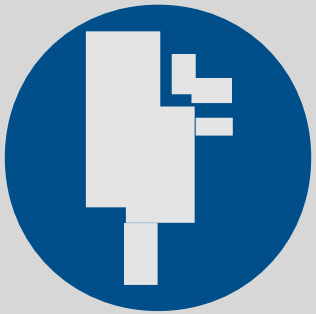


Worklight Overview

Andrew Ferrier
andrew.ferrier@uk.ibm.com

Matt Perrins
matthew_perrins@uk.ibm.com

WebSphere User Group, March 2012

Worklight
An IBM® company

The story until now...

- Mobile apps, both pure-web-based and hybrid, are becoming popular in the enterprise space.
- IBM's main focus in this area has been the Web 2.0 and Mobile Feature Pack for WebSphere, specifically Dojo Mobile with HTML5/CSS3
- Now, Worklight is in the picture to round out the picture... continuing the same technological base.
- Let's talk about Worklight...

The mobile lifecycle

- Strong demand by LoB
- Higher expectations of user experience with mobile apps
- Lack of best practices guidance on how to deliver mobile applications
- More direct involvement from users/stakeholders in design
- Native programming models are not portable across devices.
- Highly fragmented set of mobile devices and platforms
- Very large number of configurations of devices, platforms, carriers, etc. to test
- Mobile landscape evolves at a much faster pace
- More frequent releases and updates for apps with more urgent time-to-market demands

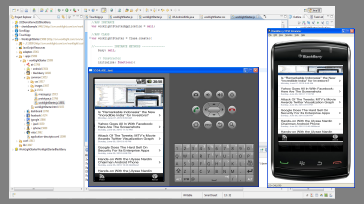
Worklight
An IBM® company

Worklight Vision

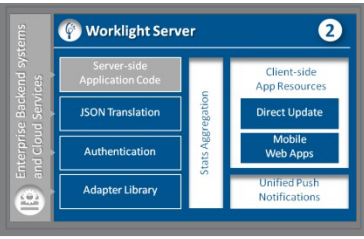
Provide the best platform in the market for enterprises to develop, run and manage smartphone and tablet apps

- Open platform, built around HTML5 & supporting hybrid apps
- Focus on new devices and OS's and their unique capabilities
- Cater to high-end enterprise needs regarding app capabilities, delivery, integration, security, scale and management
- Handle the entire lifecycle of mobile apps

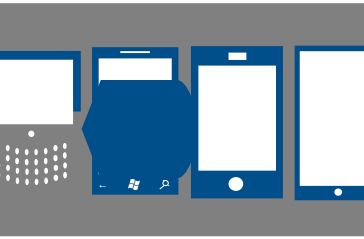
Worklight Overview



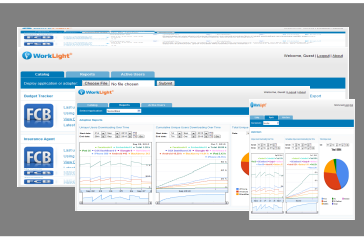
Worklight Studio
The most complete, extensible environment with maximum code reuse and per-device optimization



Worklight Server
Unified notifications, runtime skins, version management, security, integration and delivery

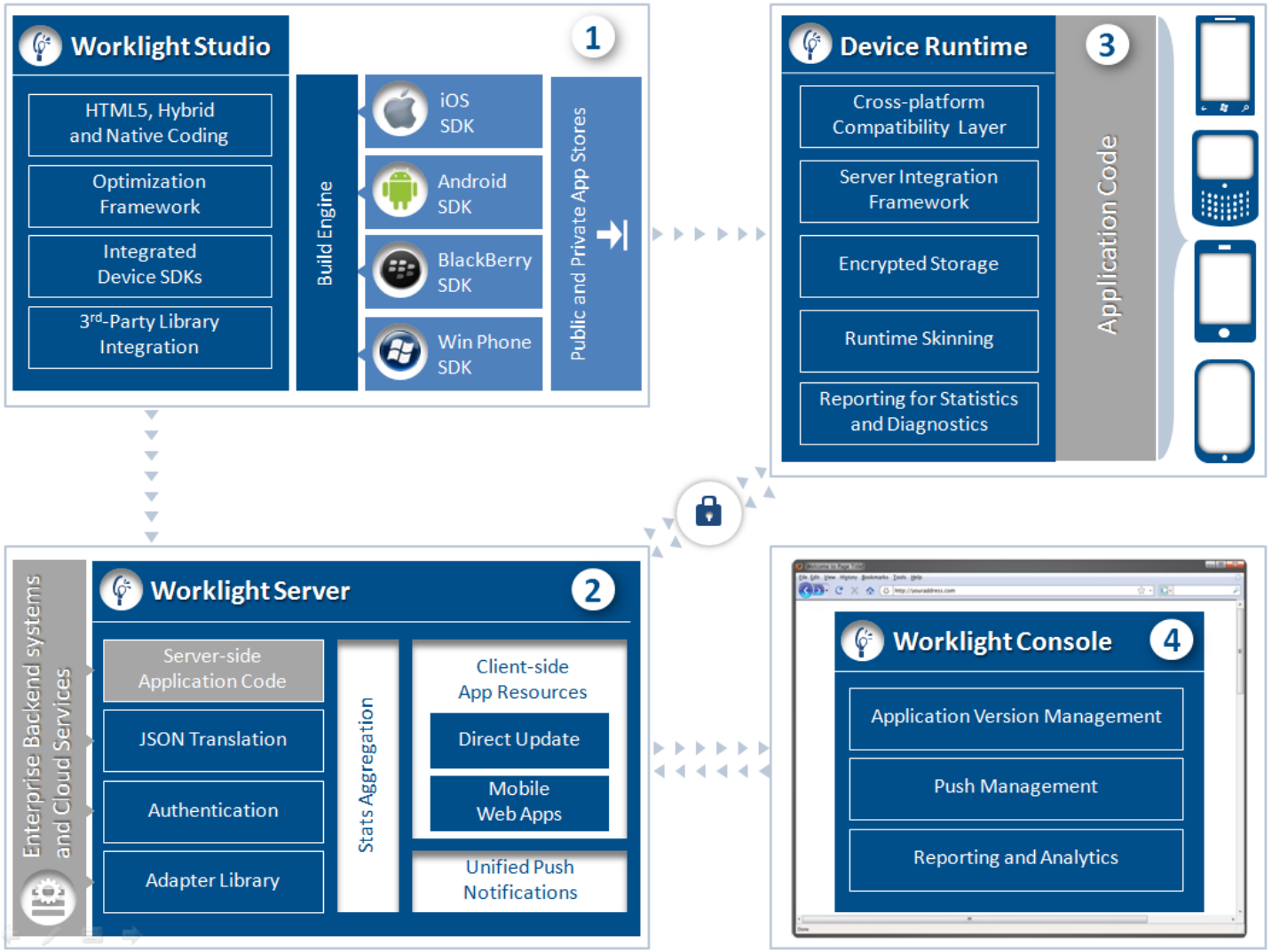


Worklight Runtime Components
Extensive libraries and client APIs that expose and interface with native device functionality



Worklight Console
A web-based console for real-time analytics and control of your mobile apps and infrastructure

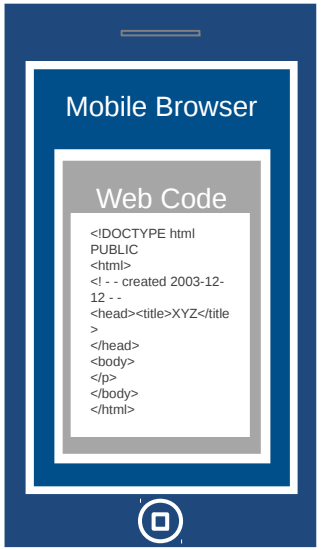
Worklight Architecture



Worklight Application Types

Browser Access

Written in HTML5 JavaScript and CSS3. Quick and cheap to develop, but less powerful than native.



Browser Access

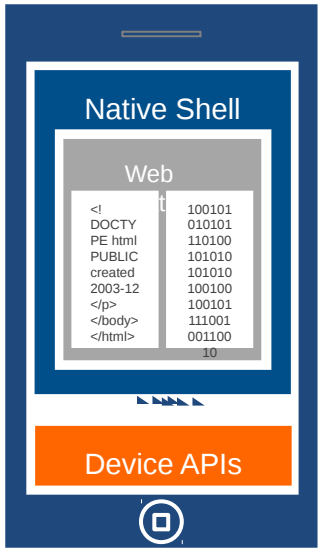
Hybrid Apps - Web

HTML5 code and Worklight runtime libraries packaged within the app and executed in a native shell.

Downloadable

Hybrid Apps - Mixed

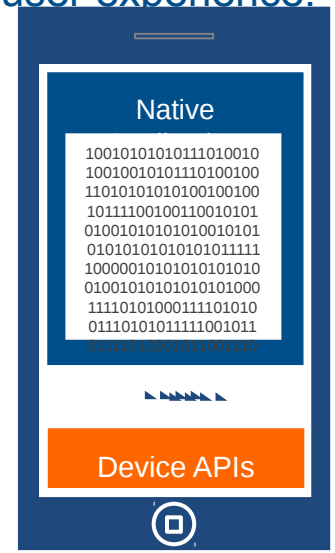
User augments web code with native language for unique needs and maximized user experience.



Downloadable

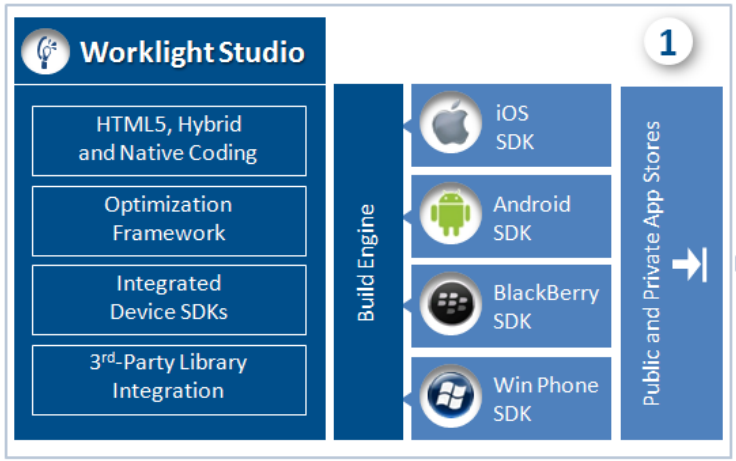
Native Apps

Platform-specific. Requires unique expertise, pricy and long to develop. Can deliver higher user experience.

