
      "content": { "type": "file", "data": "./portal-site.zip" },
      "requiredServices": ["html5-apps-repo"],
      "dependsOn": ["html5-content"]
    }
  ]
}
```

JSON Schema Validator:

A JSON schema validator is provided for the `metadata.json` file. You can use this schema to validate the `metadata.json` file before deploying it.

View JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Content Deployment",
  "description": "A JSON schema for content deployment metadata.",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "parameters": {
      "$ref": "#/definitions/parameters"
    },
    "deployments": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/contentDeploymentMetadata"
      },
      "description": "A list of deployment metadata entries."
    }
  },
  "required": [
    "deployments"
  ],
  "definitions": {
    "identifier": {
      "type": "string",
      "format": "uuid",
      "description": "A unique identifier."
    },
    "parameters": {
      "type": "object",
      "description": "A map of parameters used to configure the deployment
process.",
      "additionalProperties": true
    },
    "contentDeploymentMetadata": {
      "title": "Content Deployment Metadata",
      "description": "Metadata describing a content deployment.",
      "type": "object",
      "additionalProperties": false,
      "properties": {
```

```
        "identifier": {
          "$ref": "#/definitions/identifier"
        },
        "provider": {
          "$ref": "#/definitions/contentProviders"
        },
        "content": {
          "type": "object",
          "description": "An object that contains information about the
content.",
          "additionalProperties": false,
          "properties": {
            "type": {
              "$ref": "#/definitions/contentTypes"
            },
            "data": {
              "type": "string",
              "description": "A relative or absolute path to the file/
folder.",
              "pattern": "^(\\/|\\.\\/|\\.\\.\\/).*"
            }
          },
          "required": [
            "type",
            "data"
          ]
        },
        "parameters": {
          "$ref": "#/definitions/parameters"
        },
        "dependsOn": {
          "type": "array",
          "description": "An array of identifiers that should be deployed
before the current item.",
          "items": {
            "$ref": "#/definitions/identifier"
          }
        },
        "requiredServices": {
          "type": "array",
          "description": "An array of strings that define the required
services for the content deployment.",
          "items": {
            "$ref": "#/definitions/contentProviders"
          }
        }
      },
      "required": [
        "provider",
        "content"
      ],
      "allOf": [
        {
          "if": {
            "required": [
              "dependsOn"
            ]
          },
          "then": {
            "required": [
              "identifier"
            ]
          }
        }
      ]
    },
    "contentProviders": {
      "type": "string",
```

```
      "enum": [
        "apimanagement-apiportal",
        "apimanagement-devportal",
        "it-rt",
        "workflowmanagement",
        "workflow",
        "business-rules",
        "mobile_application",
        "process_visibility",
        "html5-apps-repo",
        "destination",
        "job-service",
        "business-logging",
        "portal",
        "app-front"
      ],
      "description": "The list of supported content providers."
    },
    "contentTypes": {
      "type": "string",
      "enum": [
        "file"
      ],
      "description": "The list of supported content types."
    }
  }
}
```

3. Build and publish the OCI image

   Build the OCI image using the `pack` command with the application content deployer buildpack. Provide the path to `metadata.json` as an environment variable.

   The `BP_JSON_PATH` environment variable is required so the buildpack can locate and package the descriptor.

   Pack command:

```
pack build --publish --trust-builder \
  --builder <builder-of-your-choice> \
  --buildpack sapse/application-content-deployer-buildpack:latest \
  --path "." \
  --env BP_JSON_PATH=metadata.json \
  <image-registry>/<image-name>:<image-tag>
```

> ### Note
>
> Set the `--path` argument based on your current terminal directory. Use `--path "."` if you are already inside the content folder; otherwise, provide the relative path to it.

4. Build and deploy the Helm charts
   a. Generate the Helm charts

      You can generate the Helm charts using the CDS CLI or manually create Helm charts:

      - Using CDS CLI: `cds add helm`
      - Manually: `helm create <app-name>`

   b. Configure service bindings

      Provide service bindings for the target deployment services. They are required for the buildpack to complete the Kubernetes job successfully.

   c. Deploy the Helm charts

Deploy your Helm chart that references the OCI image built above using the `helm install` command.

