Deploying applications to test and production environments

Overview

When you finish a development cycle of your application, deploy it to a testing environment, and then to a production environment.

Jump to

- Deploying or updating an adapter to a production environment
- Configuring SSL between MobileFirst adapters and back-end servers by using self-signed certificates
- Building an application for a test or production environment
- Registering an application to a production environment
- Transferring server-side artifacts to a test or production server
- Updating MobileFirst apps in production

Deploying or updating an adapter to a production environment

Adapters contain the server-side code of applications that are deployed on and serviced by IBM MobileFirst Foundation. Read this checklist before you deploy or update an adapter to a production environment. For more information about creating and building adapters, see Developing the server side of a MobileFirst application (../../adapters).

Adapters can be uploaded, updated, or configured while a production server is running. After all the nodes of a server farm receive the new adapter or configuration, all incoming requests to the adapter use the new settings.

1. If you update an existing adapter in a production environment, make sure that this adapter contains no incompatibilities or regressions with existing applications that are registered to a server.

   The same adapter can be used by multiple applications, or by multiple versions of the same application, that are already published to the store and used. Before you update the adapter in a production environment, run non-regression tests in a test server against the new adapter and copies of the apps that are built for the test server.
2. For Java adapters, if the adapter uses Java URLConnection with HTTPS, make sure that the back-end certificates are in the MobileFirst Server keystore.

   For more information, see Using SSL in HTTP adapters (../../adapters/javascript-adapters/js-http-adapter/using-ssl/). For more information about using self-signed certificates, see Configuring SSL between MobileFirst adapters and back-end servers by using self-signed certificates.

   > **Note:** If the application server is WebSphere Application Server Liberty, then the certificates must also be in the Liberty truststore.
3. Verify the server-side configuration of the adapter.
4. Use the `mfpadm deploy adapter` and `mfpadm adapter set user-config` commands to upload the adapter and its configuration.

   For more information about **mfpadm** for adapters, see Commands for adapters (../using-cli/#commands-for-adapters).

Configuring SSL between MobileFirst adapters and back-end servers by using self-signed certificates

You can configure SSL between MobileFirst adapters and back-end servers by importing the server self-signed SSL certificate to the MobileFirst keystore.

1. Export the server public certificate from the back-end server keystore.

   > **Note:** Export back-end public certificates from the back-end keystore by using keytool or openssl lib. Do not use the export feature in a web browser.
2. Import the back-end server certificate into the MobileFirst keystore.
3. Deploy the new the MobileFirst keystore. For more information, see Configuring the MobileFirst Server keystore (../../authentication-and-security/configuring-the-mobilefirst-server-keystore/).

Example

The **CN** name of the back-end certificate must match what is configured in the adapter-descriptor **adapter.xml** file. For example, consider an **adapter.xml** file that is configured as follows:

```
<protocol>https</protocol>
<domain>mybackend.com</domain>
```

The back-end certificate must be generated with **CN=mybackend.com**.

As another example, consider the following adapter configuration:

```
<protocol>https</protocol>
<domain>123.124.125.126</domain>
```

The back-end certificate must be generated with **CN=123.124.125.126**.

The following example demonstrates how you complete the configuration by using the Keytool program.

1. Create a back-end server keystore with a private certificate for 365 days.

   ```
   keytool -genkey -alias backend -keyalg RSA -validity 365 -keystore backend.keystore -storetype JKS
   ```

> **Note:** The **First and Last Name** field contains your server URL, which you use in the **adapter.xml** configuration file, for example **mydomain.com** or **localhost**.