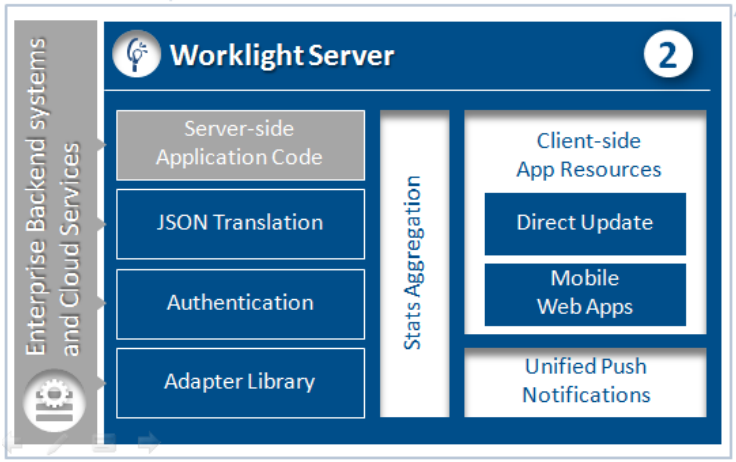


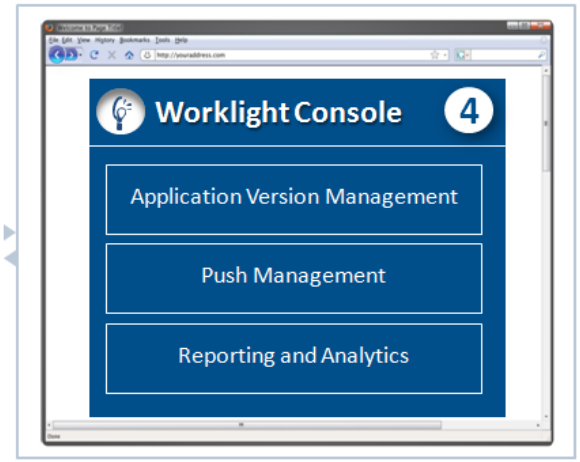
Downloadable

Worklight Studio

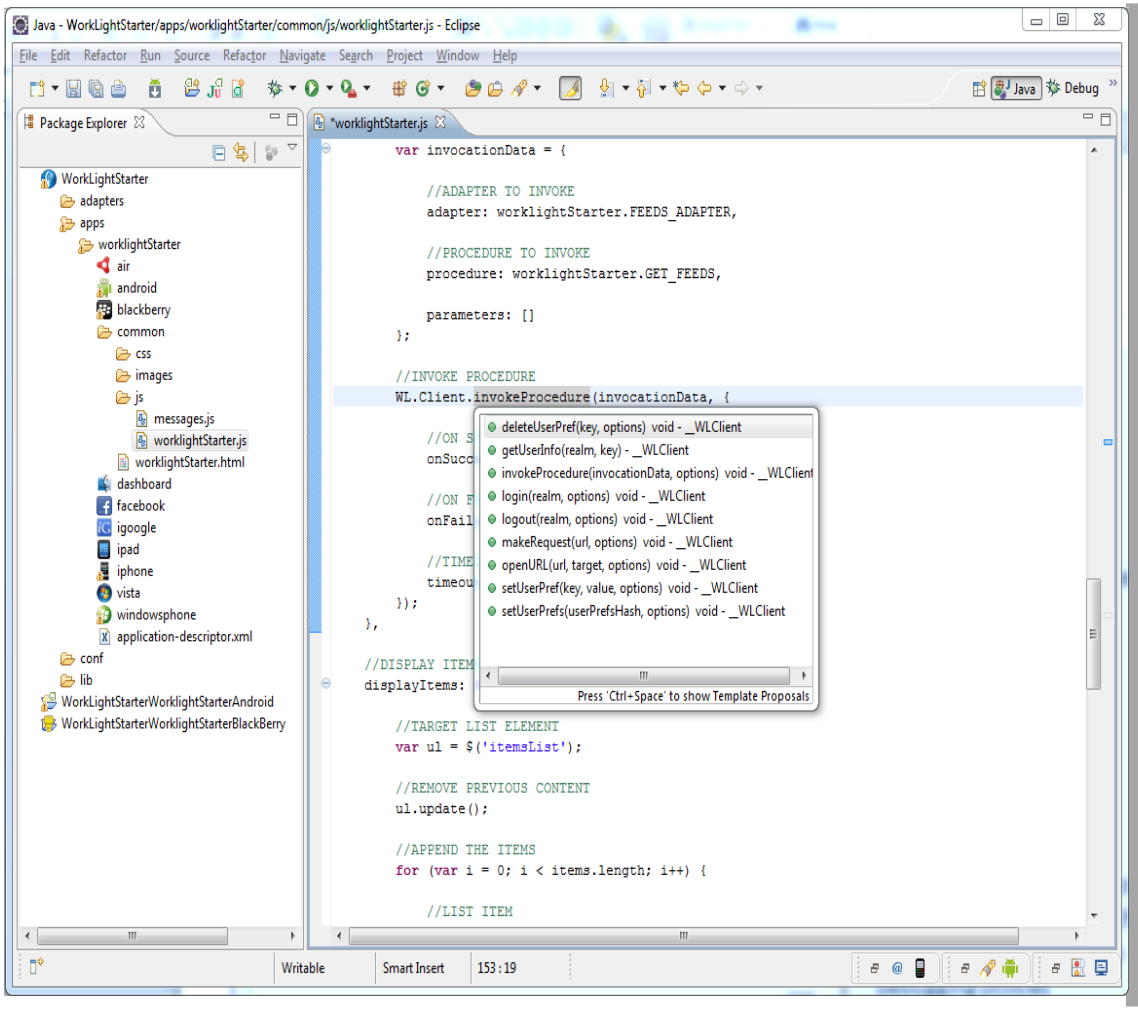


- Eclipse-based IDE
- Combining native and standard web technologies in one multiplatform app
- Environment-specific optimization
- 3rd-party libraries integration
- Device SDK integration
- Back-end connectivity utilities





Worklight Studio



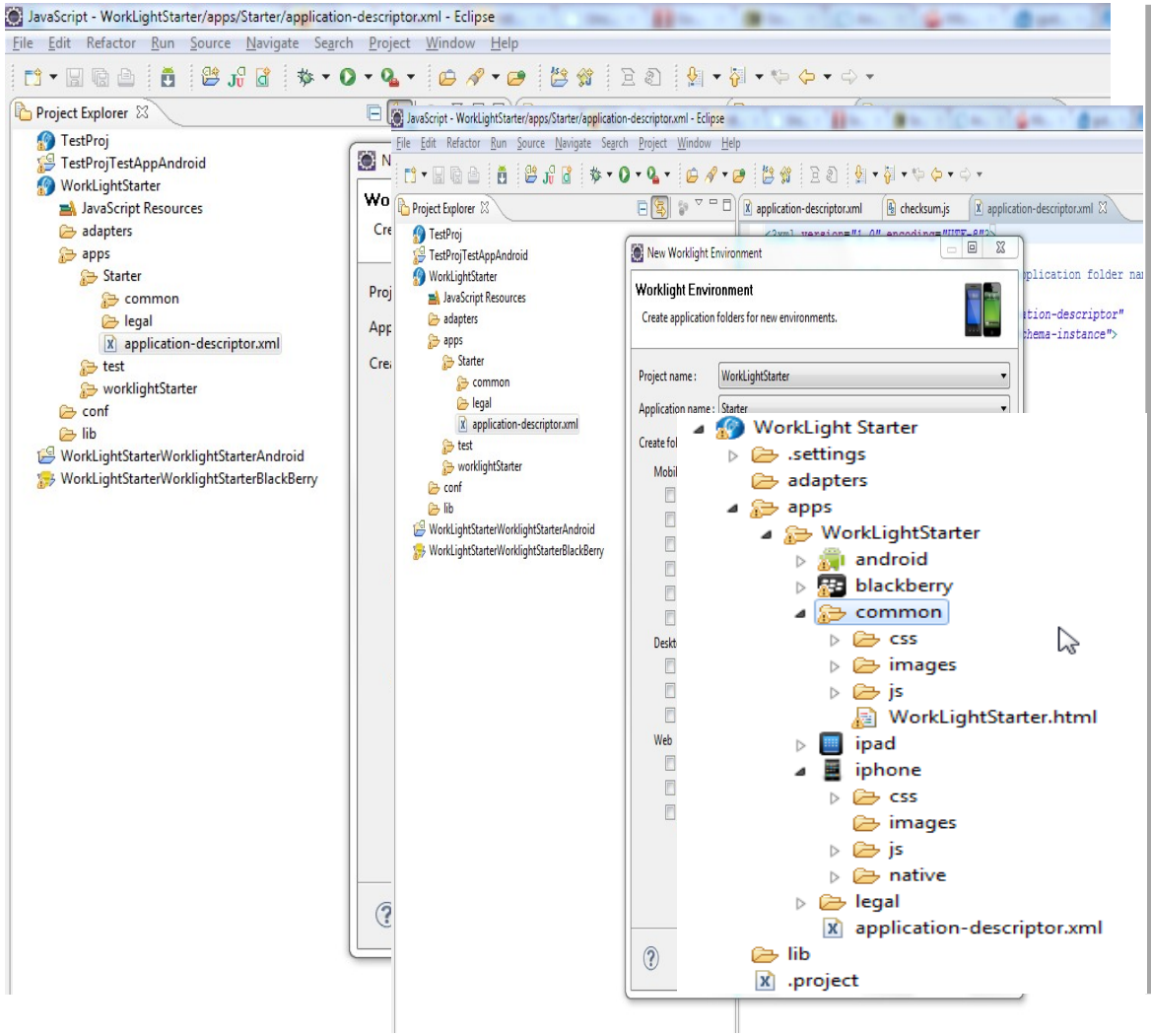
Integrated Development Environment
(Eclipse Plug-in)

Application development using native and/or familiar web technologies:

- HTML5
- CSS3
- JavaScript

Integrated device SDKs allow direct access from within the IDE to emulators and code debugging utilities

Single Shared Codebase

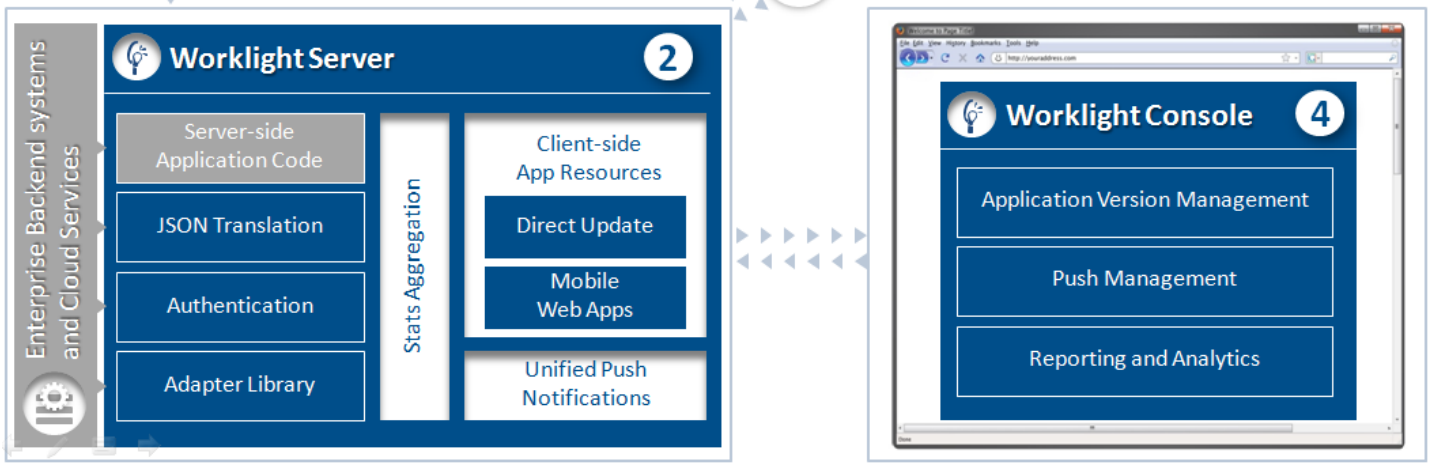


Project structure separates device-specific and device-independent “common” code.

Build procedure assembles and deploys chosen device-specific variants

Worklight Server

- Distribution of mobile web apps
- Enterprise connectivity:
  - Secure client/server connectivity
  - Direct access to enterprise back-end data and transaction capabilities
  - Authentication enforcement
- Client control:
  - Application version management and remote disabling
  - Direct update of application code
- Unified Push Notifications
- Aggregation of usage statistics

