Creatio Academy

Application components

Static content bundling service

Version 8.0

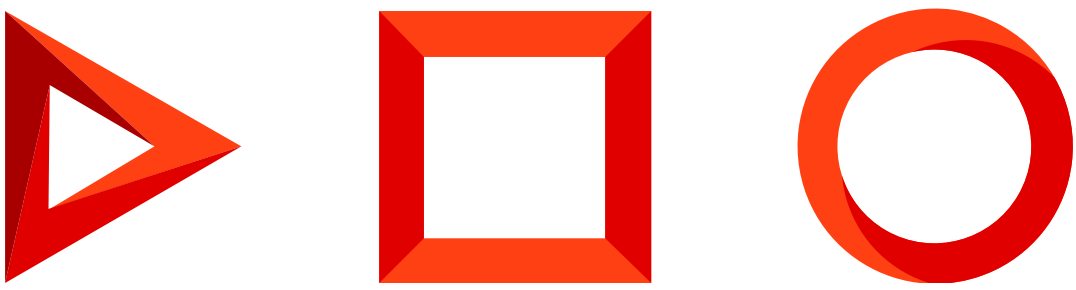


Table of Contents

Static content bundling service 4

ContentService 5

ContentWatcher 9

InfluxDb metrics 13

Deploy the static content bundling service 14

Static content bundling service

Advanced

Static content includes *.js-files and *.css-files that are located in the Creatio file system and are required to display the Creatio UI in the browser. *.js files might contain client module code or other JavaScript data. Static content has several distinct features covered in a separate article: Packages file content. By default, static content is enabled in Creatio, which improves performance.

If the browser must initially load a large amount of static content, this can significantly affect the loading time.

Ways to optimize the page loading time:

- **Minification**. Reduces the size of *.css, *.js, and *.html files. This implies removal of commented out code as well as superfluous line breaks, leading and trailing spaces. This reduces the size of the original file by 10-20%.
- **Bundling**, i. e., generation of bundle files. Optimizes the performance by combining the static files of the same type into a single bundle file. Reduces the number of requests.

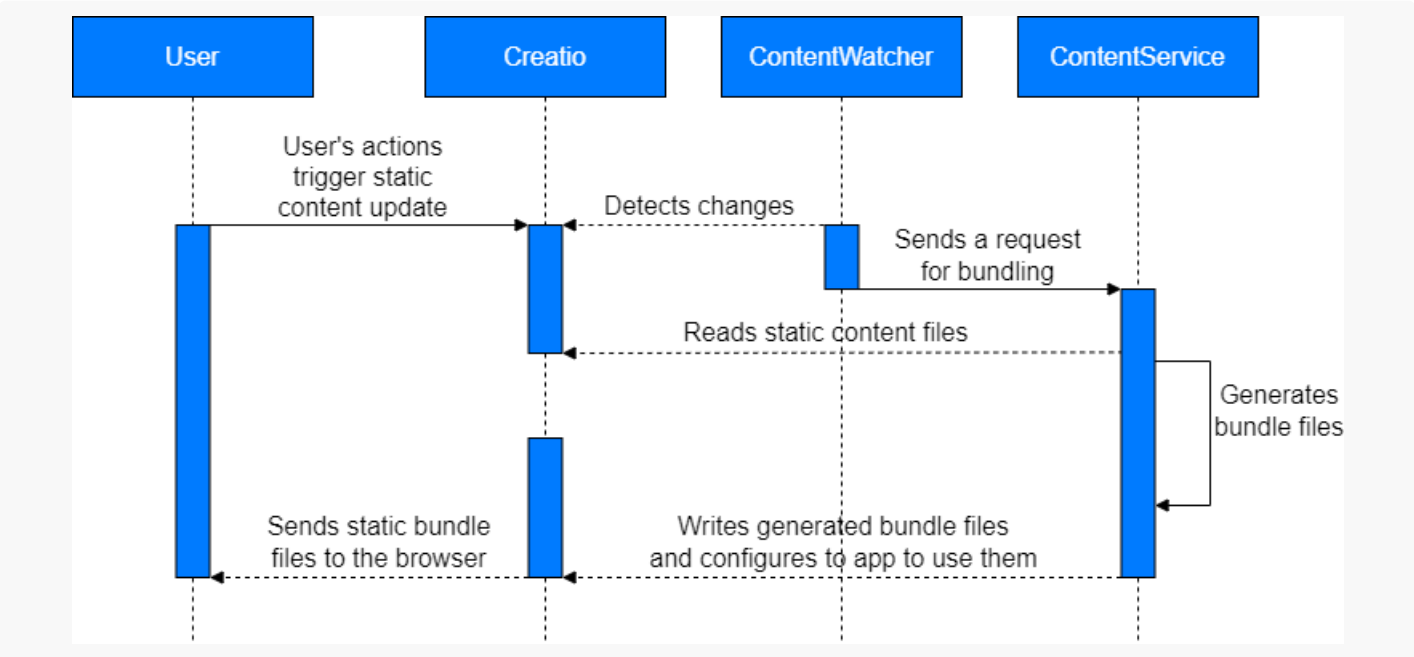
Minification and bundling are both performed by the **static content bundling service**. Most web apps are delivered with minified and bundled files. Since the Creatio configuration and static content can change during the life cycle, the service runs minification and bundling automatically when necessary.