2. Configure your back-end server to work with the keystore. For example, in Apache Tomcat, you change the **server.xml** file:

```
<Connector port="443" SSLEnabled="true" maxHttpHeaderSize="8192"
 maxThreads="150" minSpareThreads="25" maxSpareThreads="200"
 enableLookups="false" disableUploadTimeout="true"
 acceptCount="100" scheme="https" secure="true"
 clientAuth="false" sslProtocol="TLS"
 keystoreFile="backend.keystore" keystorePass="password" keystoreType="JKS"
 keyAlias="backend"/>
```

3. Check the connectivity configuration in the **adapter.xml** file:

```
<connectivity>
 <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
  <protocol>https</protocol>
  <domain>mydomain.com</domain>
  <port>443</port>
  <!-- The following properties are used by adapter's key manager for choosing a specific certificate from the key store
  <sslCertificateAlias></sslCertificateAlias>
  <sslCertificatePassword></sslCertificatePassword>
  -->
 </connectionPolicy>
 <loadConstraints maxConcurrentConnectionsPerNode="2"/>
</connectivity>
```

4. Export the public certificate from the created back-end server keystore:

```
keytool -export -alias backend -keystore backend.keystore -rfc -file backend.crt
```

5. Import the exported certificate into your MobileFirst Server keystore:

```
keytool -import -alias backend -file backend.crt -storetype JKS -keystore mfp.keystore
```

6. Check that the certificate is correctly imported in the keystore:

```
keytool -list -keystore mfp.keystore
```

7. Deploy the new the MobileFirst Server keystore.

Building an application for a test or production environment

To build an application for a test or production environment, you must configure it for its target server. To build an application for a production environment, additional steps apply.

1. Make sure that the target server keystore is configured. For more information, see Configuring the MobileFirst Server keystore (../../authentication-and-security/configuring-the-mobilefirst-server-keystore/).
2. If you plan to distribute the app installable artifact, increment the app version.
3. Before you build your app, configure it for the target server.

   You define the URL and runtime name of the target server in the client properties file. You can also change the target server by using the IBM MobileFirst Command Line Interface (CLI). To configure the app for a target server without registering the app to a running server, you can use the `mfpdev app config server <server URL>`and `mfpdev app config runtime <runtime_name>` commands. Alternatively, you can register the app to a running server by running the `mfpdev app register` command. Use the public URL of the server. This URL is used by the mobile app to connect to MobileFirst Server.

   For example, to configure the app for a target server mfp.mycompany.com with a runtime that has the default name mfp, run `mfpdev app config server https://mfp.mycompany.com` and `mfpdev app config runtime mfp`.
4. Configure the secret keys and authorized servers for your application.
   - If your app implements certificate pinning, use the certificate of your target server. For more information about certificate pinning, see Certificate pinning (../../authentication-and-security/certificate-pinning).
   - If your iOS app uses App Transport Security (ATS), configure ATS for your target server.
   - To configure secure Direct Update for an Apache Cordova application, see Implementing secure Direct Update on the client side (../../application-development/direct-update).
   - If you develop your app with Apache Cordova, configure the Cordova Content Security Policy (CSP).
5. If you plan to use Direct Update for an application that is developed with Apache Cordova, archive the versions of the Cordova plug-ins you used to build the app. Direct Update cannot be used to update native code. If you changed a native library or one of the build tools in a Cordova project and uploaded such a file to MobileFirst Server, the server detects the difference and does not send any updates for the client application. The changes in the native library might include a different Cordova version, a newer Cordova iOS plug-in, or even an mfpdev plug-in fix pack that is newer than the one that was used to build the original application.
6. Configure the app for production use.
   - Consider disabling printing to the device log.
   - If you plan to use IBM MobileFirst Analytics, verify that your app sends collected data to the MobileFirst Server.
   - Consider disabling features of your app that call the setServerURL API, unless you plan to make a single build for multiple test servers.
7. If you build for a production server and plan to distribute the installable artifact, archive the app source code to be able to run non regression-tests for this app on a test server.

For example, if you later update an adapter, you might run non-regression tests on already distributed apps that use this adapter. For more information, see Deploying or updating an adapter to a production environment.

8. Optional: Create the application-authenticity file for your application.

   You use the application-authenticity file after you register the application to the server to enable the application-authenticity security check.

   - For more information, see Enabling the application-authenticity security check (../../authentication-and-security/application-authenticity).
   - For more information about registering an application to a production server, see Registering an application to a production environment.