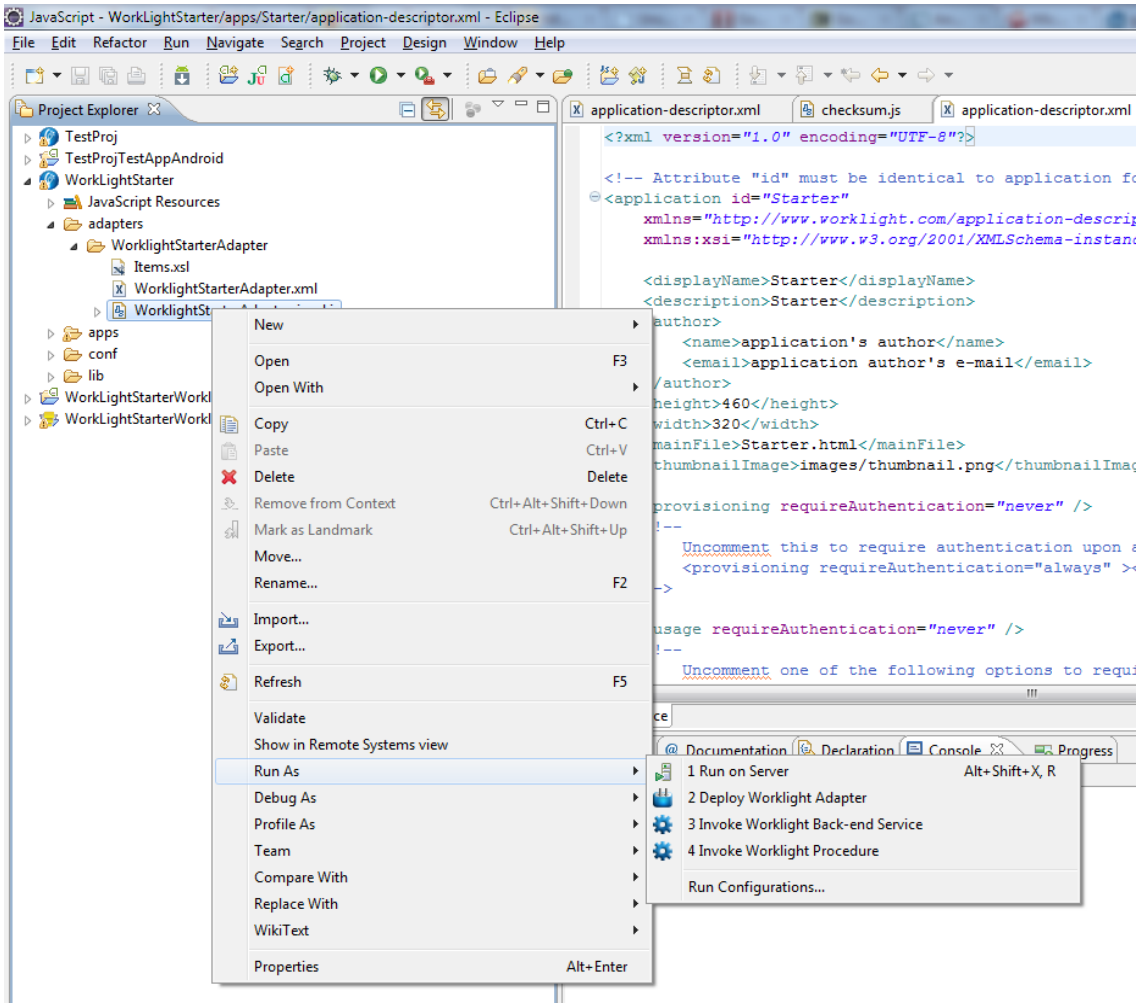


Worklight
An IBM® company

Direct Update – On-device Logic

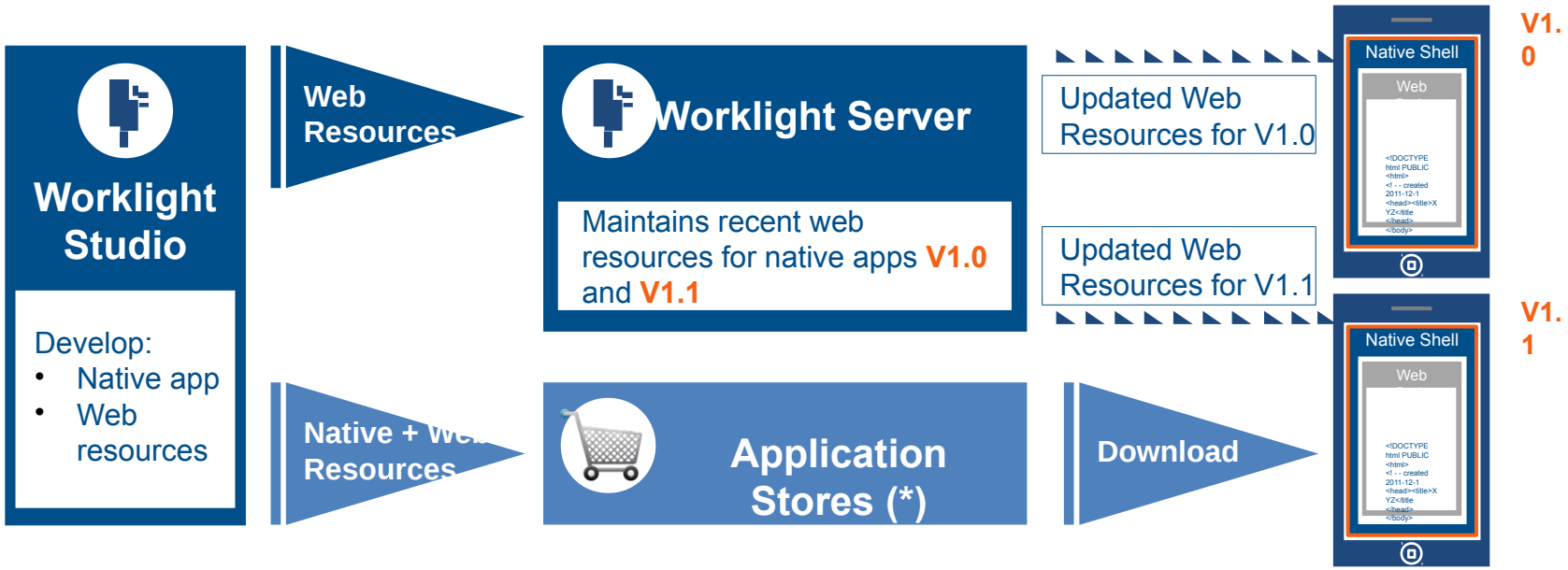
- Web resources packaged with app to ensure initial offline availability
- Web resources transferred to app's cache storage
- App checks for updates
  - On startup
  - On foreground
- Updated web resources downloaded when necessary

Enterprise Connectivity: Adapters



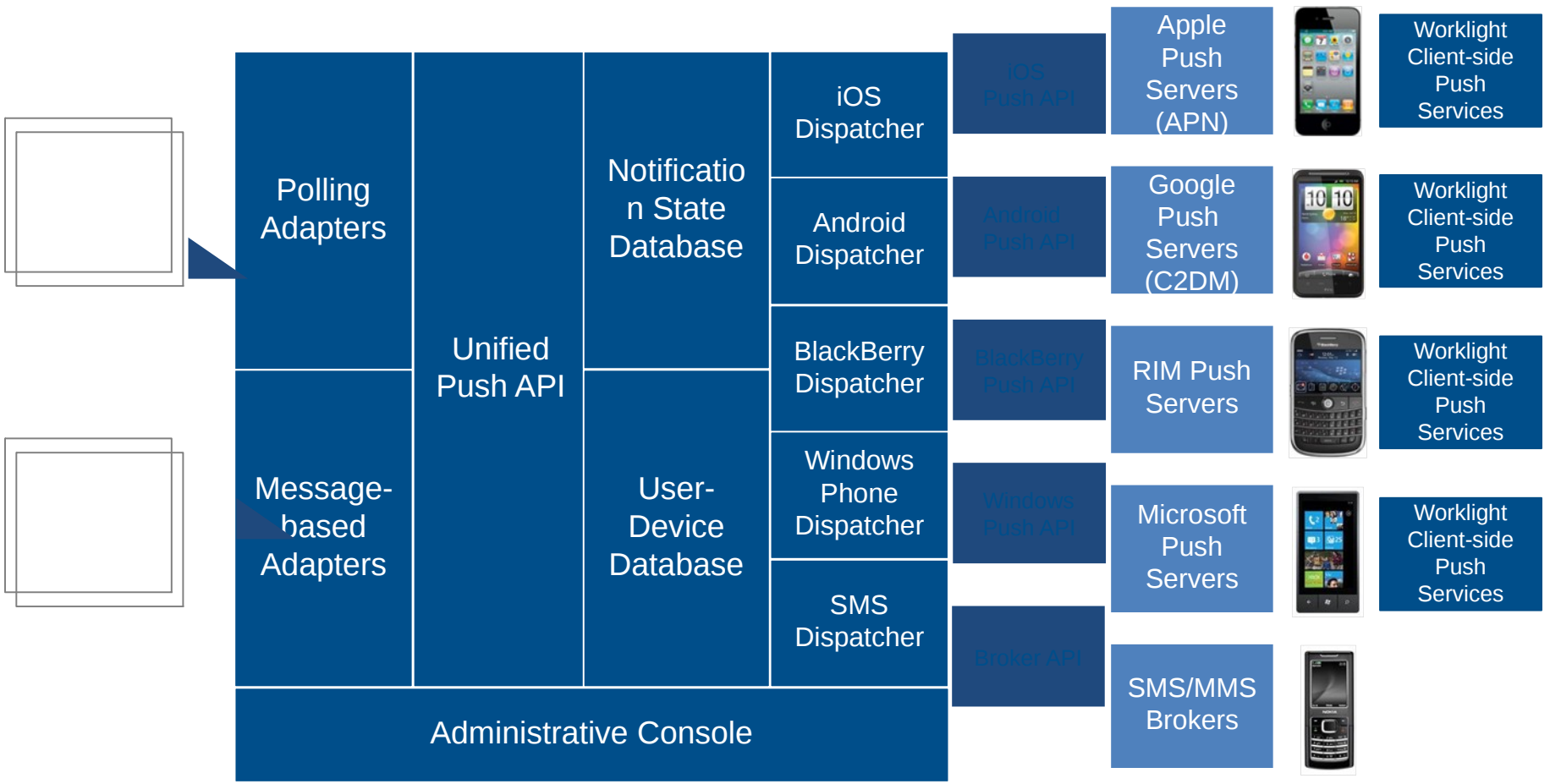
- Secure back-end integration
- XML-based declarative specification
- Multi-source data mashups
- Eclipse plug-in supporting auto-complete and validation
- Simplified adapter testing
- Server-side debugging
- Web services and JDBC integration
- Access to session data and user properties

Direct Update - Distribution



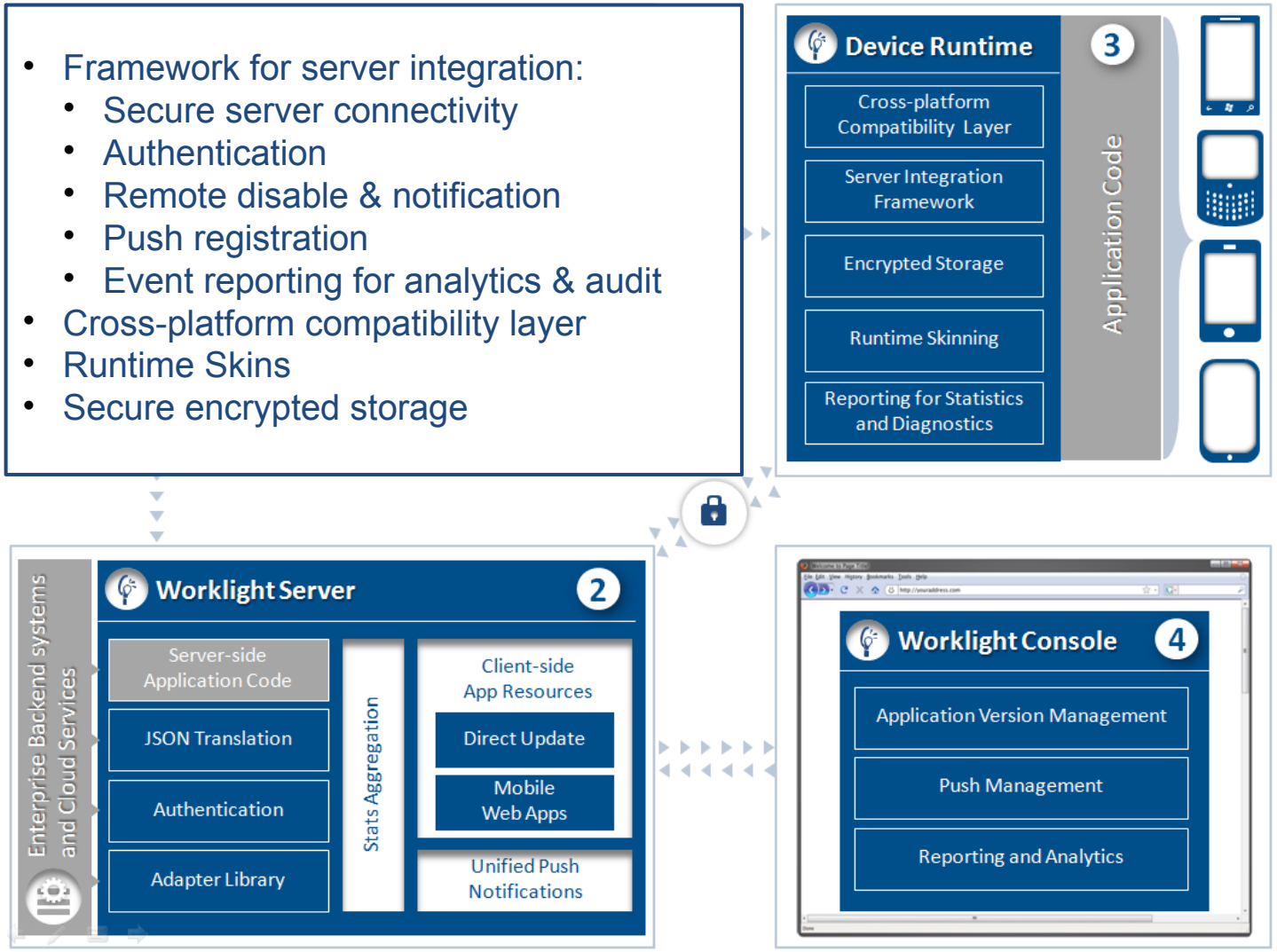
(*) During development cycles, testers automatically get recent web resources via internal distribution mechanisms and not application stores.

Unified Push Notifications



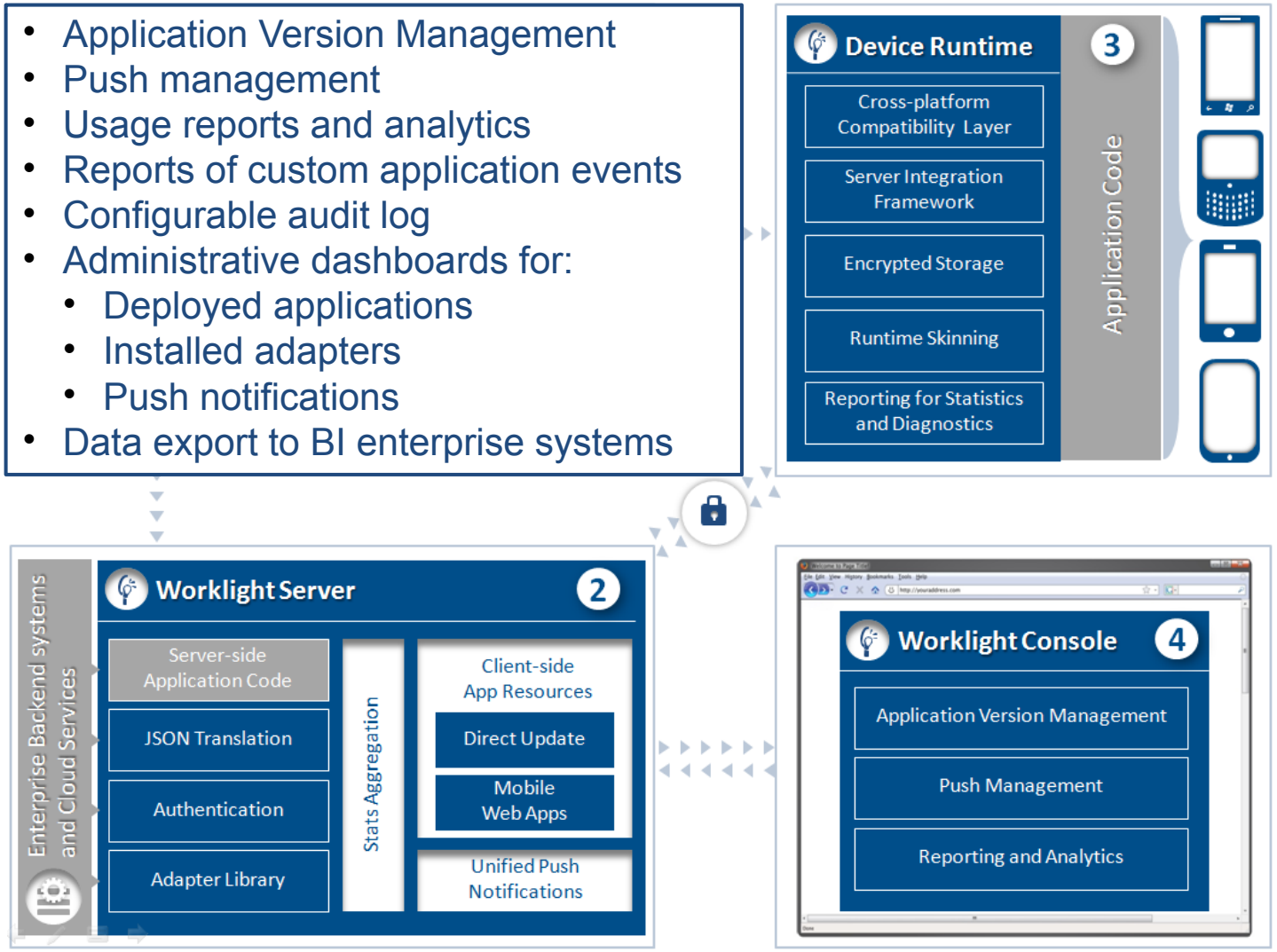
Device Runtime Components

- Framework for server integration:
  - Secure server connectivity
  - Authentication
  - Remote disable & notification
  - Push registration
  - Event reporting for analytics & audit
- Cross-platform compatibility layer
- Runtime Skins
- Secure encrypted storage

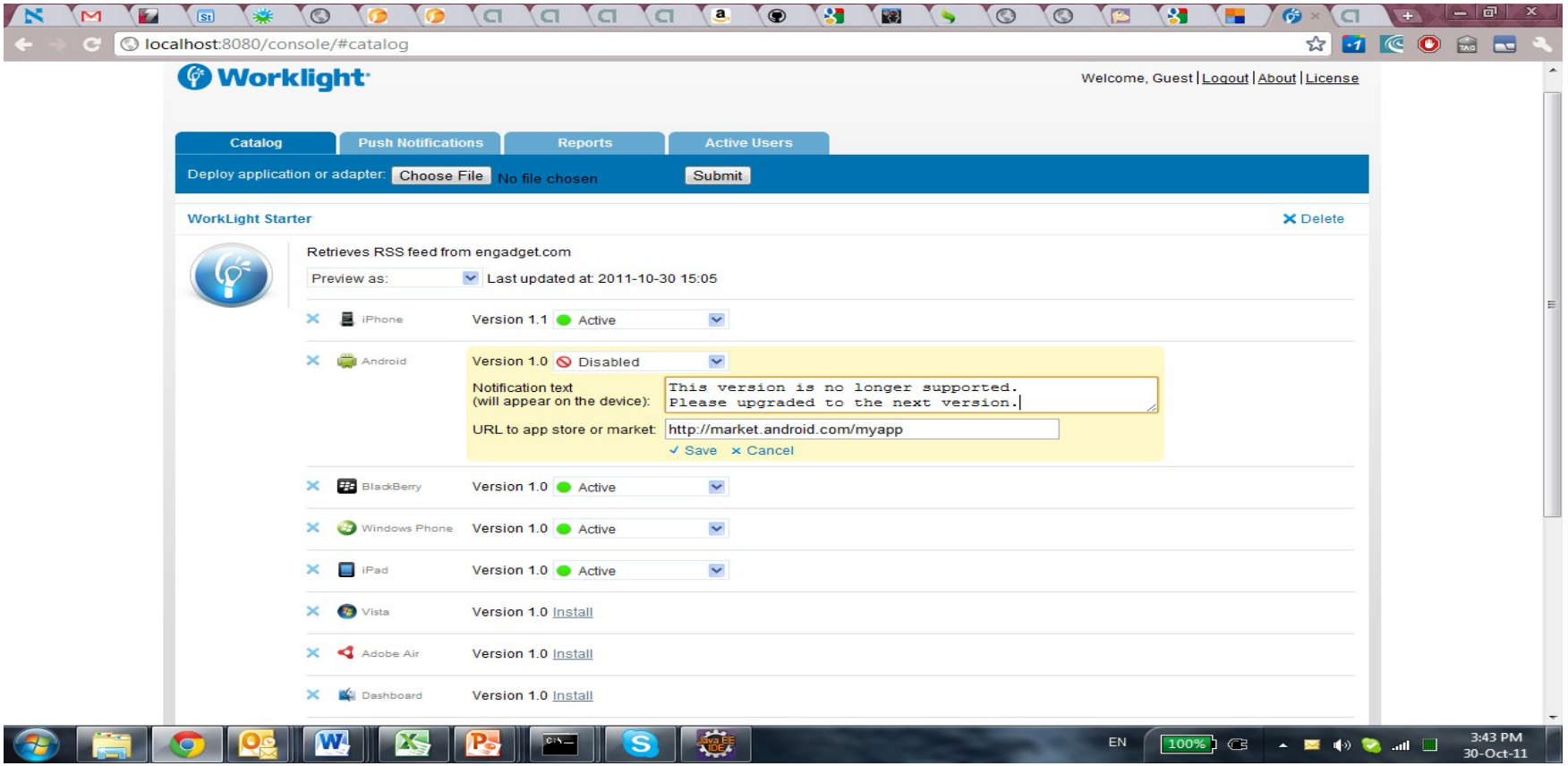


Worklight Console

- Application Version Management
- Push management
- Usage reports and analytics
- Reports of custom application events
- Configurable audit log
- Administrative dashboards for:
  - Deployed applications
  - Installed adapters
  - Push notifications
- Data export to BI enterprise systems

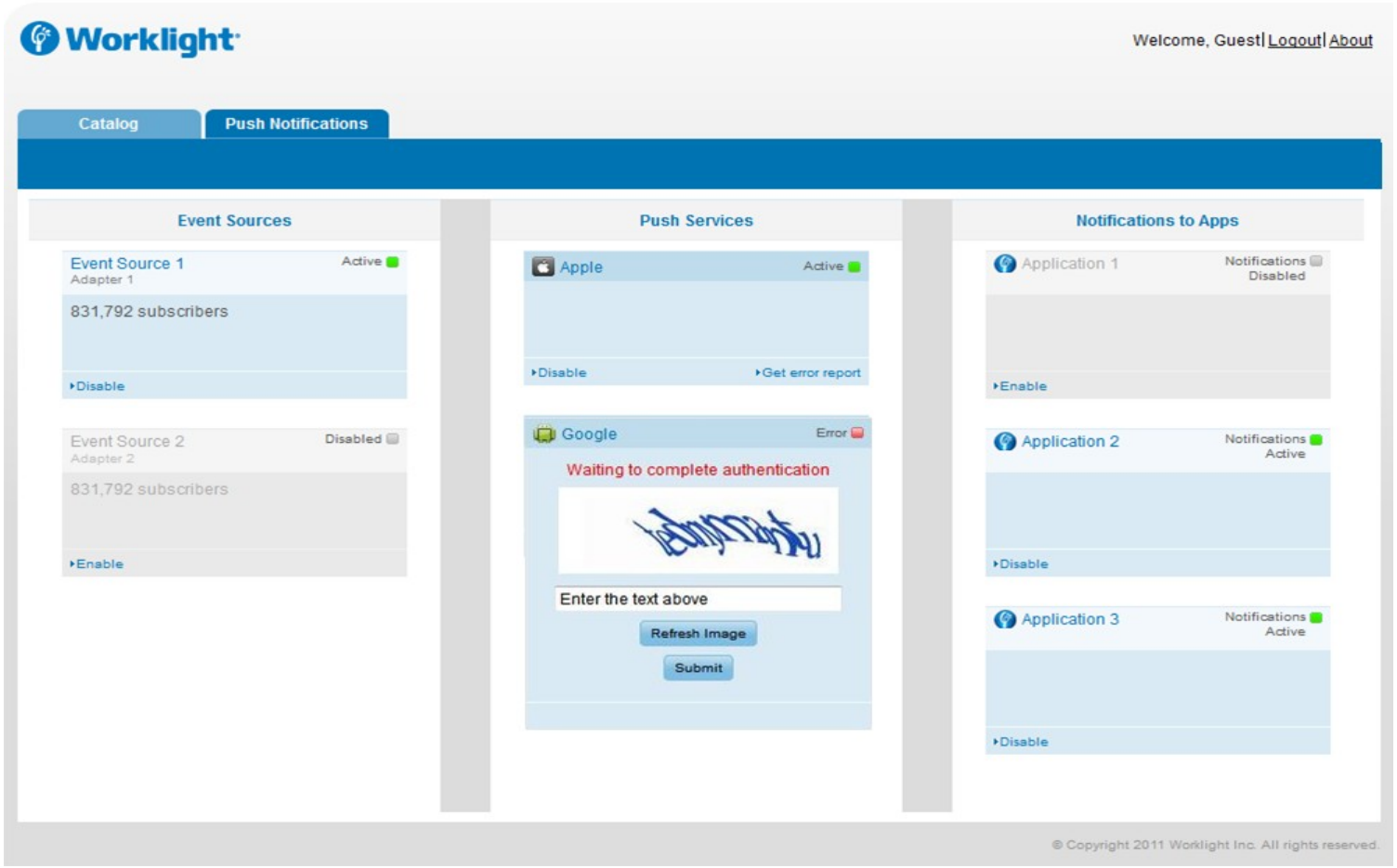


Dynamic Control of Deployed Apps

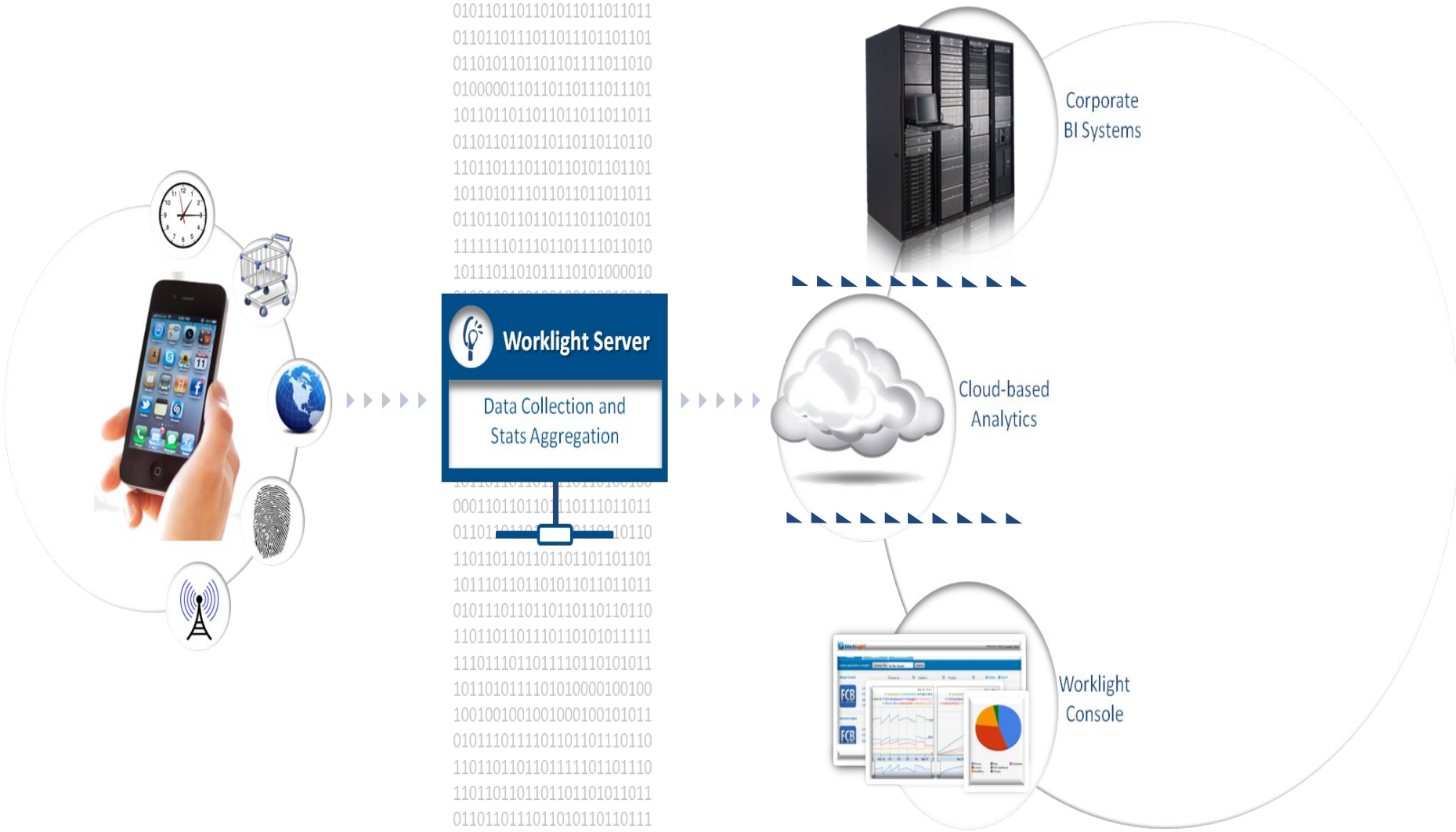


- Centralized control of all installed applications and adapters
- Remotely disable apps by device and version
- Customize user messages

Push Services Management



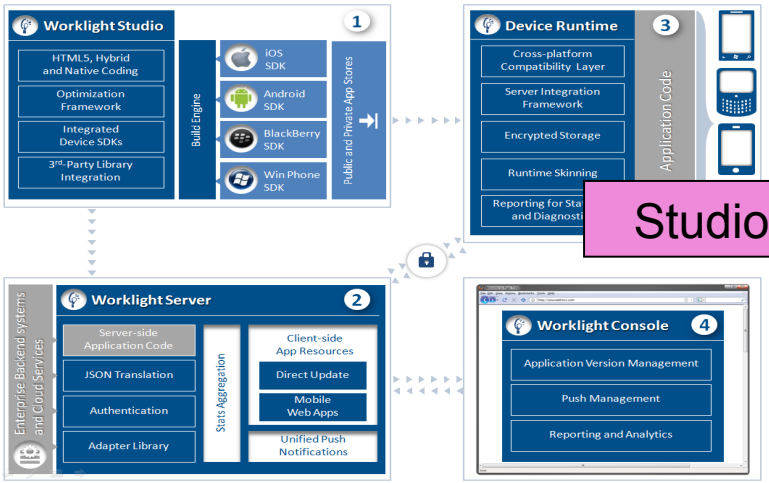
Data Collection and Analytics

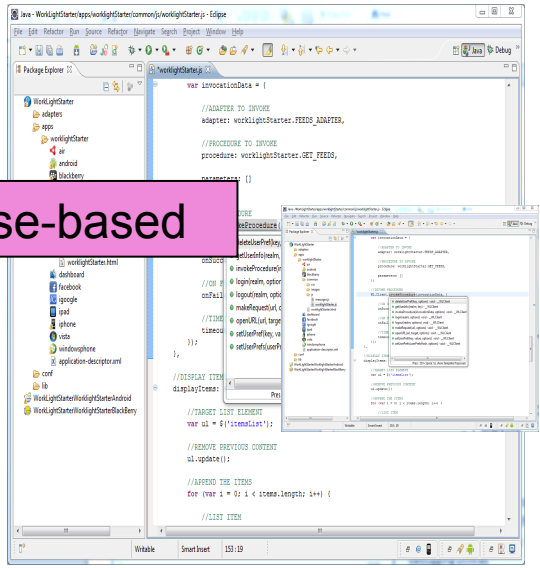


Worklight Studio

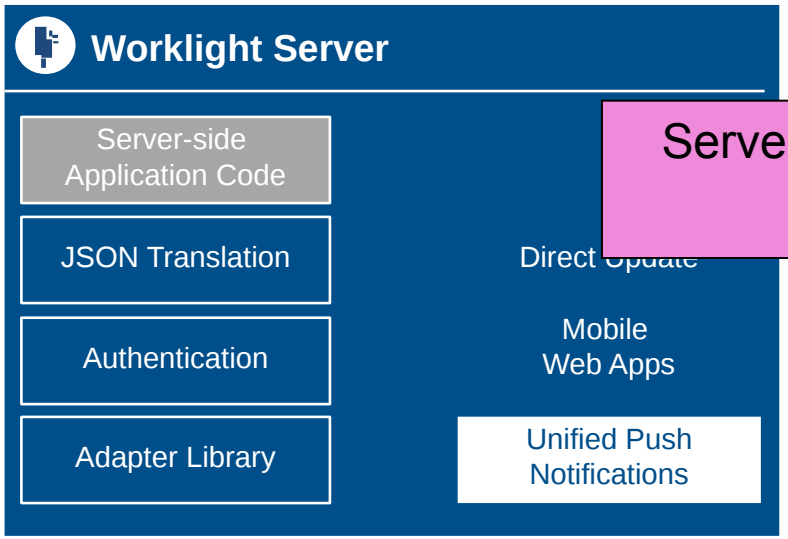
Writing Your First Application

Development: Server + Studio

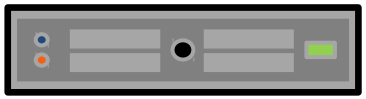




Studio is Eclipse-based

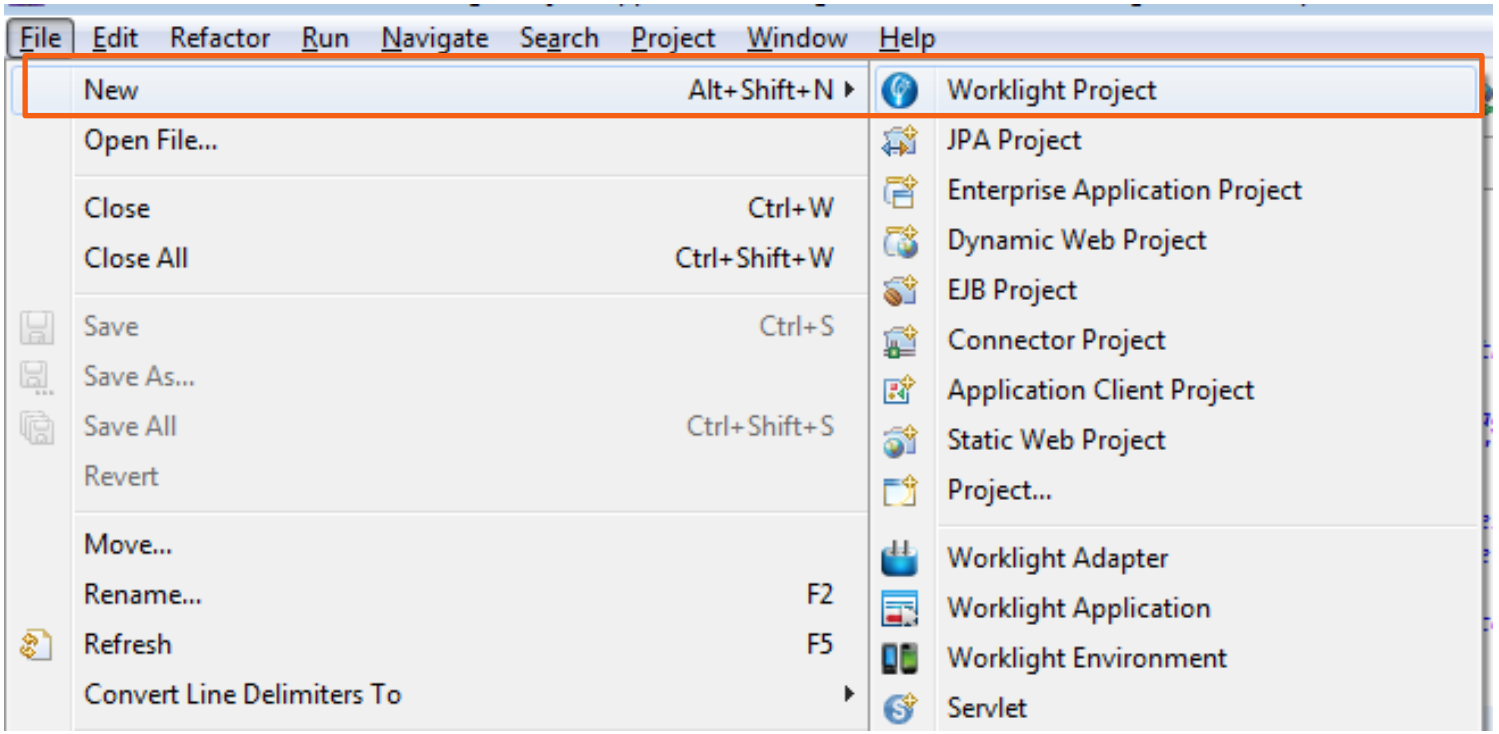


Server runs on Tomcat (today)
wl_start to launch



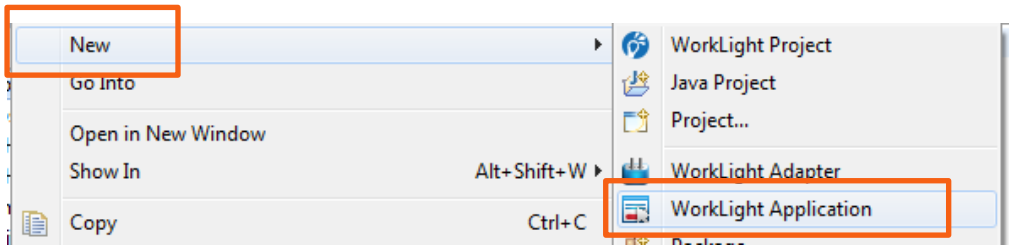
Hello Worklight Application

- Create a Worklight Project

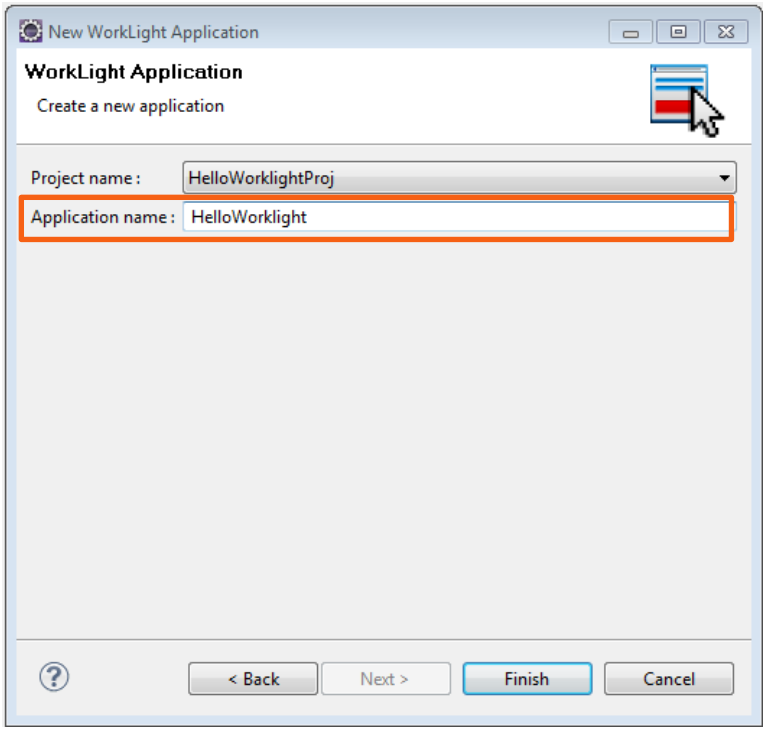


Hello Worklight Application

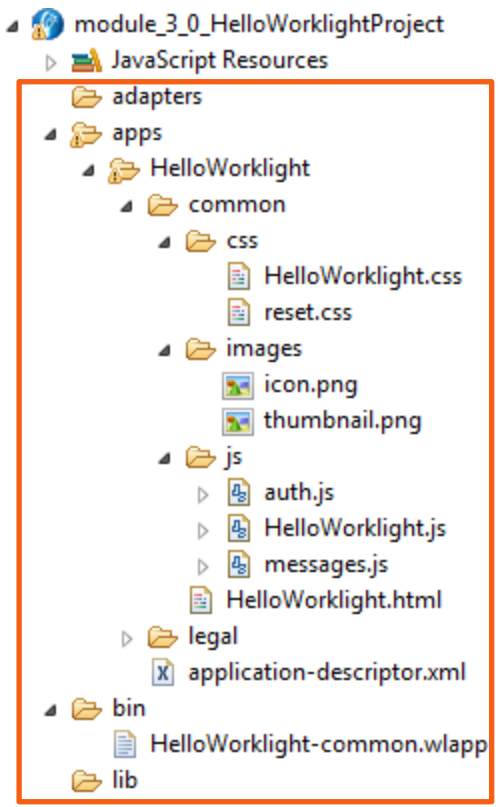
- Right Click on the project node and create a new Worklight Application



- Name it HelloWorklight

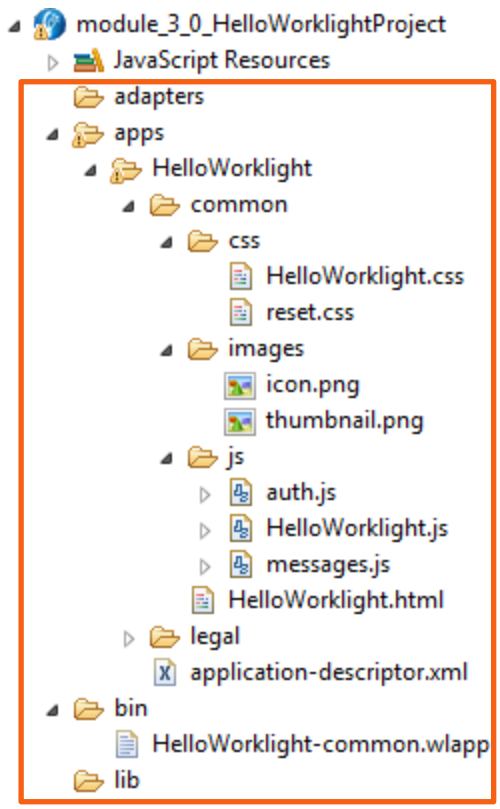


Worklight Project Structure



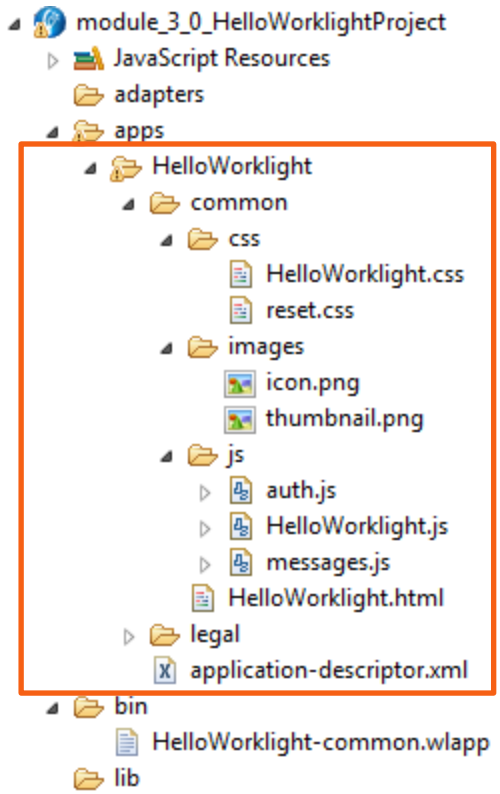
- A Worklight Project consists of the following folders:
  - **adapters:**
    - Contains the project's adapters
    - Right-click → Create Adapter to create a new one
  - **apps:**
    - Contains the project's applications
    - Right-click → Create Application to create a new one
  - **bin:**
    - A destination folder for deliverables
  - **lib:**
    - Contains the project's 3rd-party libraries

HelloWorklight - Project Structure



- The default environment is called ***common***
- The ***common*** environment contains all the resources that are shared between environments
- You can add new environments by right-clicking application folder and choosing the ***Add new environment*** option
  - A new environment will be created.
  - The new environment's resources will have the following relationship with the common resources:
    - ***images*** - override the ***common*** images in case both share the same name
    - ***css*** – extend and/or override the ***common*** CSS files
    - ***js*** - extends the ***common*** application instance JS object
    - (The environment class extends the common app class)
    - ***HTML -*** override the common HTML code in case both share the same name

HelloWorklight - Project Structure



- Common Environment:
  - **HelloWorklight.html**
    - The main HTML file.
  - **css**
    - ***HelloWorklight.css*** - main application's CSS file.
    - ***reset.css*** - bringing all rendering engines to one common ground.
  - **images**
    - Default Worklight images for the common environment
  - **js**
    - ***HelloWorklight.js***
      - The main JavaScript file for the application.
    - ***messages.js***
      - JSON object holding all app messages. Can be used for localization
    - **Auth.js**
      - Application's custom authentication mechanism implementation
- Legal
  - This folder should hold all legal related docs.
- Application-descriptor
  - Application's meta data

HelloWorklight - Application Descriptor

- An **XML** file to hold all the application's meta data
- Based on the W3C Widget Packaging and Configuration

Specify the application name, description and author's name to be displayed in the Worklight Console

```
3  <!-- Attribute "id" must be identical
4  <application id="HelloWorklight" plat
5      xmlns="http://www.worklight.com/a
6      xmlns:xsi="http://www.w3.org/2001/
7
8      <displayName>HelloWorklight</displayName>
9      <description>HelloWorklight</description>
10     <author>
11         <name>application's author</name>
12         <email>application author's e-mail</email>
13     </author>
14     <height>460</height>
15     <width>320</width>
16     <mainFile>HelloWorklight.html</mainFile>
17     <thumbnailImage>common/images/thumbnail.png</thumbnailImage>
18
```

HelloWorklight - Application Descriptor

- An **XML** file to hold all the application's meta data
- Based on the W3C Widget Packaging and Configuration

```
18
19      <usage requireAuthentication="never" />
20      <!--
21          Uncomment one of the following options to require authentication upon using the application
22          <usage requireAuthentication="onStartup"><realm name="SampleAppRealm"/></usage>
23          <usage requireAuthentication="onDemand" ><realm name="SampleAppRealm"/></usage>
24      -->
25
26      <worklightRootURL>http://${local.IPAddress}:8080</worklightRootURL>
27      <!--
28          Uncomment one of the following options to use a different value for the Worklight root URL
29          <worklightRootURL>http://${local.hostname}:8080</worklightRootURL>
30          <worklightRootURL>http://${local.fullhostname}:8080</worklightRootURL>
31          <worklightRootURL>http://${local.IPAddress}:8080</worklightRootURL>
32       -->
33
34      <!--
35      <android version="1.0">
36          Uncomment and update push sender ID in orde
37          <pushSender id="my.company@gmail.com" passw
38      </android>
39      -->
40  </application>
```

Mobile worklightRootURL
The URL to be used as a root URL in generated mobile applications – iOS, Android etc.

HelloWorklight.html

During the runtime of an application, the main HTML document cannot be replaced by another HTML document.

Default application HTML template complies with HTML5 standard markup, but any other DOCTYPE can be specified.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-sca
        <title>HelloWorklight</title>
        <link rel="shortcut icon" href="images/favicon.png" />
        <link rel="apple-touch-icon" href="images/apple-touch-icon.png" />

        <link rel="stylesheet" href="css/reset.css" />
        <link rel="stylesheet" href="css/HelloWorklight.css" />
    </head>
    <body onload="WL.Client.init({})" id="content" style='display: none'>

        <script src="js/HelloWorklight.js"></script>
        <script src="js/messages.js"></script>
        <script src="js/auth.js"></script>
    </body>
</html>
```

HelloWorklight.html

During the runtime of an application, the main HTML document cannot be replaced by another HTML document.

Worklight Client framework initialization bound to body onload event. For possible init options see Developers Guide

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-sca
        <title>HelloWorklight</title>
        <link rel="shortcut icon" href="images/favicon.png" />
        <link rel="apple-touch-icon" href="images/apple-touch-icon.png" />

        <link rel="stylesheet" href="css/reset.css" />
        <link rel="stylesheet" href="css/HelloWorklight.css" />
    </head>
    <body onload="WL.Client.init({})" id="content" style='display: none'>

        <script src="js/HelloWorklight.js"></script>
        <script src="js/messages.js"></script>
        <script src="js/auth.js"></script>
    </body>
</html>
```

HelloWorklight.html

During the runtime of an application, the main HTML document cannot be replaced by another HTML document.

This is the right place to insert your html code

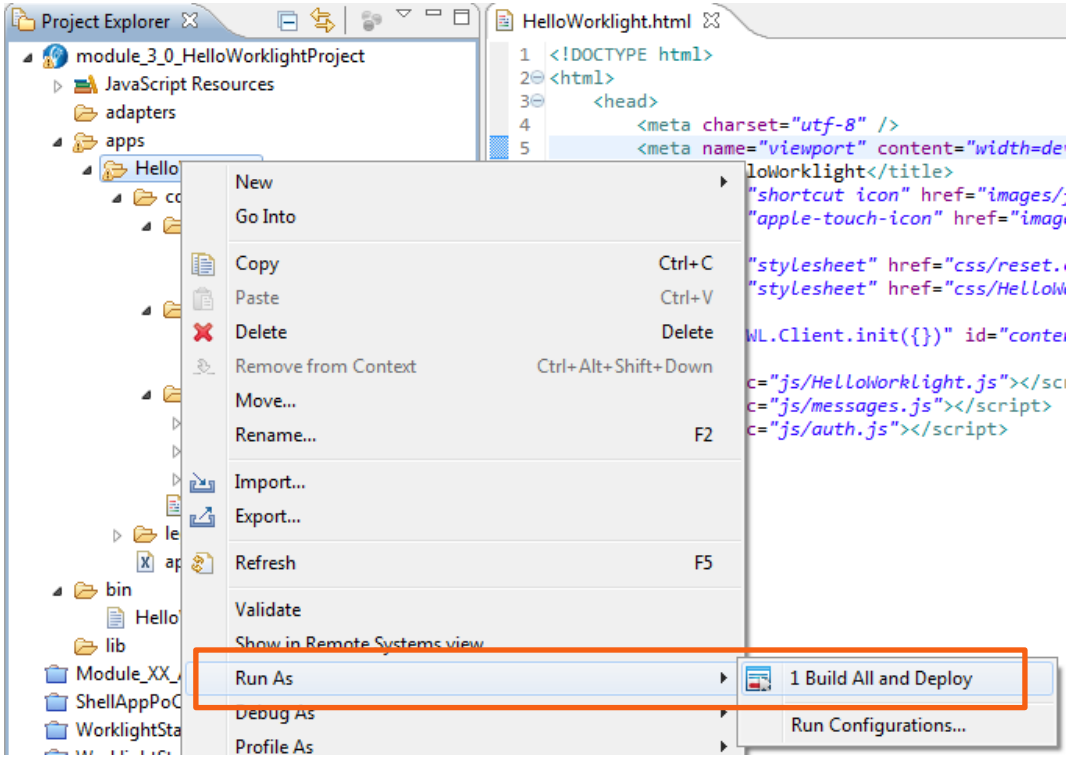
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-sca
        <title>HelloWorklight</title>
        <link rel="shortcut icon" href="images/favicon.png" />
        <link rel="apple-touch-icon" href="images/apple-touch-icon.png" />

        <link rel="stylesheet" href="css/reset.css" />
        <link rel="stylesheet" href="css/HelloWorklight.css" />
    </head>
    <body onload="WL.Client.init({})" id="content" style='display: none'>

        <scri                    loWorklight.js"></script>
        <script src="js/messages.js"></script>
        <script src="js/auth.js"></script>
    </body>
</html>
```

HelloWorklight.js

- The app's main **.js** file contains its JavaScript portion
- It has a **wlCommonInit()** function that will be invoked automatically once Worklight framework initialization finishes
- You can add your application's initialization code here
- This function will be used in environment-specific JavaScript files to have a common initialization starting point