Results

The application content is deployed to the specified content runtime. The buildpack creates a Kubernetes job that executes the deployment based on your `metadata.json` configuration.

5 Troubleshooting

Common Issues and Solutions

MTAR not detected by the buildpack

Ensure the MTAR file is in the mta_archives/ directory, or set BP_MTAR_PATH to its relative path. Verify the file extension is .mtar.

Image build fails

Verify the buildpack reference is correct and accessible from your build environment. Check that your container registry credentials are configured and that the --publish flag is used only when credentials are available.

Service credentials not found at runtime

Ensure all required service bindings are created in your SAP BTP subaccount and that the BTP Service Operator has mounted the secrets into the Kubernetes namespace. Check the SERVICE_BINDING_ROOT environment variable.

Destinations not working (Application Frontend / HTML5)

Verify the destination configuration in the Helm values (env.destinations). Ensure the URLs follow the correct Kyma pattern and that all required service instances are bound to the application-content-deployment module.

Application not accessible after deployment

Check that the Application Frontend Service is properly subscribed and that the Application Frontend Developer and Viewer roles are assigned. Review deployer pod logs for deployment errors.

ZIP file not found during deployment

Confirm the ZIP file is in the correct location (resources/ folder).

Authentication errors during deployment

Verify that the XSUAA service instance is correctly bound to the deployer module and that the credentials are properly mounted.

Error Code Reference

The Application Content Deployer exits with a non-zero code when an error occurs. The following table lists all error codes and their meanings. To capture more detail, re-run the deployer with the --verbose flag.

General Errors

Error Code	Description
1	Unexpected error occurred
2	Missing arguments
3	Error fetching token
4	Size exceeded error

Not Found Errors

Error Code	Description
101	Environment not found
102	Folder not found
103	File not found
104	Service not found
105	Some details for service are missing
106	No deployments available to deploy

Read Errors

Error Code	Description
111	Error reading folder
112	Error reading file

Parsing Errors

Error Code	Description
121	Error parsing folder
122	Error parsing file
123	Error parsing credentials
124	Error parsing environment
125	Error parsing arguments
126	Illegal arguments
127	Invalid type
128	Field missing
129	Content not supported
130	Duplicate deployments found

Creation Errors

Error Code	Description
131	Error creating service
132	Error creating folder
133	Error creating file
134	Error creating zip
135	Error creating connector for service

Mismatch Errors

Error Code	Description
201	Expected service not found
202	Expected service plan not found
203	Expected content data not found
204	Expected deployment not found

Verification Errors

Error Code	Description
211	Error verifying module
212	Error verifying content deployment order
213	Error verifying archive files
214	Error verifying JSON file

Content Agent API Errors

Error Code	Description
221	Error uploading content
222	Error creating deployment resource
223	Error deleting deployment resources
224	Error fetching deploy resource status
225	Error fetching deploy resource logs
226	Review the provided content or submit a support ticket with --verbose logs

Debugging Tips

- Review the full Kubernetes Job pod log: kubectl logs <pod-name> -n <namespace>
- Check that all referenced ZIP files and folders exist at the expected paths.

- Verify service bindings and credential mounts before initiating deployment.
- Test network connectivity from the deployer container to the Content Agent Service endpoint.
- Confirm the buildpack version referenced in your build command is available and accessible.

Getting Support

If you encounter an issue that cannot be resolved using the troubleshooting guidance in this document, open a support ticket through the SAP Support Portal.

You can report an incident or error through the SAP Support Portal.

Use the following component for your incident:

Component Name	Component Details
BC-CP-LCM-CAS	Subject prefix: [Application Content Deployer Buildpack] <short description>

> **Note**
>
> When submitting a support ticket, include the deployer pod logs captured with the --verbose flag, the buildpack version used, and your mta.yaml (sanitised of credentials) to help the support team investigate faster.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon  : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon  : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap