Deploy to Kyma

You can run your CAP application in the SAP BTP Kyma Runtime , the SAP-managed offering for the Kyma project .

▶ *This guide is available for Node.js and Java.*

Table of Contents

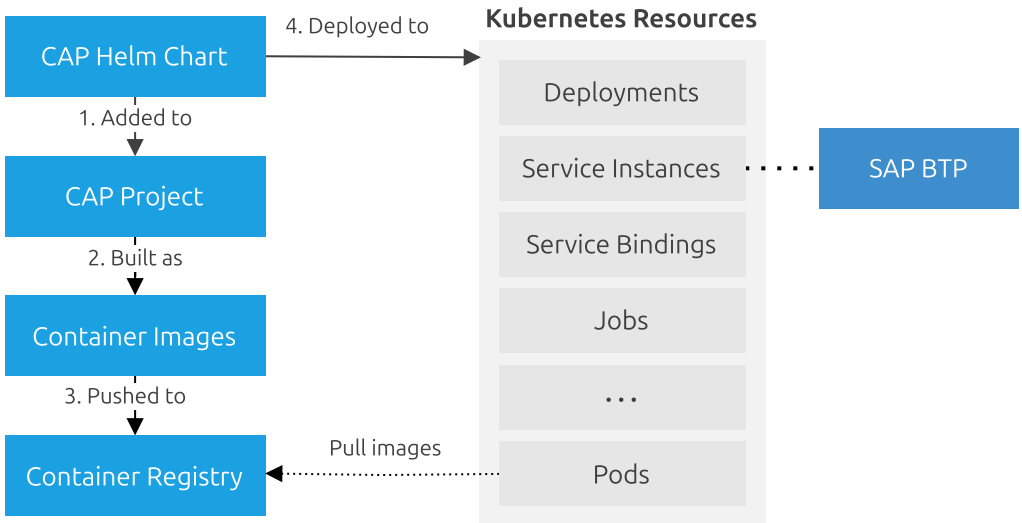
Overview

Kyma is a Kubernetes-based runtime for deploying and managing containerized applications. Applications are packaged as container images—typically Docker images—and their deployment and operations are defined using Kubernetes resource configurations.

Deploying apps on the SAP BTP Kyma Runtime requires two main artifact types:

1. **Container Images** – Your application packaged in a container
2. **Kubernetes Resources** – Configurations for deployment and scaling

The following diagram illustrates the deployment workflow:



Prerequisites

- Use a Kyma-enabled Trial Account or purchase a Kyma cluster from SAP
- You need a Container Image Registry
- Get the required SAP BTP service entitlements
- Install Docker Desktop or Docker for Linux
- Download and install the following command line tools:
  - `kubectl` command line client for Kubernetes
  - `pack` command line tool
  - `helm` command line tool
- Make sure your SAP HANA Cloud is mapped to your namespace

- Ensure SAP HANA Cloud is accessible from your Kyma cluster by configuring trusted source IPs

Configure Kubernetes

Download the Kubernetes configuration from SAP BTP and move it to *$HOME/.kube/config*.

↳ *Learn more in the SAP BTP Kyma documentation*

Get Access to a Container Registry

SAP BTP doesn't provide a container image registry (or container repository), but you can choose from offerings of hosted open source and private container image registries, as well as solutions that can be run on premise or in your own cloud infrastructure.

> **Ensure network access**
>
> Verify the Kubernetes cluster has network access to the container registry, especially if hosted behind a VPN or within a restricted network environment.

Set Up Your Cluster for a Private Container Registry

To use a docker image from a private repository, you need to create an image pull secret and configure this secret for your containers.

> **Interactive setup**
>
> If a pull secret does not exist for your namespace when deploying your application, the CLI will prompt you to set up the pull secret interactively.

> **Assign limited permissions to the technical user**
>
> For this secret, use a technical user with read-only permissions. This limits the risk, as anyone with access to the Kubernetes cluster could retrieve the password from the secret and potentially modify or publish images to the registry.

Deploy to Kyma

Let's start with a new sample project and prepare it for production using an SAP HANA database and XSUAA for authentication:

```sh
cds init bookshop --add sample && cd bookshop
cds add hana,xsuaa
```

User Interfaces beta

If you need a UI, you can also add SAP Build Work Zone support:

```sh
cds add workzone
```

> This is currently only supported for single-tenant scenarios.

Add CAP Helm Charts

CAP provides a configurable Helm chart for Node.js and Java applications, which can be added like so:

```sh
cds add kyma
```

> You will be asked to provide a Kyma cluster domain and your container registry name.

▶ *Running `cds build` now creates a gen/chart folder*

Build and Deploy

First, ensure the Docker daemon is running, for example by starting Docker Desktop.

You can now quickly deploy the application like so:

```sh
cds up -2 k8s [ -n <your-namespace> ]
```

▶ *Essentially, this automates the following steps...*

This command uses checksums to detect changes in your code. If any modifications are found, it automatically triggers a rebuild. The checksums are reflected in the Docker image tags.

This process can take a few minutes to complete and logs output like this:

```log
[…]
The release bookshop is installed in namespace [namespace].

Your services are available at:
  [workload] - https://bookshop-[workload]-[namespace].[configured-domain]
[…]
```

You can use this URL to access the approuter as the entry point of your application.

For **multitenant applications**, you have to subscribe a tenant first. The application is accessible via a tenant-specific URL after subscription.

> **SaaS Extensibility**
>
> Share the above App-Router URL with SaaS consumers for logging in as extension developers using `cds login` or other extensibility-related commands .

Next Up...

You would then set up your CI/CD for automating deployments, for example after merging pull requests.

Deep Dives

Configure Image Repository

Specify the repository where you want to push the images:

containerize.yaml

```yaml
...
repository: <your-container-registry>
```

Customize Helm Chart

About CAP Helm Charts

The following files are added to a *chart* folder by executing `cds add kyma` :

```zsh
chart/
├── values.yaml         # Default configuration of the chart
├── Chart.yaml          # Chart metadata
└── values.schema.json  # JSON Schema for values.yaml file
```

↳ *Learn more about values.yaml.*

↳ *Learn more about Chart.yaml.*

In addition, a `cds build` also puts some files to the *gen/chart* folder:

```zsh
chart/
├── templates/
│   ├── NOTES.txt # Message printed after Helm upgrade
│   ├── *.tpl     # Template libraries used in template resources
│   ├── *.yaml    # Template files for Kubernetes resources
```

↳ *Learn how to create a Helm chart from scratch.*

Configure

You can change the configuration of CAP Helm charts by editing the *chart/values.yaml* file. The `helm` CLI also offers you other options to overwrite settings from *chart/values.yaml* file:

- Overwrite properties using the `--set` parameter.
- Overwrite properties from a YAML or JSON file using the `-f` parameter.

Multiple deployment types

> It is recommended to do the main configuration in the *chart/values.yaml* file and have additional YAML files for specific deployment types (dev, test, productive) and targets.

Global Properties

values.yaml

```yaml
# Secret name to access container registry, only for private registries
imagePullSecret:
  name: <docker-secret>

# Kubernetes cluster ingress domain (used for application URLs)
domain: <cluster-domain>

# Container image registry where to pull the image from
image:
  registry: <registry-server>
```

Deployment Properties

The following properties are available for the `srv` key:

values.yaml

```yaml
srv:
  # [Service bindings](#configuration-options-for-service-bindings)
  bindings:

  # Kubernetes container resources
  # https://kubernetes.io/docs/concepts/configuration/manage-resources-contai
  resources:

  # Map of additional env variables
  env:
    MY_ENV_VAR: 1

  # Kubernetes Liveness, Readiness and Startup Probes
  # https://kubernetes.io/docs/tasks/configure-pod-container/configure-livene
  health:
    liveness:
      path: <endpoint>
```

```
    readiness:
      path: <endpoint>
    startupTimeout: <seconds>

  # Container image
  image:
```

> You can explore more configuration options in the subchart's directory *gen/chart/charts/web-application*.