```
function wlCommonInit(){
    // Common initialization code goes here
}
```

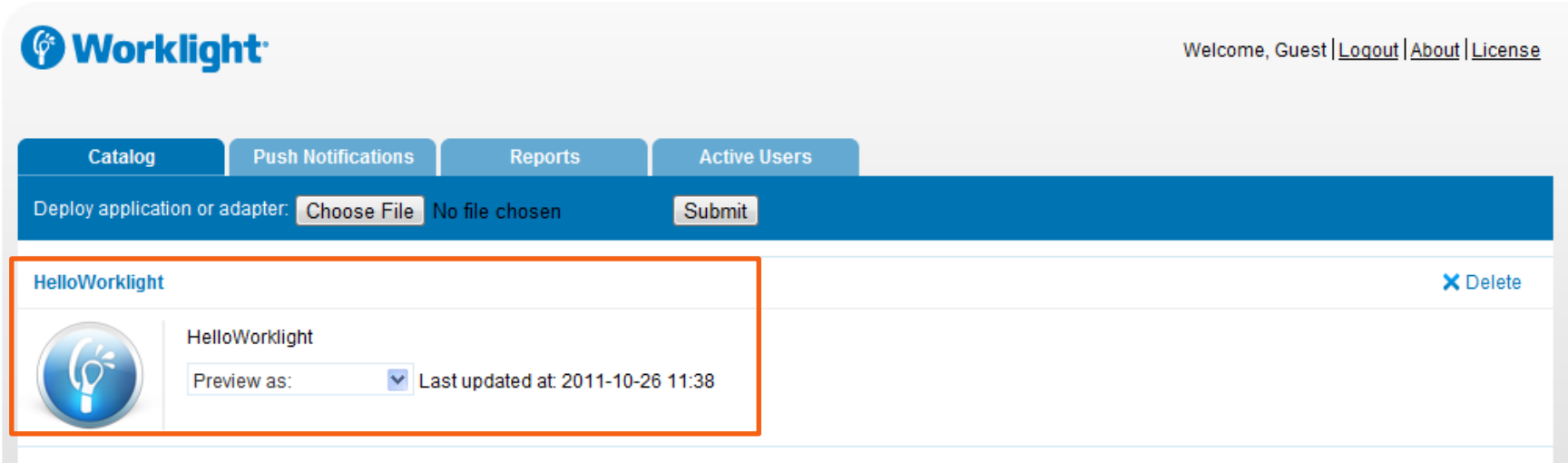
Building an Application



- Make sure your Worklight Server is up and running
- Select an app to build and right click on it
- Click **Run As**
- Select **Build All and Deploy**
- After build completes, the application will be available for preview in the catalog tab of the Worklight Console

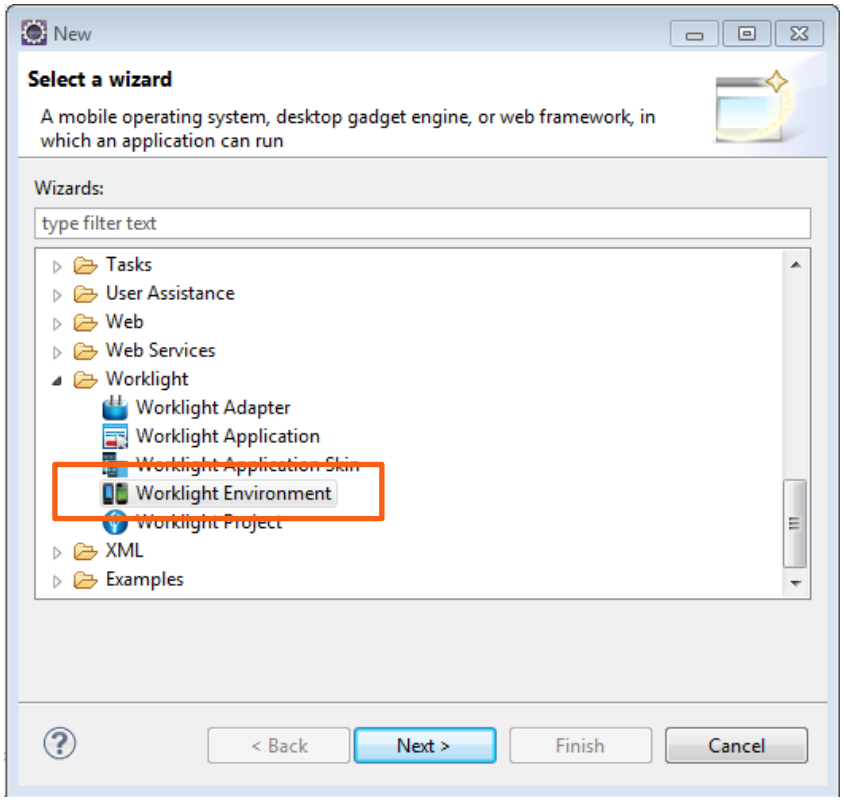
Deploying Apps Using the Worklight Console

`http://{Worklight Server}/console`



Adding a New Environment

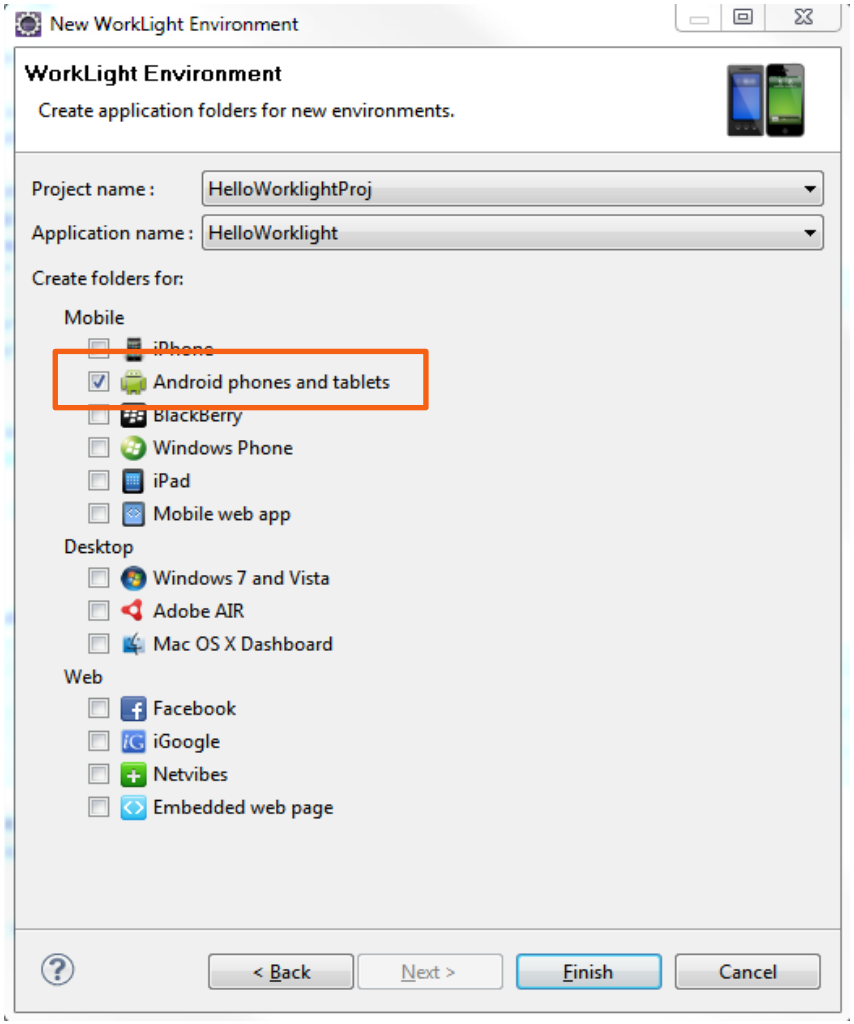
- To add a new environment right-click on your application folder and select New → Worklight Environment



Adding a New Environment

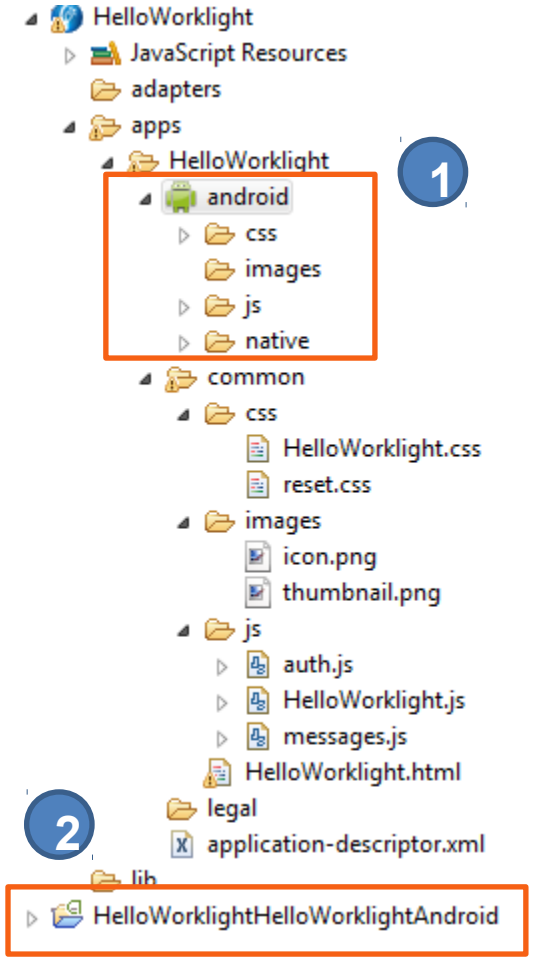
Select the ***Android phones and tablets***

- checkbox and click Finish



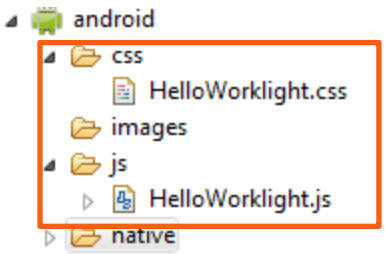
Adding a New Environment

- Two folders will be automatically added:
  1. ***android*** folder inside the application folder
  2. Android Project folder in your workspace.
- **IMPORTANT**
  - The auto-generated Android Project folder does not contain a copy of the code, but it is mapped to a ***native*** folder within the ***android*** folder of the application.



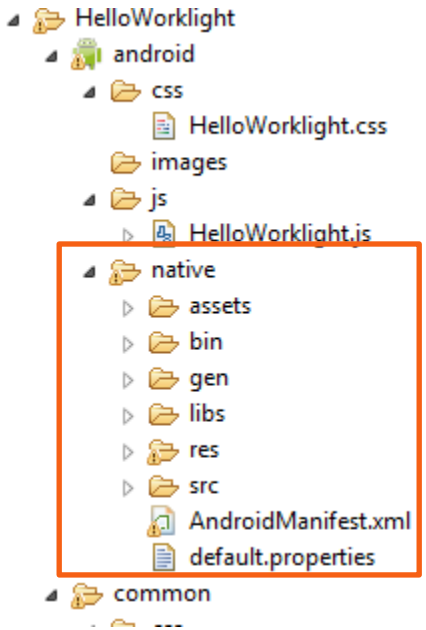
Review of the Android Folder Structure

- The Android environment consists of the following folders:
  - ***css*** – properties specified in here will override CSS files from the ***common*** folder.
  - ***images*** – Android specific images can be added here. If an image with same filename exist in the ***common*** folder it will be overwritten in the Android application.
  - ***js*** – JavaScript that can extend (and override if required) JavaScript code from the ***common*** folder.



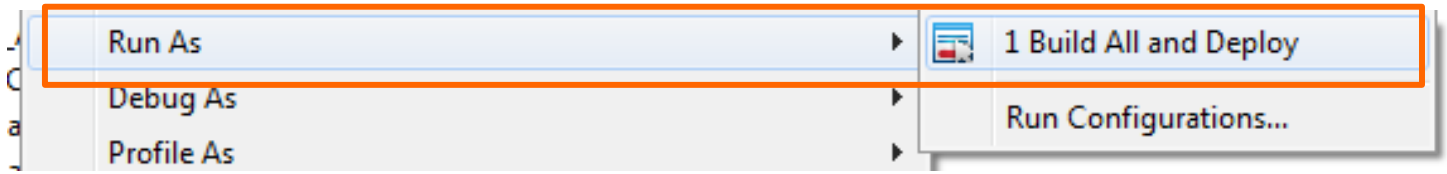
Review of the Android Folder Structure Cont.

- The ***native*** folder under ***android*** contains automatically generated android application code that is automatically imported into the eclipse workspace as an Android Project.
- It is not recommended to edit files under the ***assets*** folder, as each time the application is built they are regenerated.

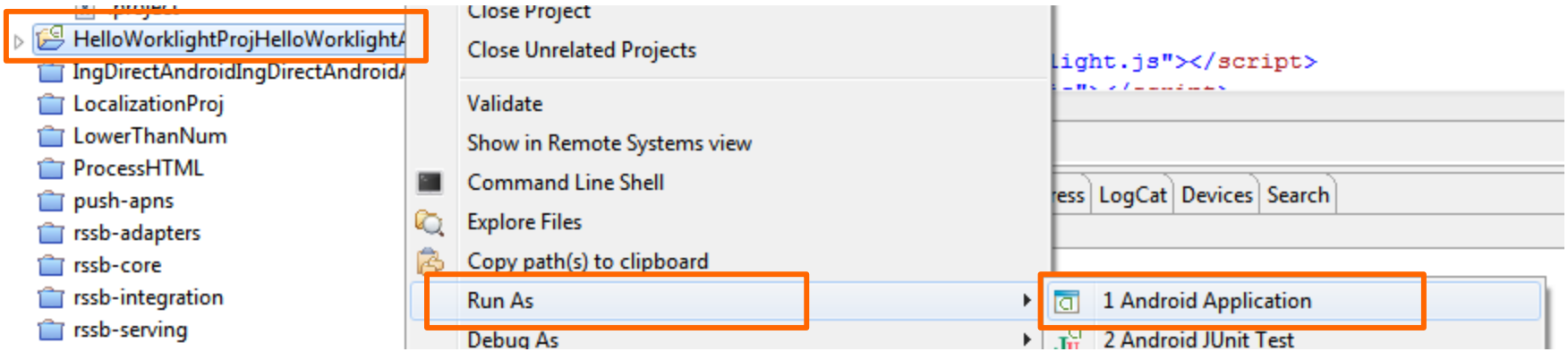


Running Your App on the Android Emulator

- Build and deploy your application on the Worklight Server



- Right-click the automatically generated Android Project and select Run As > Android Application



Running Your App on the Android Emulator

Congratulations, you've just created your first Android application

Running Your Application On a Real Device

- When an Android device is connected to the computer via USB cable, the Eclipse ADT plug-in will automatically recognize it and attempt to deploy applications onto it.

- More info and device drivers can be found at: http://developer.android.com/sdk/win-usb.html

Development

- HTML5 / CSS3
- JavaScript, both on Client and Server
  - For Client: optionally, a JavaScript Framework
    - Dojo Mobile, JQuery Mobile, Sencha Touch
- Optionally on Client
  - Native code, as a PhoneGap plugin
- Optionally on Server
  - Java code

The WL Namespace – client APIs

- To use Worklight API, a ***WL*** namespace is used
  - **WL**.Client, **WL**.Utils, …
- Exposes the API objects, methods and constants (usually enums)
- Automatically added to the app's main HTML file
  - wlcommon.js
  - wlclient.js
  - worklight.js
  - wlfragments.js
- WL Namespace is automatically available on application initialization

WL.Client

- WL.Client lets you perform the following types of functions:

WL.Client

- WL.Client lets you perform the following types of functions:
  - Initialize and reload the application.

WL.Client.init (options)
- onSuccess
- onFailure
- showLogger
- minAppWidth
- busyOptions

WL.Client.reloadApp()

WL.Client

- WL.Client lets you perform the following types of functions:
  - Initialize and reload the application.
  - Manage authenticated sessions.

WL.Client.getUserName (realm)
WL.Client.getLoginName (realm)
WL.Client.login (realm, options)
WL.Client.logout (realm, options)
WL.Client.isUserAuthenticated (realm)
WL.Client.getUserInfo (realm, key)
WL.Client.updateUserInfo (options)

WL.Client

- WL.Client lets you perform the following types of functions:
  - Initialize and reload the application.
  - Manage authenticated sessions.
  - Obtain general app information.

```
WL.Client.getEnvironment ()
 WL.Environment.ADOBE_AIR
 WL.Environment.FACEBOOK
…
```

WL.Client

- WL.Client lets you perform the following types of functions:
  - Initialize and reload the app.
  - Manage authenticated sessions.
  - Obtain general app information.
  - Retrieve and update data from corporate information systems.

WL.Client.invokeProcedure (invocationData, options)

WL.Client.makeRequest (url, options)

WL.Client

- WL.Client lets you perform the following types of functions:
  - Initialize and reload the app.
  - Manage authenticated sessions.
  - Obtain general app information.
  - Retrieve and update data from corporate information systems.
  - Store and retrieve user preferences across sessions.

