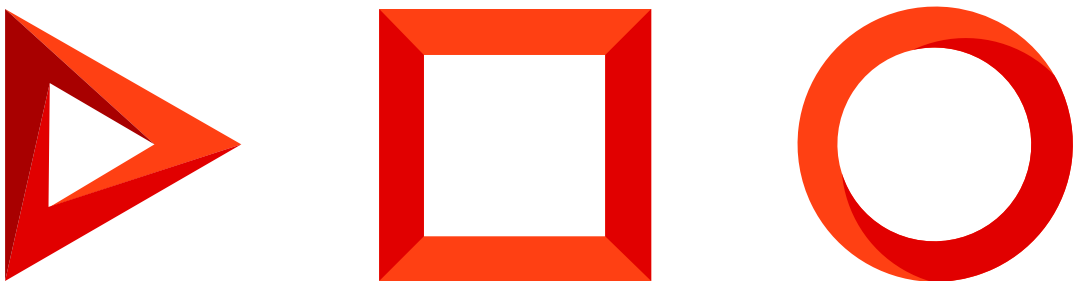
Creatio Academy

Сервис бандлирования статического контента

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

Содержание

Сервис бандлирования статического контента

Сложный

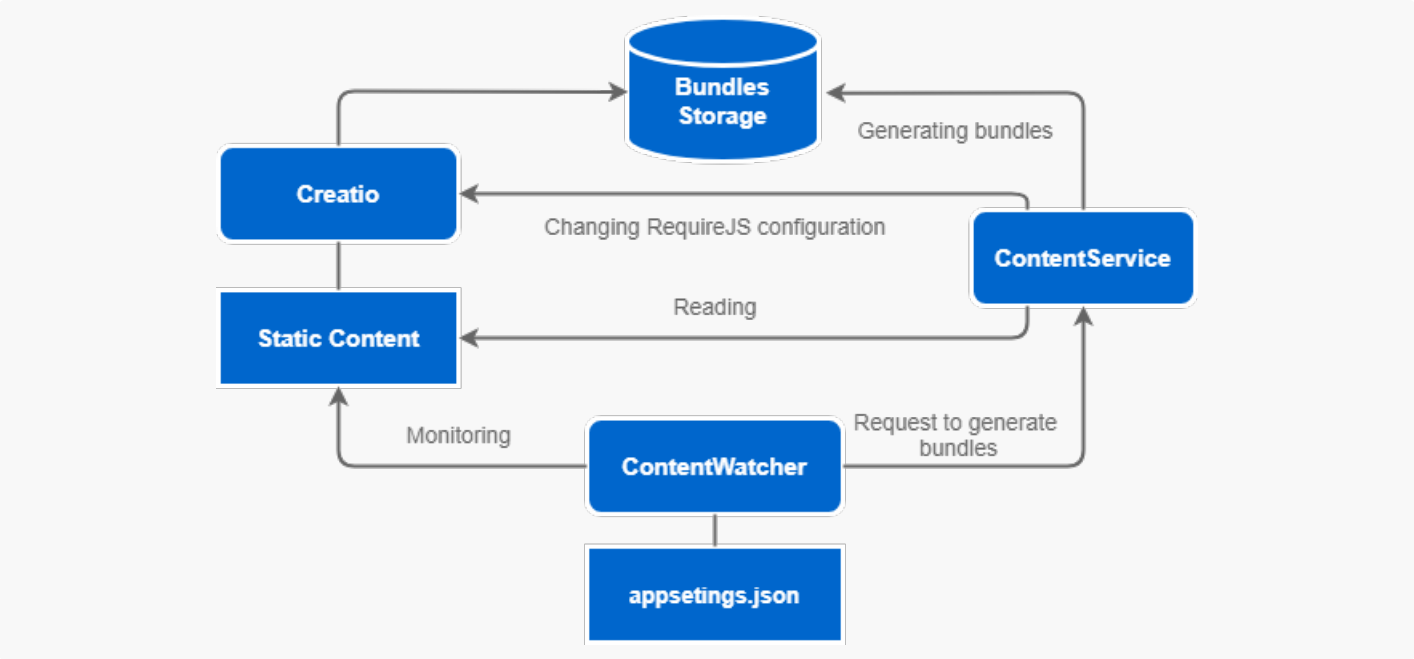
Для повышения производительности весь клиентский контент (исходный код клиентских схем, css-стили) генерируется в специальном каталоге приложения Creatio. Преимущества такого подхода приведены в статье. Однако, из-за большого количества файлов браузеру необходимо выполнять большое количество запросов к приложению при его первоначальной загрузке. Чтобы избежать этого, обычно выполняется объединение всех однотипных файлов в один bundle-файл — бандлирование.

Для объединения однотипных файлов статического контента Creatio разработан сервис бандлирования статического контента.