SAP BTP Services

You can find a list of SAP BTP services in the Discovery Center . To find out if a service is supported in the Kyma and Kubernetes environment, go to the **Service Marketplace** of your Subaccount in the SAP BTP Cockpit and select Kyma or Kubernetes in the environment filter.

You can find information about planned SAP BTP, Kyma Runtime features in the product road map .

Built-in SAP BTP Services

The Helm chart supports creating service instances for commonly used services. Services are pre-populated in *chart/values.yaml* based on the used services in the `requires` section of the CAP configuration.

You can use the following services in your configuration:

values.yaml

yaml

```yaml
xsuaa:
  parameters:
    xsappname: <name>
    HTML5Runtime_enabled: true # for SAP Launchpad service
event-mesh: …
connectivity: …
destination: …
html5-apps-repo-host: …
hana: …
```

```
service-manager: …
saas-registry: …
```

Arbitrary BTP Services

These are the steps to create and bind to an arbitrary service, using the binding of the feature toggle service to the CAP application as an example:

1. In the *chart/Chart.yaml* file, add an entry to the `dependencies` array.

```yaml
dependencies:
  ...
  - name: service-instance
    alias: feature-flags
    version: 0.1.0
```

2. Add service configuration and binding in *chart/values.yaml*:

```yaml
feature-flags:
  serviceOfferingName: feature-flags
  servicePlanName: lite
...
srv:
   bindings:
     feature-flags:
        serviceInstanceName: feature-flags
```

> The `alias` property in `dependencies` must match the property added in the root of *chart/values.yaml* and the value of `serviceInstanceName` in the binding.

▶ *Additional requirements for the SAP Connectivity service...*

Finally, you should see a success message as follows:

```sh
kyma.operator.kyma-project.io/default edited
```

↳ *Learn more about adding modules from the Kyma Dashboard.*

Configuration Options for Services

Services have the following configuration options:

values.yaml

```yaml
### Required ###
serviceOfferingName: my-service
servicePlanName: my-plan

### Optional ###

# Use instead of generated nname
fullNameOverride: <use instead of the generated name>

# Name for service instance in SAP BTP
externalName: <name for service instance in SAP BTP>

# List of tags describing service,
# copied to ServiceBinding secret in a 'tags' key
customTags:
  - foo
  - bar

# Some services support additional configuration,
# as found in the respective service offering
parameters:
  key: val
jsonParameters: {}

# List of secrets from which parameters are populated
parametersFrom:
  - secretKeyRef:
      name: my-secret
      key: secret-parameter
```

> You can explore more configuration options in the subchart's directory *gen/chart/charts/service-instance*.

Configuration Options for Service Bindings

values.yaml

```yaml
<service name>:
  # Exactly one of these must be specified
  serviceInstanceName: my-service # within Helm chart
```

```yaml
  serviceInstanceFullname: my-service-full-name # using absolute name
  # Additional parameters
  parameters:
    key: val
```

Configuration Options for Container Images

values.yaml

```yaml
repository: my-repo.docker.io # container repo name
tag: latest # optional container image version tag
```

HTML5 Applications

values.yaml

```yaml
html5-apps-deployer:
  image:
  bindings:
  resources:
  env:
    # Name of your business service (unique per subaccount)
    SAP_CLOUD_SERVICE: <service-name>
```

Backend Destinations

Backend destinations maybe required for HTML5 applications or for App Router deployment. They can be configured using `backendDestinations`.

If you want to add an external destination, you can do so by providing the `external` property like this:

values.yaml

```yaml
...
srv: # Key is the target service, e.g. 'srv'
  backendDestinations:
    srv-api:
      service: srv
+   ui5:
```

```
+       external: true
+       name: ui5
+       Type: HTTP
+       proxyType: Internet
+       url: https://ui5.sap.com
+       Authentication: NoAuthentication
```

> Our Helm chart will remove the `external` key and add the rest of the keys as-is to the environment variable.