Registering an application to a production environment

When you register an application to a production server, you upload its application descriptor, define its license type, and optionally activate application authenticity.

Before you begin

- Verify that MobileFirst Server keystore is configured and is not the default keystore. Do not use a server in production with a default keystore. The MobileFirst Server keystore defines the identity of MobileFirst Server instances, and is used to digitally sign OAuth tokens and Direct Update packages. You must configure the server's keystore with a secret key before you use it in production. For more information, see Configuring the MobileFirst Server keystore (../../authentication-and-security/configuring-the-mobilefirst-server-keystore/).
- Deploy the adapters used by the app. For more information, see Deploying or updating an adapter to a production environment.
- Build the application for your target server. For more information, see Building an application for a test or production environment.

When you register an application with a production server, you upload its application descriptor, define its license type, and optionally activate application authenticity. You might also define your update strategy if an older version of your app is already deployed. Read the following procedure to learn about important steps, and about ways to automate them with the **mfpadm** program.

1. If your MobileFirst Server is configured for token licensing, make sure that you have enough available tokens on the License Key Server. For more information, see Token license validation (../license-tracking/#token-license-validation) and Planning for the use of token licensing (../../installation-configuration/production/token-licensing/#planning-for-the-use-of-token-licensing).

   > **Tip:** You can set the token license type of your app before you register the first version of your app. For more information, seeSetting the application license information (../license-tracking/#setting-the-application-license-information).

2. Transfer the application descriptor from a test server to the production server.

   This operation registers your application to the production server and upload its configuration. For more information about transferring an application descriptor, see Transferring server-side artifacts to a test or production server.

3. Set the application license information For more information, see Setting the application license information (../license-tracking/#setting-the-application-license-information).
4. Configure the application-authenticity security check. For more information about configuring the application-authenticity security check, see Configuring the application-authenticity security check (../../authentication-and-security/application-authenticity/#configuring-application-authenticity).

   > **Note:** You need the application binary file to create the application-authenticity file. For more information, see Enabling the application-authenticity security check (../../authentication-and-security/application-authenticity/#enabling-application-authenticity).

5. If your application uses push notification, upload the push notification certificates to the server. You can upload the push certificates for your application with MobileFirst Operations Console. The certificates are common to all versions of an application.

   > **Note:** You might not be able to test the push notification for your app with production certificates before your app is published to the store.

6. Verify the following items before you publish the application to the store.
   - Test any mobile application management feature that you plan to use, such as disabling remote applications or displaying of an administrator message. For more information, see Mobile-application management (../using-console/#mobile-application-management).
   - In the case of an update, define the update strategy. For more information, see Updating MobileFirst apps in production.

Transferring server-side artifacts to a test or production server

You can transfer an application configuration from a one server to another by using command-line tools or a REST API.

The application descriptor file is a JSON file that contains the description and configuration of your application. When you run an app that connects to a MobileFirst Server instance, the app must be registered with that server and configured. After you define a configuration for your app, you can transfer the application descriptor to another server, for example to a test server or to a production server. After you transfer the application descriptor to the new server, the app is registered with the new server. Different procedures are available, depending on whether you develop mobile applications and have access to the code, or whether you administer servers and do not have access to the code of the mobile app.

> **Important:** If you import an application that includes authenticity data, and if the application itself has been recompiled since the authenticity data was generated, you must refresh the authenticity data. For more information, see Configuring the application-authenticity security check (../../authentication-and-security/application-authenticity/#configuring-application-authenticity).

- If you have access to the code of the mobile app, use the `mfpdev app pull` and `mfpdev app push` commands.
- If you do not have access to the code of the mobile app, use the administration service.

Jump to

- Transferring an application configuration by using mfpdev
- Transferring an application configuration with the administration service
- Transferring server-side artifacts by using the REST API
- Exporting and importing applications and adapters from the MobileFirst Operations Console

Transferring an application configuration by using mfpdev

After you have developed an application, you can transfer it from your development environment to a test or production environment.

- You must have an existing MobileFirst app on your local computer. The app must be registered to a MobileFirst Server. For information about creating a server profile, run **mfpdev app register**, or the topic about registering your type of app in the Developing applications section of this documentation.
- You must have connectivity from your local computer to the server that your app is currently registered to and to the server that you want to transfer your app to.
- You must have a server profile on the local computer for both the original MobileFirst Server and the server that you want to transfer your app to. For information about creating a server profile, run **mfpdev server add**.
- You must have the IBM MobileFirst Command Line Interface (CLI) installed.

You use the **mfpdev app pull** command to send a copy of the server-side configuration files for your app to your local computer. Then you use the **mfpdev app push** command to send it to another MobileFirst Server. The **mfpdev app push** command also registers the app on the specified server.

You can also use these commands to transfer a runtime configuration from one server to another.

The configuration information includes the contents of the application descriptor, which uniquely identifies the app to the server and other information that is specific to the app. The configuration files are provided as compressed files (.zip format). The .zip files are placed in the directory **appName**/**mobilefirst** and named as follows:

```
appID-platform-version-artifacts.zip
```

where **appID** is the application name, **platform** is one of **android**, **ios**, or **windows**, and version is the version level of your app. For Cordova apps, a separate .zip file is created for each target platform.

When you use the **mfpdev app push** command, the application's client properties file is modified to reflect the profile name and URL of the new MobileFirst Server.

1. On your development computer, navigate to a directory that is the root directory of your app or one of its subdirectories.
2. Run the **mfpdev app pull** command. If you specify the command with no parameters, the app is pulled from the default MobileFirst Server. You can also specify a particular server and its administrator password. For example, for an Android application named **myapp1**:

   ```
   cd myapp1
   mfpdev app pull Server10 -password secretPassword!
   ```

   This command finds the configuration files for the current application on the MobileFirst Server whose server profile is named Server10. Then, it sends the compressed file **myapp1-android-1.0.0-artifacts.zip**, which contains these configuration files, to the local computer and places it in the directory **myapp1**/**mobilefirst**.
3. Run the **mfpdev app push** command. If you specify the command with no parameters, the app is pushed to the default MobileFirst Server. You can also specify a particular server and its administrator password. For example, for the same application that was pushed in the previous step: `mfpdev app push Server12 -password secretPass234!`.

   This command sends the file **myapp1-android-1.0.0-artifacts.zip** to the MobileFirst Server whose server profile is named Server12, that has the administrator password **secretPass234!** The client properties file **myapp1**/**app**/**src**/**main**/**assets**/**mfpclient.properties** is modified to reflect that the server that the app is registered to is Server12, with the server's URL.

The app's server-side configuration files are present on the MobileFirst Server that you specified in the mfpdev app push command. The app is registered to this new server.

Transferring an application configuration with the administration service

As an administrator, you can transfer an application configuration from one server to another by using the administration service of MobileFirst Server. No access to the application code is required, but the client app must be built for the target server.

Before you begin

Build the client app for your target server. For more information, see Building an application for a test or production environment.

You download the application descriptor from the server where the application is configured and you deploy it to the new server. You can see the application descriptor with MobileFirst Operations Console.

1. Optional: Review the application descriptor in the server where the application server is configured. Open the MobileFirst Operations Console for that server, select your application version, and go to the **Configuration Files** tab.
2. Download the application descriptor from the server where the application is configured. You can download it by using the REST API or **mfpadm**.

   > **Note:** You can also export an application or application version from the MobileFirst Operations Console. See Exporting and importing applications and adapters from the MobileFirst Operations Console.

   - To download the application descriptor with the REST API, use the Application Descriptor (GET) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_application_descriptor_get.html?view=kc#Application-Descriptor--GET-) REST API.

     The following URL returns the application descriptor for the application of app ID **my.test.application**, for the **ios** platform, and version **0.0.1**. The call is made to MobileFirst Development Server: `http://localhost:9080/mfpadmin/management-apis/2.0/runtimes/mfp/applications/my.test.application/ios/0.0.1/descriptor`

     For example, you can use such URL with a tool like curl: `curl -user admin:admin http://[...]/ios/0.0.1/descriptor > desc.json`.

     Change the following elements of the URL according to your server configuration: * 9080 is the HTTP port of MobileFirst Development Server. * mfpadmin is the context root of the administration service. This context root is mfpadmin in MobileFirst Development Server.

     For information about the REST API, see REST API for the MobileFirst Server administration service.
   - Download the application descriptor by using **mfpadm**.

     The **mfpadm** program is installed when you run the MobileFirst Server installer. You start it from the **product_install_dir**/**shortcuts**/ directory, where **product_install_dir** indicates the installation directory of MobileFirst Server.

     The following example creates a password file, which is required by the **mfpadm** command, then downloads the application descriptor for the application of app ID **my.test.application**, for the **ios** platform, and version **0.0.1**. The provided URL is the HTTPS URL of MobileFirst Development Server.

```
echo password=admin > password.txt
mfpadm --url https://localhost:9443/mfpadmin --secure false --user admin \ --passwordfile password.txt \ app version mfp my.test.application ios 0.0.1 get descrip
tor > desc.json
rm password.txt
```

Change the following elements of the command line according to your server configuration: * 9443 is the HTTPS port of MobileFirst Development Server. * mfpadmin is the context root of the administration service. This context root is mfpadmin in MobileFirst Development Server. * --secure false indicates that the server's SSL certificate is accepted even if self-signed or if created for a different host name from the server's host name used in the URL.

For more information about the **mfpadm** program, see Administering MobileFirst applications through the command line (../using-cli).

3. Upload the application descriptor to the new server to register the app or update its configuration. You can upload it by using the REST API or **mfpadm**.
   - To upload the application descriptor with the REST API, use the Application (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_application_post.html?view=kc#Application--POST-) REST API.

   The following URL uploads the application descriptor to the mfp runtime. You send a POST request, and the payload is the JSON application descriptor. The call in this example is made to server that runs on the local computer and that is configured with an HTTP port set to 9081.

   ```
   http://localhost:9081/mfpadmin/management-apis/2.0/runtimes/mfp/applications/
   ```

   For example, you can use such URL with a tool like curl.

   ```
   curl -H "Content-Type: application/json" -X POST -d @desc.json -u admin:admin \ http://localhost:9081/mfpadmin/management-apis/2.0/runtimes/mfp/applications/
   ```

   - Upload the application descriptor by using mfpadm.

     The following example creates a password file, which is required by the mfpadm command, then uploads the application descriptor for the application of app ID my.test.application, for the ios platform, and version 0.0.1. The provided URL is the HTTPS URL of a server that runs on the local computer but is configured with an HTTPS port set to 9444, and for a runtime named mfp.

     ```
     echo password=admin > password.txt
     mfpadm --url https://localhost:9444/mfpadmin --secure false --user admin \ --passwordfile password.txt \ deploy app mfp desc.json
     rm password.txt
     ```

Transferring server-side artifacts by using the REST API

Whatever your role, you can export applications, adapters, and resources for back-up or reuse purposes by using the MobileFirst Server administration service. As an administrator or deployer, you can also deploy an export archive to a different server. No access to the application code is required, but the client app must be built for the target server.

Before you begin

Build the client app for your target server. For more information, see Building an application for a test or production environment.

The export API retrieves the selected artifacts for a runtime as a .zip archive. Use the deployment API to reuse archived content.

> **Important:** Carefully consider your use case:
>
> - The export file includes the application authenticity data. That data is specific to the build of a mobile app. The mobile app includes the URL of the server and its runtime name. Therefore, if you want to use another server or another runtime, you must rebuild the app. Transferring only the exported app files would not work.
> - Some artifacts might vary from one server to another. Push credentials are different depending on whether you work in a development or production environment.
> - The application runtime configuration (that contains the active/disabled state and the log profiles) can be transferred in some cases but not all.
> - Transferring web resources might not make sense in some cases, for example if you rebuild the app to use a new server.

- To export all resources, or a selected subset of resources, for one adapter or for all adapters, use the Export adapter resources (GET) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_export_adapter_resources_get.html?view=kc) or Export adapters (GET) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_export_adapters_get.html?view=kc) API.
- To export all the resources under a specific application environment (such as Android or iOS), that is all the versions and all the resources for the version for that environment, use the Export application environment (GET) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_export_application_environment_get.html?view=kc) API.
- To export all the resources for a specific version of an application (for example, version 1.0 or 2.0 of an Android application), use the Export application environment resources (GET) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_export_application_environment_resources_get.html?view=kc) API.
- To export a specific application or all applications for a runtime, use the Export applications (GET) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_export_applications_get.html?view=kc) or Export application resources (GET) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_export_application_resources_get.html?view=kc) API. **Note:** Credentials for push notification are not exported among the application resources.
- To export the adapter content, descriptor, license configuration, content, user configuration, keystore, and web resources of an application, use the Export resources (GET) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_export_resources_get.html?view=kc#Export-resources--GET-) API.

- To export all or selected resources for a runtime, use the Export runtime resources (GET) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_export_runtime_resources_get.html?view=kc) API. For example, you can use this general curl command to retrieve all resources as a .zip file.

  ```
  curl -X GET -u admin:admin -o exported.zip
  "http://localhost:9080/worklightadmin/management-apis/2.0/runtimes/mfp/export/all"
  ```

- To deploy an archive that contains such web application resources as adapter, application, license configuration, keystore, web resource, use the Deploy (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_deploy_post.html?view=kc) API. For example, you can use this curl command to deploy an existing .zip file that contains artifacts.

  ```
  curl -X POST -u admin:admin -F
  file=@/Users/john_doe/Downloads/export_applications_adf_ios_2.zip
  "http://localhost:9080/mfpadmin/management-apis/2.0/runtimes/mfp/deploy/multi"
  ```

- To deploy application authenticity data, use the Deploy Application Authenticity Data (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_deploy_application_authenticity_data_post.html?view=kc) API.
- To deploy the web resources of an application, use the Deploy a web resource (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_deploy_a_web_resource_post.html?view=kc) API.

If you deploy an export archive to the same runtime, the application or version is not necessarily restored as it was exported. That is, the redeployment does not remove subsequent modifications. Rather, if some application resources are modified between export time and redeployment, only the resources that are included in the exported archive are redeployed in their original state. For example, if you export an application with no authenticity data, then you upload authenticity data, and then you import the initial archive, the authenticity data is not erased.

Exporting and importing applications and adapters from the MobileFirst Operations Console

From the console, under certain conditions, you can export an application or one of its versions, and later import it to a different runtime on the same server or a different server. You can also export and reimport adapters. Use this capability for reuse or back-up purposes.

If you are granted the **mfpadmin** administrator role and the **mfpdeployer** deployer role, you can export one version or all versions of an application. The application or version is exported as a .zip compressed file, which saves the application ID, descriptors, authenticity data, and web resources. You can later import the archive to redeploy the application or version to another runtime on the same or on a different server.

> **Important:** Carefully consider your use case:
>
> - The export file includes the application authenticity data. That data is specific to the build of a mobile app. The mobile app includes the URL of the server and its runtime name. Therefore, if you want to use another server or another runtime, you must rebuild the app. Transferring only the exported app files would not work.
> - Some artifacts might vary from one server to another. Push credentials are different depending on whether you work in a development or production environment.
> - The application runtime configuration (that contains the active/disabled state and the log profiles) can be transferred in some cases but not all.
> - Transferring web resources might not make sense in some cases, for example if you rebuild the app to use a new server.

You can also transfer application descriptors by using the REST API or the mfpadm tool. For more information, see Transferring an application configuration with the administration service.

1. From the navigation sidebar, select an application or application version, or an adapter.
2. Select **Actions → Export Application** or **Export Version** or **Export Adapter**.

   You are prompted to save the .zip archive file that encapsulates the exported resources. The aspect of the dialog box depends on your browser and the target folder depends on your browser settings.
3. Save the archive file.

   The archive file name includes the name and version of the application or adapter, for example **export*applications*com.sample.zip**.
4. To reuse an existing export archive, select **Actions → Import Application** or **Import Version**, browse to the archive, and click **Deploy**.

The main console frame displays the details of the imported application or adapter.

If you import to the same runtime, the application or version is not necessarily restored as it was exported. That is, the redeployment at import time does not remove subsequent modifications. Rather, if some application resources are modified between export time and redeployment at import time, only the resources that are included in the exported archive are redeployed in their original state. For example, if you export an application with no authenticity data, then you upload authenticity data, and then you import the initial archive, the authenticity data is not erased.

Updating MobileFirst apps in production

There are general guidelines for upgrading your MobileFirst apps when they are already in production, on the Application Center or in app stores.

When you upgrade an app, you can deploy a new app version and leave the old version working, or deploy a new app version and block the old version. In the case of an app developed with Apache Cordova, you can also consider only updating the Web resources.

Deploying a new app version and leaving the old version working

The most common upgrade path, used when you introduce new features or modify native code, is to release a new version of your app. Consider following these steps:

1. Increment the app version number.
2. Build and test your application. For more information, see Building an application for a test or production environment.
3. Register the app to MobileFirst Server and configure it.
4. Submit the new .apk, .ipa, .appx, or .xap files to their respective app stores.
5. Wait for review and approval, and for the apps to become available.

6. Optional - send notification message to users of the old version, announcing the new version. See Displaying an administrator message (../using-console/#displaying-an-administrator-message) and Defining administrator messages in multiple languages (../using-console/#defining-administrator-messages-in-multiple-languages).

Deploying a new app version and blocking the old version

This upgrade path is used when you want to force users to upgrade to the new version, and block their access to the old version. Consider following these steps:

1. Optional - send notification message to users of the old version, announcing a mandatory update in a few days. See Displaying an administrator message (../using-console/#displaying-an-administrator-message) and Defining administrator messages in multiple languages (../using-console/#defining-administrator-messages-in-multiple-languages).
2. Increment the app version number.
3. Build and test your application. For more information, see Building an application for a test or production environment.
4. Register the app to MobileFirst Server and configure it.
5. Submit the new .apk, .ipa, .appx, or .xap files to their respective app stores.
6. Wait for review and approval, and for the apps to become available.
7. Copy links to the new app version.
8. Block the old version of the app in MobileFirst Operations Console, supplying a message and link to the new version. See Remotely disabling application access to protected resources (../using-console/#remotely-disabling-application-access-to-protected-resources).

> **Note:** If you disable the old app, it is no longer able to communicate with MobileFirst Server. Users can still start the app and work with it offline unless you force a server connection on app startup.

Direct Update (no native code changes)

Direct Update is a mandatory upgrade mechanism that is used to deploy fast fixes to a production app. When you redeploy an app to MobileFirst Server without changing its version, MobileFirst Server directly pushes the updated web resources to the device when the user connects to the server. It does not push updated native code. Things to keep in mind when you consider a Direct Update include:

1. Direct Update does not update the app version. The app remains at the same version, but with a different set of web resources. The unchanged version number can introduce confusion if used for the wrong purpose
2. Direct Update also does not go through the app store review process because it is technically not a new release. This should not be abused because vendors can become displeased if you deploy a whole new version of your app that bypasses their review. It is your responsibility to read each store's usage agreements and abide by them. Direct Update is best used to fix urgent issues that cannot wait for several days.
3. Direct Update is considered a security mechanism, and therefore it is mandatory, not optional. When you initiate the Direct Update, all users must update their app to be able to use it.
4. Direct Update does not work if an application is compiled (built) with a different version of IBM MobileFirst Foundation than the one that was used for the initial deployment.

Last modified on