Components of the bundling service:

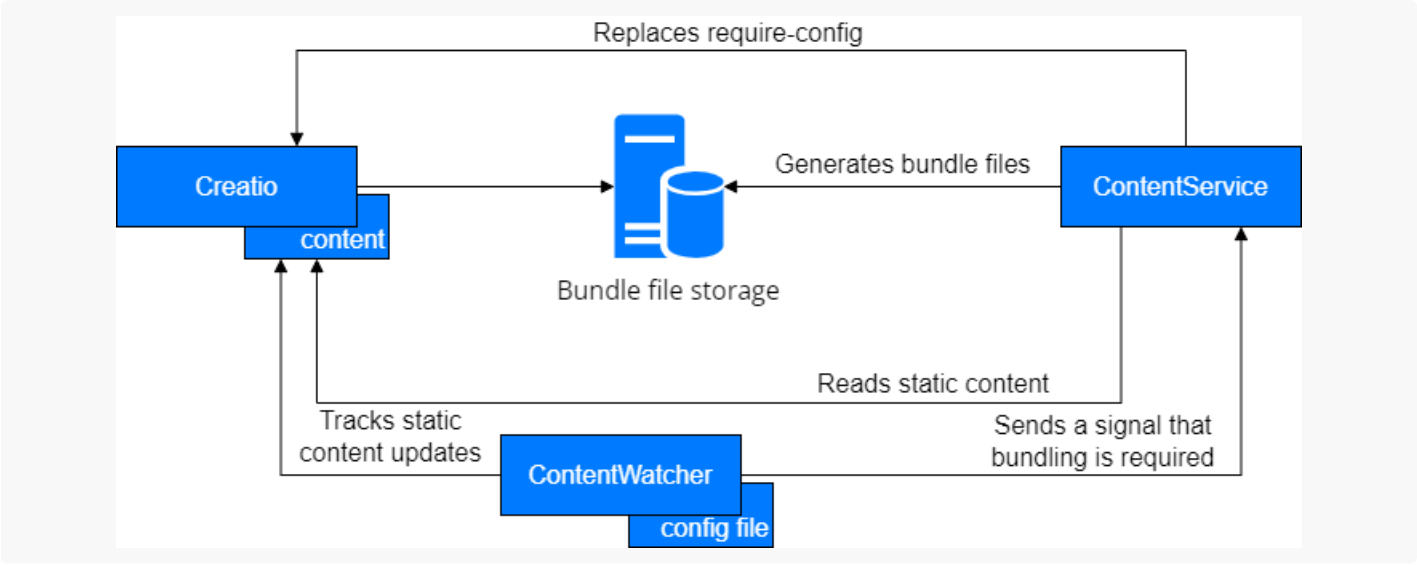
- `ContentService`. The service that minifies and bundles Creatio files. Deployed independently of Creatio.
- `ContentWatcher`. A utility that interacts with Creatio files and `ContentService` to ensure timely minification and bundling.

To improve performance, the app is configured to generate minified bundle files by default.

View the bundling service flowchart in the figure below.



View the static content bundling workflow in the figure below.



Requirements for the correct operation of the bundling service:

- Uninterrupted operation of `ContentService` and `ContentWatcher`.
- `ContentService` and `ContentWatcher` access to static content.
- Ability to access static content from `ContentService` to `ContentWatcher` via HTTP.
- A valid configuration of `ContentService` and `ContentWatcher`.