Modify

Modifying the Helm chart allows you to customize it to your needs. However, this has consequences if you want to update with the latest changes from the CAP template.

You can run `cds add kyma` again to update your Helm chart. It has the following behavior for modified files:

1. Your changes of the *chart/values.yaml* and *chart/Chart.yaml* will not be modified. Only new or missing properties will be added by `cds add kyma` .
2. To modify any of the generated files such as templates or subcharts, copy the files from *gen/chart* folder and place it in the same level inside the *chart* folder. After the next `cds build` executions the generated chart will have the modified files.
3. If you want to have some custom files such as templates or subcharts, you can place them in the *chart* folder at the same level where you want them to be in *gen/chart* folder. They will be copied as is.

Extend

Instead of modifying consider extending the CAP Helm chart. Just make sure adding new files to the Helm chart does not conflict with `cds add kyma` .

> **Consider Kustomize**
>
> A modification-free approach to change files is to use Kustomize as a post-processor for your Helm chart. This might be usable for small changes if you don't want to branch-out from the generated `cds add kyma` content.

Services from Cloud Foundry

To bind service instances created on Cloud Foundry (CF) to a workload (*srv* , *hana-deployer* , *html5-deployer* , *approuter* or *sidecar*) in the Kyma environment, do the following:

1. Create a secret with credentials from the service key of that instance.
2. Use the *fromSecret* property inside the *bindings* key of the workload.

For example, if you want to use an *hdi-shared* instance created on CF:

1. Create a Kubernetes secret with service key credentials from CF
2. Add additional properties to the Kubernetes secret:

yaml

```yaml
stringData:
  # <…>
  .metadata: |
    {
      "credentialProperties":
        [
          { "name": "certificate", "format": "text"},
          { "name": "database_id", "format": "text"},
          { "name": "driver", "format": "text"},
          { "name": "hdi_password", "format": "text"},
          { "name": "hdi_user", "format": "text"},
          { "name": "host", "format": "text"},
          { "name": "password", "format": "text"},
          { "name": "port", "format": "text"},
          { "name": "schema", "format": "text"},
          { "name": "url", "format": "text"},
          { "name": "user", "format": "text"}
        ],
      "metaDataProperties":
        [
          { "name": "plan", "format": "text" },
          { "name": "label", "format": "text" },
          { "name": "type", "format": "text" },
          { "name": "tags", "format": "json" }
        ]
    }
  type: hana
  label: hana
```

```
    plan: hdi-shared
    tags: '[ "hana", "database", "relational" ]'
```

> Update the values of the properties accordingly.

3 Change *`serviceInstanceName`* to *`fromSecret`* for each workload with that service instance in *`bindings`* in *chart/values.yaml*:

yaml

```
  …
  srv:
    bindings:
      db:
-         serviceInstanceName: ##
+         fromSecret: <your secret> ##
  hana-deployer:
    bindings:
      hana:
-       serviceInstanceName: ##
+       fromSecret: <your secret> ##
```

4 Delete *`hana`* in *chart/values.yaml*:

yaml

```
  …
- hana: ##
-   serviceOfferingName: hana ##
-   servicePlanName: hdi-shared ##
  …
```

5 Make the following changes to *chart/Chart.yaml*:

yaml

```
  …
  dependencies:
    …
-   - name: service-instance ##
-     alias: hana ##
-     version: ">0.0.0" ##
    …
```

Cloud Native Buildpacks

Cloud Native Buildpacks provide advantages like embracing best practices and secure standards such as:

- Resulting images use an unprivileged user
- Builds are reproducible
- Software Bill of Materials (SBoM) baked into the image
- Auto-detection of base images

Additionally Cloud Native Buildpacks can be easily plugged together to fulfill more complex requirements. For example the ca-certificates enables adding additional certificates to the system trust-store at build and runtime. When using Cloud Native Buildpacks you can continuously benefit from best practices coming from the community without any changes required.

↳ *Learn more about Cloud Native Buildpacks Concepts.*