```
WL.Client.setUserPref (key, value, options)
WL.Client.setUserPrefs ({key1:value1,  …}, options)
WL.Client.getUserPref (key)
WL.Client.removeUserPref (key, options)
WL.Client.hasUserPref (key)
```

WL.Client

- WL.Client lets you perform the follow functions:
  - Initialize and reload the app.
  - Manage authenticated sessions.
  - Obtain general app information.
  - Retrieve and update data from corporate i
  - Store and retrieve user preferences acros
  - Internationalize app texts.
  - Specify environment-specific user interface behavior.

WL.App.openURL
WL.App.getDeviceLanguage
WL.App.getDeviceLocale
WL.Client.onDock
WL.Client.onShow
WL.Client.onHide
WL.BusyIndicator
WL.TabBar
WL.SimpleDialog
WL.OptionsMenu

WL.Client

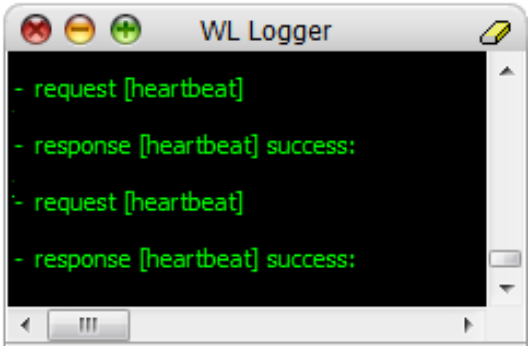
- WL.Client lets you perform
  functions:
  - Initialize and reload the app.
  - Manage authenticated session
  - Obtain general app information
  - Retrieve and update data from
  - Store and retrieve user preferences across sessions.
  - Internationalize app texts.
  - Specify environment-specific user interface behavior.
  - Store custom log lines for auditing and reporting purposes in special database tables.

WL.Client.logActivity (activityType)

WL.Client

- WL.Client lets you perform functions:
  - Initialize and reload the app.
  - Manage authenticated session
  - Obtain general app information
  - Retrieve and update data from
  - Store and retrieve user preferences across sessions.
  - Internationalize app texts.
  - Specify environment-specific user interface behavior.
  - Store custom log lines for auditing and reporting purposes in special database tables.
  - Write debug lines to a logger window.

WL.Logger.debug (msg)
WL.Logger.error (msg)



WL.Client

- WL.Client lets you perform the follo functions:
  - Initialize and reload the app.
  - Manage authenticated sessions.
  - Obtain general app information.
  - Retrieve and update data from corporate information systems.
  - Store and retrieve user preferences across sessions.
  - Internationalize app texts.
  - Specify environment-specific user interface behavior.
  - Store custom log lines for auditing and reporting purposes in special database tables.
  - Write debug lines to a logger window.
  - Dynamic page and fragments loading.

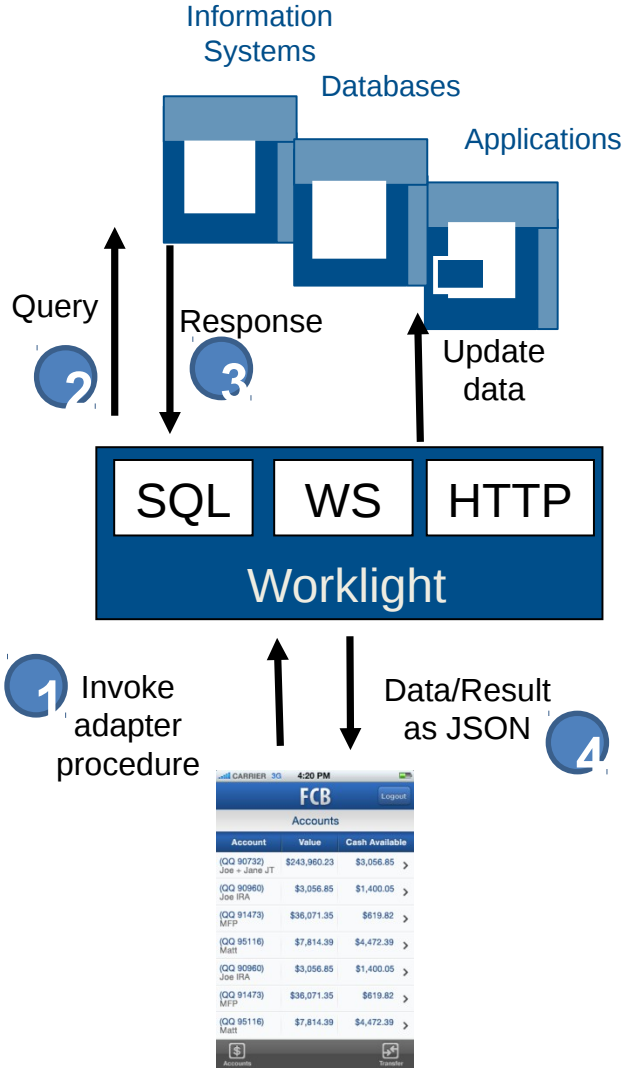
WL.Fragment.load
WL.Page.load
WL.Page.back
WL.Page.isLoaded

WL.Client

- WL.Client lets you perform the following types of functions:
  - Initialize and reload the app.
  - Manage authenticated sessions.
  - Obtain general app information.
  - Retrieve and update data from corporate information systems.
  - Store and retrieve user preferences across sessions.
  - Internationalize app texts.
  - Specify environment-specific user interface behavior.
  - Store custom log lines for auditing and reporting purposes in special database tables.
  - Write debug lines to a logger window.
  - Dynamic page and fragments loading.
- For full documentation please refer to the ***Developer Reference Guide***.

Server-Side Development

- An Adapter is a transport layer used by the Worklight Platform to connect to various back-end systems.
- Adapters are used for:
  - Retrieving information
  - Performing actions
- Out of the box:
  - SQL Adapter
  - HTTP Adapter (supports both
  - REST and SOAP)



Example: Build a SQL Adapter

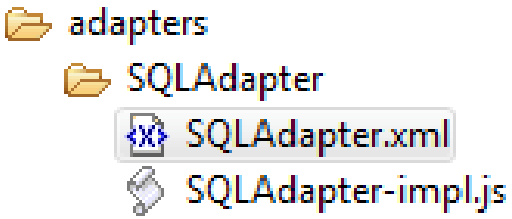
- A Worklight SQL Adapter is designed to communicate with any SQL data source
- Both plain SQL queries or stored procedures can be used

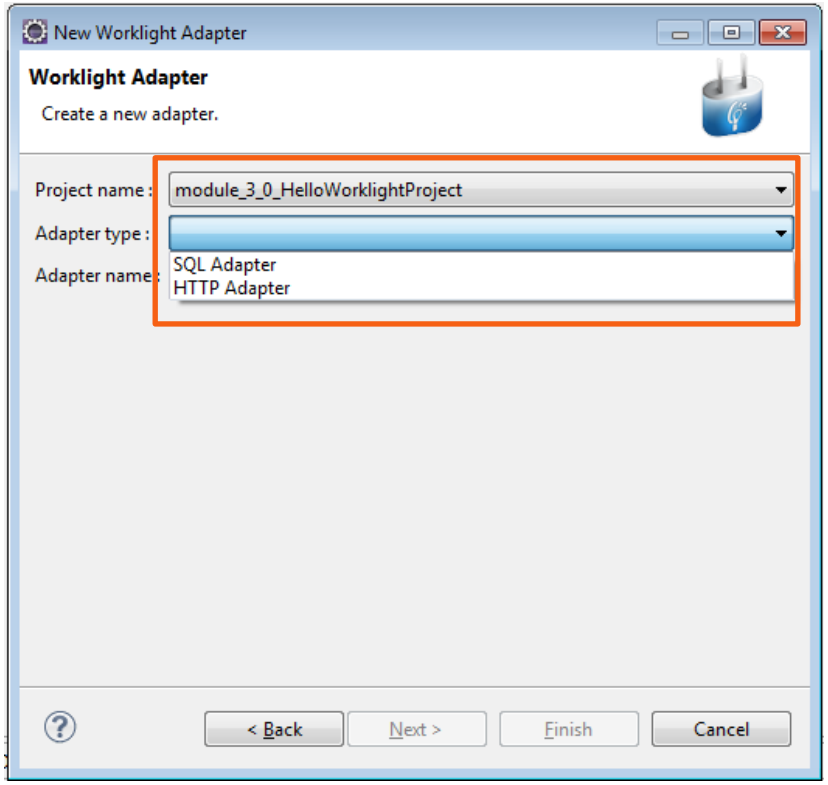
Creating a Worklight SQL Adapter

- Within the Worklight Studio, right click on the Worklight Project and create a new Worklight SQL Adapter

A standard SQL Adapter

- structure will be created





XML File

Connectivity Settings

- Connection can be either JNDI or data source definition
  - JNDI - Java Naming and Directory Interface - providing Java applications with a unified interface for multiple naming and directory services.
  - A JNDI name can be declared as a parameter that is defined in a **worklight.properties** file. This is an adapter's XML file:

```
<connectivity>
    <connectionPolicy xsi:type="sql:SQLConnectionPolicy">
        <dataSourceJNDIName>${training-jndi-name}</dataSourceJNDIName>
    </connectionPolicy>
    <loadConstraints maxConcurrentConnectionsPerNode="5" />
</connectivity>
```

- And these are corresponding entries in the **worklight.properties** file

```
training-jndi-name=${custom-db.1.jndi-name}
custom-db.1.relative-jndi-name=jdbc/worklight_training
custom-db.1.driver=com.mysql.jdbc.Driver
custom-db.1.url=jdbc:mysql://localhost:3306/worklight_training
custom-db.1.username=Worklight
custom-db.1.password=Worklight
```

JS File
Procedure

- A procedure must be declared in the Adapter's XML file

```
<procedure name="GetAccounts"/>
```

- The adapter's JavaScript file is used to implement the procedure logic

- IMPORTANT – The same name declared in the XML file should be used for the procedure's JavaScript function

- There are two ways of invoking SQL statements:
  - Using SQL statement query
  - Using SQL Stored Procedure

JS File
SQL Query

- To execute an SQL query
  1. Prepare an SQL query using the ***WL.Server.createSQLStatement*** API
  2. ***WL.Server.createSQLStatement*** should ALWAYS be called outside of the function
  3. Add additional parameters if required
  4. Use the ***WL.Server.invokeSQLStatement*** API to invoke prepared queries
  5. Return invocation result to procedure invocator (application or another procedure)

```
var procedure1Statement =
    WL.Server.createSQLStatement("select FirstName, LastName from Customers where CustomerID = ?");
function GetAccounts(param) {
    return WL.Server.invokeSQLStatement({
        preparedStatement : procedure1Statement,
        parameters : [param]
    });
}
```

1

4

3

JS File

Stored SQL Procedure

- To execute a stored SQL procedure
  1. Use the ***WL.Server.invokeSQLStoredProcedure*** API to execute a stored procedure
  2. Specify an SQL stored procedure name as an invocation parameter
  3. Add additional parameters if required
  4. Return invocation result to procedure invocator (application or another procedure)

```
function GetAccounts(param) {
    return WL.Server.invokeSQLStoredProcedure({
        procedure : "GetAccountsByUsedID",
        parameters : [param]
    });
}
```

1

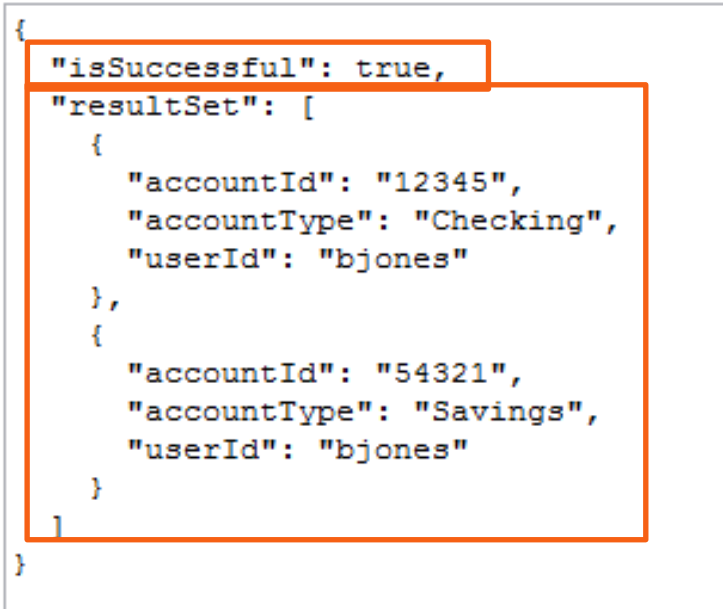
2

3

WL Adapters
Invocation Result

- Result retrieved as a JSON object
- ***isSuccessful*** property defines whether invocation was successful
- ***resultSet*** is an array of returned records

Invocation Result from the WorkLight Server:



```
{
  "isSuccessful": true,
  "resultSet": [
    {
      "accountId": "12345",
      "accountType": "Checking",
      "userId": "bjones"
    },
    {
      "accountId": "54321",
      "accountType": "Savings",
      "userId": "bjones"
    }
  ]
}
```

Summary

- Worklight extends IBM's existing mobile strategy and offerings
- Fully integrated platform for mixed web / hybrid / native development
- More information:
- http://www.youtube.com/user/WorklightInc?feature=watch
- http://worklight.com/