You must configure the bundling service for Creatio **on-site**. Creato **cloud** has the bundling service configured out-of-the-box.

ContentService

`ContentService` is the service that minifies and bundles Creatio static content. Deployed independently of Creatio.

ContentService endpoints

`ContentService` endpoints

Endpoint	Description	Methods
`/`	Checks whether the bundling service is functional.	`GET`
`/process-content`	Primary endpoint. Enables generation of minified bundled files. Changes the `_ContentBootstrap.js` configuration file.	`POST`
`/clear-bundles`	Disables generation of minified bundled files and removes bundled files. Changes the `_ContentBootstrap.js` configuration file.	`POST`
`/minify-content`	Minifies static content.	`POST`

`ContentService` can be paired with `ContentWatcher` or work independently. If the `ContentService` works independently, access the endpoints manually or using a third-party automation, for example, when deploying or updating Creatio.

The `/` endpoint takes no parameters. Other endpoints accept various parameters passed within the request body in JSON.

Properties of the `ContentService` endpoints

Property	Endpoints	Description
Content Path **required**	All endpoints.	Path to the directory that contains the static content.
Output Path	All endpoints.	Path to the directory that stores the files generated by the bundling service, for example, bundle files. • For the `/minify-content` endpoint, the value of the current parameter matches the value of the `ContentPath` parameter. • For the `/process-content` and `/clear-bundles` endpoints, the value of the current parameter matches the value of the `ContentPath` parameter that has the `/bundles` suffix added.
Schemas Bundles Require Url	`/process-content`	The URL used in the RequireJs configuration file for the browser to load bundled and/or minified files. Required when changing the default value of the `OutputPath` parameter to a different value.
MinifyJs	`/process-content`	A flag that specifies whether to minify *.js files. The default value is `true`.
Minify Configs	`/process-content`	A flag that specifies whether to minify the generated RequireJs config files. The default value is `true`.
Apply Bundles For Schemas	`/process-content`	A flag that specifies whether to bundle *.js files of static schema content. The default value is `true`.
Apply Bundles For Resources	`/process-content`	A flag that specifies whether to bundle *.js files of schema resources. The default value is `true`.
Force	`/clear-bundles`	A flag that specifies whether to remove bundle files. The default value is `false` (configures Creatio to use unbundled static content).

Configure `ContentService`

The bundling service does not require a change in the `ContentService` configuration to work correctly. By default, the bundling service comes optimally configured, but you can change the settings to boost performance or enable additional functionality.

`ContentService` settings

Setting	Description
`Logging`	Manage logging levels.
`UseParallelism`	Manage parallelism. The default value is `false`.
`UseCancellation`	Manage automatic undo operations.
`ShutdownFinalizationDelay`	Manage the amount of extra waiting time in milliseconds to complete the bundling service safely.
`BundleGeneration`	Manage bunding. Contains the `DictionaryPath`, `IgnoredFileNamePatterns`, `BundleMinLength` settings.
`DictionaryPath`	The path to the dictionary. The setting is contained in the `BundleGeneration` block.
`IgnoredFileNamePatterns`	Manage how to ignore files. The setting is contained in the `BundleGeneration` block.
`BundleMinLength`	Size threshold in characters/bytes for generated bundle files. If the bundle file size exceeds the specified value during bundling, its generation stops and the next bundle file is created. The default value is `204800` (generates files between 200 and 250 Kb in size). The setting is contained in the `BundleGeneration` block.
`InfluxDbConnectionString`	InfluxDb connection string. Learn more in a separate article: InfluxDb metrics.
`InfluxDbConnectionTimeout`	Timeout in milliseconds for InfluxDb connection. The default value is `5000`.

Parallelism speeds up minification and bundling. The `UseParallelism` setting toggles parallelism. If parallelism is enabled, file processing operations are split into independent blocks. Depending on the infrastructure, this can speed up the bundling service significantly. Parallelism significantly increases the load on the CPU as well as network and/or drives.

Automatic cancellation of operations prevents overusing resources when repeatedly requesting static content if the previous request has not been completed. The `UseCancellation` setting, which is enabled by default, toggles automatic cancellation of operations. If automatic cancellation is enabled and the bundling receives a request to process static content that is already being processed, the bundling service schedules the next operation and tries to cancel the current operation.

Note. Automatic cancelation of operations is performed only when bundling and does not affect the execution of other operations.