Структура и принцип работы

Приложение-наблюдатель (`ContentWatcher`) мониторит файлы в каталоге со статическим контентом и сообщает об изменениях в них веб-сервису. Веб-сервис (`ContentService`) выполняет перегенерацию bundle-файлов приложения при поступлении определенных запросов (например, от `ContentWatcher` или вручную). После бандлирования веб-сервис изменяет определенный конфигурационный файл Creatio таким образом, чтобы он содержал информацию о необходимости использовать bundle-файлы вместо оригинального статического контента.

Общая схема работы сервиса:



На заметку. Веб-сервис может быть установлен без приложения-наблюдателя (ContentWatcher). В

таком случае запросы к ContentService на бандлирование или минификацию необходимо выполнять вручную.

На заметку. Компоненты сервиса могут быть установлены на том же компьютере, что и Creatio, или на отдельном компьютере. Если они установлены отдельно, то у них должен быть сетевой доступ к файлам статического контента Creatio.

ContentService

Является .NET Core 2.1 веб-сервисом и имеет следующие операции (точки доступа):

- `/` — проверка работоспособности сервиса (метод `GET`).
- `/process-content` — генерирует минифицированные bundle-файлы (метод `POST`).
- `/clear-bundles` — очищает bundle-файлы (метод `POST`).
- `/minify-content` — минифицирует контент (метод `POST`).

Важно. Каждая операция, (кроме `/`) не только выполняет действия над файлами, но и изменяет конфигурационный файл `_ContentBootstrap.js`.

Параметры операций:

- `ContentPath` — путь к статическому контенту приложения. Обязательный параметр.
- `OutputPath` — локальный или сетевой путь для `ContentService`, куда будут сохранятся бандлированные и/или минифицированные файлы. По умолчанию `ContentPath` + "`/bundles`".
- `SchemasBundlesRequireUrl` — путь, который будет использоваться в конфигурационном файле `RequireJs` для бандлированных и/или минифицированных файлов. При изменении `OutputPath` со значения по умолчанию на любое другое этот параметр нужно тоже указывать. По умолчанию считается, что `OutputPath` = `ContentPath` + "`/bundles`".
- `BundleMinLength` — пороговое значение длины одного бандла. Длина, при которой создается новый бандл. По умолчанию 204800Б (200КБ).
- `MinifyJs` — применять ли минификацию к JavaScript файлам (`true`/`false`). По умолчанию `true`.
- `MinifyCss` — применять ли минификацию к CSS файлам (`true`/`false`). По умолчанию `true`.
- `MinifyConfigs` — применять ли минификацию к конфигурационным файлам `RequireJs` (`true`/`false`). По умолчанию `true`.
- `ApplyBundlesForSchemas` — применять ли бандлирование к схемам (`true`/`false`). По умолчанию `true`.
- `ApplyBundlesForSchemasCss` — применять ли бандлирование к CSS схем (`true`/`false`). По умолчанию `false`.
- `ApplyBundlesForAloneCss` — применять ли бандлирование к отдельным CSS файлам, у которых нет связанного JavaScript-модуля (`true`/`false`). По умолчанию `false`.

- `ApplyBundlesForResources` — применять ли бандлирование к ресурсам схем (`true` / `false`). По умолчанию `true`.

ContentWatcher

Является .NET Core 2.1 приложением. Запускается как служба (также может запускаться через .NET Core CLI Tools). Основная задача — наблюдать за изменением указанного в параметре `fileFilter` файла, который находится по пути, указанном в параметре `directory`. По умолчанию это файл `_MetaInfo.json`. Изменение файла сообщает об обновлении статического контента. При обнаружении изменений `ContentWatcher` оповещает `ContentService` о необходимости перегенерировать bundle-файлы.

Параметры запуска (указываются в `appsettings.json`):

- `ContentServiceUrl` — ссылка на приложение ContentService.
- `CloudServiceId` — идентификатор сервиса для считывания настроек из базы данных AzManager. Необязательный параметр. Необходим для считывания настроек облачной инфраструктуры Creatio.
- `DefaultFileFilter` — имя по умолчанию отслеживаемого файла.
- `AzManagerConnectionString` — строка подключения к базе данных AzManager. Необязательный параметр. Необходим для считывания настроек облачной инфраструктуры Creatio.
- `ConfigurationRefreshInterval` — период обновления конфигурации сервиса.
- `ContentWorkers` — массив конфигурационных объектов всех приложений, за которыми будет следить `ContentWatcher`. Свойства конфигурационных объектов:
  - `name` — имя для отслеживаемого сайта, которое будет отображено в логах работы службы.
  - `directory` — путь к отслеживаемому файлу для настраиваемого сайта.
  - `fileFilter` — имя отслеживаемого файла.
  - `ContentWorkerArguments` — массив дополнительных параметров, которые будут передаваться в ContentService. Элементы массива — объекты ключ-значение. Свойства объектов:
    - `key` — ключ. По умолчанию "contentPath".
    - `value` — путь к каталогу с файлами статического контента настраиваемого сайта.

Пример конфигурационного файла `appsettings.json`

```
{
    "ContentServiceUrl": "http://localhost:9563/process-content",
    "ConfigurationRefreshInterval": "60000",
    "DefaultFileFilter": "_MetaInfo.json",
    "CloudServiceId": "151",
    "AzManagerConnectionString": "Data Source=azserver\\mssql2016;Initial Catalog=azmanager;Inte
    "ContentWorkers": [
        {
            "name": "My Creatio",
            "directory": "C:/Creatio/Terrasoft.WebApp/conf",
            "fileFilter": "_MetaInfo.json",
```

```
            "ContentWorkerArguments": [
                {
                    "key": "contentPath",
                    "value": "C:/Creatio/Terrasoft.WebApp/conf/content"
                }
            ]
        }
    ]
}
```

Метрики InfluxDb

ContentWatcher и ContentService при необходимости могут записывать метрики успешных и неуспешных операций в InfluxDb. Для этого в соответствующем файле `appsettings.json` следует указать строку `InfluxDbConnectionString` с параметрами подключения:

- `url` — адрес сервера InfluxDb, обязательный параметр.
- `db` — имя базы данных, куда будут записываться метрики. Необязательный параметр, значение по умолчанию: `content`.
- `user` — имя пользователя для соединения с сервером. Необязательный параметр, значение по умолчанию: `<пустая строка>`.
- `password` — пароль для соединения с сервером. Необязательный параметр, значение по умолчанию: `<пустая строка>`.
- `version` — версия сервера InfluxDb. Возможные значения: `Latest`, `v_1_3`, `v_1_0_0`, `v_0_9_6`, `v_0_9_5`, `v_0_9_2`, `v_0_8_x`. Необязательный параметр, значение по умолчанию: `Latest`.

Пример

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

Метрики, записываемые `ContentWatcher`:

- `watcher_notifying_duration` — длительность оповещения `ContentService`.
- `watcher_error` — ошибки при загрузке настроек, мониторинге, оповещении `ContentService`.

Метрики, записываемые `ContentService`:

- `service_processing_duration` — длительность обработки контента.
- `service_error` — ошибки при генерации бандлов, удалении временных папок, очистке бандлов.

Развернуть сервис в Docker


Сложный

Системные требования

- сервер под управлением Linux OS (рекомендуются стабильные версии Ubuntu или Debian), с установленной и настроенной стабильной версией docker. С этого сервера должны быть разрешены запросы к хранилищу образов Docker Hub.
- на сервере должны быть установлены Docker и Docker Compose (см. документацию Docker).

Структура конфигурации сервисов

- `ets/content-watcher/appsettings.json` — конфигурационный файл ContentWatcher.
- `docker-compose.yml` — конфигурационный файл утилиты docker-compose.
- `.env` — файл с переменными окружения для запуска компонентов.

Установка сервисов

1. В произвольный каталог (например, `/opt/services`) скачайте файлы конфигурации сервисов и разархивируйте их.
2. В `./docker-compose/.env` укажите необходимые параметры:
   - `ContentServicePort` — порт, на котором будет работать ContentService.
   - `ContentPath` — каталог с сайтами/контентом сайтов, который будет смонтирован в контейнер.
3. В `./docker-compose/etc/content-watcher/appsettings.json` настройте список сайтов для отслеживание ContentWatcher (см. выше).
4. Из каталога, в котором находятся конфигурационные файлы (например, `/opt/services`) выполните команду:

   Команда для загрузки необходимых docker-образов сервисов

   ```
   sudo docker-compose pull
   ```

   Результатом выполнения команды будет загрузка из hub.docker.com необходимых docker-образов сервисов.

   Важно. Если на сервере запрещен доступ в интернет, скачайте на машине с открытым доступом все необходимые образы вручную (см.конфигурационный файл `docker-compose.yml`). Затем воспользуйтесь командами `sudo docker export` и `sudo docker import` для переноса образов в виде файлов на целевую машину.

5. Из каталога, в котором находятся конфигурационные файлы (например, `/opt/services`) выполните команду:

Команда для запуска сервисов

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

Результатом выполнения команды будет запуск сервисов.

6. Проверьте установку, отправив `GET`-запрос по адресу: `{IP-адрес сервера}: {ContentServicePort}`, где:

   - `IP-адрес сервера` — IP-адрес сервера, на котором установлены сервисы.
   - `ContentServicePort` — порт, на котором работает ContentService. Должен совпадать с портом, указанном в файле `./docker-compose/.env` (см. п. 2).

Пример

```
10.17.17.7:9999
```

Ожидаемый ответ

```
"Service is running"
```