On a standard exit (for example, by pressing `Ctrl + C` in the console or stopping the pool via IIS), the bundling service stops receiving new requests as well as tries to end the ongoing processing safely. The safe termination option that the service uses depends on the value of the `UseCancellation` setting.

Safe termination **options**:

- If the `UseCancellation` setting is enabled, Creatio terminates the execution of cancelable tasks.
- If the `UseCancellation` setting is disabled or a task is not cancelable, Creatio awaits until the started tasks are complete.

Attention. Always prefer safe termination unless a forcible termination is required. Forcibly terminating the `ContentService` might cause Creatio to fail.

Dictionary is the schema bundling order. Use it to optimize the creation of bundle files. Schemas not in the dictionary are included in the bundle files in alphabetical order. Specify the path to the `BundlesDictionary.txt` file of the dictionary in the `DictionaryPath` setting of the `BundleGeneration` block. The `BundlesDictionary`.txt file is pre-generated and delivered with `ContentService`. We recommend generating a custom dictionary for heavily customized instances.

To **generate a custom dictionary**:

1. Download the *.zip archive that contains the FileListGetter utility. The *.zip-archive is available via this link. Please note that the utility requires the .NET Framework package version 4.7.2 or later in the Creatio instance to generate a custom dictionary.
2. Open the app page to generate a dictionary.
3. Open the developer tools and go to the [*Network*] tab. Learn more about developer tools in a separate article: Front-end debugging.
4. Refresh the page to load static content files.
5. Right-click the downloaded file area and save the *.har file to the `HarFiles` directory of the FileListGetter utility.
6. Repeat steps 1 through 4 for all app pages to generate a dictionary.
7. Copy the `BundlesDictionary.txt` file to the `Output` directory of the FileListGetter utility.
8. Run the `FileListGetter.exe` utility file.

As a result, the `BundlesDictionary.txt` file in the `Output` directory will contain a custom dictionary.

Ignoring files lets you specify files that must not be bundled. The `IgnoredFileNamePatterns` setting of the `BundleGeneration` block manages files to ignore.

Files when bundling:

- Static content files that are not valid RequireJS modules. Browsers cannot load these files from bundle files.
- Static content configuration files that are loaded in a special way and must not be bundled.

ContentWatcher

`ContentWatcher` is a utility that interacts with Creatio files and `ContentService` to ensure timely minification and bundling.

Actions that `ContentWatcher` performs:

- Monitors the specified static content files to see if their content changes.
- Notifies `ContentService` that the file contents are updated and passes the appropriate parameters to the service.

`ContentWatcher` watches for changes to the `_MetaInfo.json` file, which changes when static content is updated. `ContentWatcher` then sends a notification to `ContentService` about a pending bundling. `ContentWatcher` can track changes to any file and send a notification to any URL. For the correct orchestration of Creatio, `ContentWatcher`, and `ContentService`, configure `ContentWatcher`.

`ContentWatcher` can monitor multiple Creatio instances.

`ContentWatcher` Settings

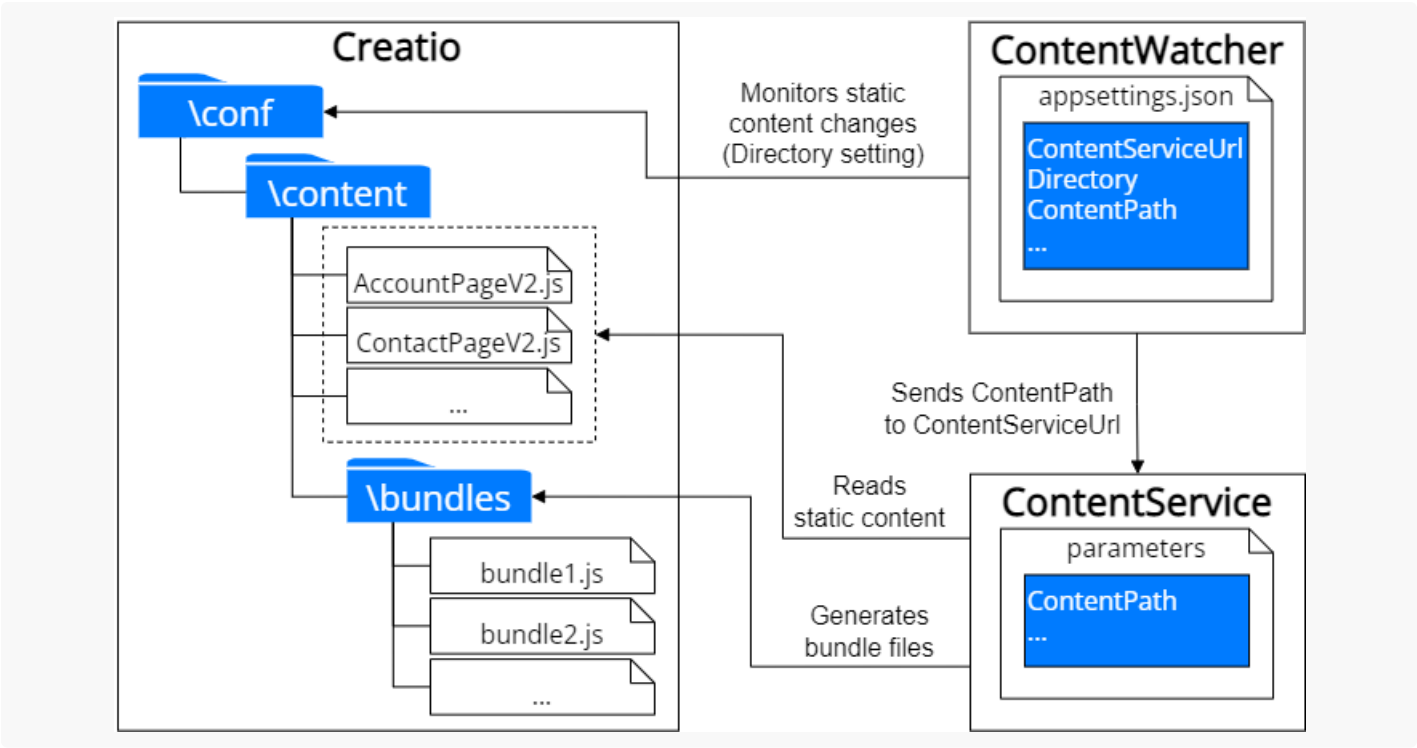
Setting	Description
General settings	
`ContentServiceUrl` **required**	Endpoint URL to call when changes are detected. Use the endpoint URL template below. **Endpoint URL pattern** `http://[Address of ContentService]/process-content`
`DefaultFileFilter`	A pattern that filters files whose changes to monitor. The default value is `_MetaInfo.json`. Used if the `FileFilter` setting is not specified for Creatio.
`ConfigurationRefresh Interval`	The interval in milliseconds between readings of Creatio instance settings by the bundling service. The default value is `300000`, which is equivalent to five hours. When changes are detected in site settings, those changes are applied without interrupting `ContentWatcher`.
`InfluxDbConnectionString`	InfluxDb connection string. Learn more in a separate article: InfluxDb metrics.
`InfluxDbConnectionTimeout`	Timeout in milliseconds for InfluxDb connection. By default, `5000`.
Settings for Creatio on-site	
`ContentWorkers` **required**	Setting block for each Creatio instance that `ContentWatcher` serves.
`Name` **required**	A custom name you can use to identify the current instance in the logs.
`Directory` **required**	The path to the directory to monitor for changes. I. e., the path to the `\conf` directory of the current Creatio instance that contains the `_MetaInfo.json` file.
`FileFilter`	A pattern that filters files whose changes to monitor. Not used if you specify the `DefaultFileFilter` setting for the instance.
`ContentWorkerArguments` **required**	Parameters in key-value format to pass to the `ContentService` when changes are detected. For the `ContentService` to work correctly, specify an object that has the `ContentPath` key and a value that contains the path to the `\conf\content` directory.

If `ContentWatcher` and `ContentService` are deployed on Linux (including Docker) and the Creatio file system is shared with `ContentWatcher` and `ContentService`, specify the paths to the directories where the Creatio file system is mounted in the `Directory` and `ContentPath` settings. For example, if the `\conf` directory of the Creatio instance is mounted into a container in the `/app/content/` path, specify `/app/content/` in the `Directory` setting and `/app/content/content` in the `ContentPath` setting.

Configuration file example (Creatio on-site)

```
{
    "ContentServiceUrl": "http://host.docker.internal:29572/process-content",
    "ConfigurationRefreshInterval": "86400000",
    "DefaultFileFilter": "_MetaInfo.json",
    "ContentWorkers": [
        {
            "Name": "Site 1",
            "Directory": "/path_to_creatio_conf_folder/",
            "ContentWorkerArguments": {
                    "key": "ContentPath",
                    "value": "/path_to_creatio_conf_folder/content"
                }
            ]
        }
    ]
}
```

Below is a diagram of the interaction of `ContentService`, `ContentWatcher` and Creatio, which are deployed using the default configuration. The diagram takes into account the general settings.



InfluxDb metrics

`ContentWatcher` and `ContentService` enable recording the metrics of successful and unsuccessful operations in InfluxDb. The InfluxDb documentation is available on the official InfluxDb website. Configure the connection to InfluxDb in the `appsettings.json` file using the parameter values of the `InfluxDbConnectionString` connection string.

`InfluxDbConnectionString` connection string

```
"InfluxDbConnectionString": "url=http://1.2.3.4:5678; db=content; user=User1; password=QwErTy; v
```

View the **parameters** of the `InfluxDbConnectionString` connection string in the table below.

`InfluxDbConnectionString` connection string

Setting	Description
`url` **required**	InfluxDb server address.
`db`	The name of the database to write the metrics. The default value is `content`.
`user`	The user name to connect to the InfluxDb server. The default value is an empty string.
`password`	The password to connect to the InfluxDb server. The default value is an empty string.
`version`	InfluxDb server version. The default value is `Latest`. Available values: `Latest`, `v_1_3`, `v_1_0_0`, `v_0_9_6`, `v_0_9_5`, `v_0_9_2`, `v_0_8_x`.

View the InfluxDb **metrics** that `ContentService` and `ContentWatcher` record in the table below.

InfluxDb metrics

Metric	Description
Metrics written by `ContentService`	
`service_processing_duration`	Content processing duration.
`service_error`	Errors received while bundling, deleting temporary directories, cleaning bundle files.
Metrics written by `ContentWatcher`	
`watcher_notifying_duration`	The duration of the `ContentService` notification.
`watcher_error`	Errors received while loading settings, monitoring, notifying `ContentService`.

Deploy the static content bundling service

Deploy the static content bundling service in Docker. Docker documentation is available on the official Docker website.

To deploy the bundling service, use a server running a Linux OS (we recommend using stable versions of Ubuntu or Debian), with a stable version of Docker installed and configured. The server must be allowed to query the Docker Hub image library. The server serves as a host machine for the components of the bundling service.

To **deploy the bundling service**:

1. Download the *.zip archive that contains the service configuration files. The *.zip-archive is available via this link.
2. Unpack the archive to an arbitrary directory (for example, `\opt\services`).

   Service configuration structure:

   - `\etc\content-watcher\appsettings.json` is the `ContentWatcher` configuration file.
   - `docker-compose.yml` is the configuration file of the docker-compose utility.
   - `*.env` is the file that contains the environment variables to run the components.
3. Specify the following **parameters** in the `.\docker-compose\*.env` file:

   - `ContentServicePort`. The port where `ContentService` runs.
   - `ContentPath`. The container's built-in directory of sites and their contents specified.
4. Configure the list of sites to track using `ContentWatcher`.
5. Download the required docker service images to the directory where the configuration files are deployed, for example, `\opt\services`.

   Command that downloads the required docker service images

   ```
   docker-compose pull
   ```

   As a result of executing the command, Docker will download the necessary docker service images from the official Docker Hub website.

   steps to download docker service images if online access is disabled on the server:

   a. Download the necessary images manually on a public machine (see the `docker-compose.yml` configuration file).

   b. Transfer the downloaded docker images to the target machine as files.

      Command that transfers the downloaded docker service images

      ```
      docker export/import
      ```
6. Start the services.

   Command that starts the services

   ```
   docker-compose up -d
   ```

   As a result, Docker will start the services.

7. Verify that the bundling service is deployed correctly. To do this, make a `GET` request to the address below.

   GET-request pattern to validate the bundling service

   ```
   {Server IP address}: {CONTENT_SERVICE_PORT}
   ```

   GET-request pattern to validate the bundling service

   ```
   17.17.17.7:9999
   ```

   `Server IP address` is the IP address of the server that contains the installed services.

   `CONTENT_SERVICE_PORT` is the port where `ContentService` runs. Matches the port specified in the `.\docker-compose\.*env` file on step 2.

   As a result of the command, you will receive the `Service is running` response.