This tutorial shows how to build a Node.js app that uses cloud services on Bluemix and integrates with enterprise data and processes. Our app is a supply chain application that uses Business Rules, dashDB, and custom cloud services on Bluemix, integrated with IBM Worklight, Business Process Management (BPM), Content Management, and API Management for enterprise connectivity to corporate back-end systems.

The IBM Bluemix™ cloud development platform provides the infrastructure, runtime environments, reusable services, rich ecosystem and DevOps tooling to help you quickly build and deploy enterprise apps. For example, we built an enterprise supply chain app on Bluemix as a system of engagement integrated with various technologies.

We built the app with the Node.js runtime, custom cloud services on Bluemix, BLU Acceleration, and Business Rules on Bluemix integrated it with IBM Worklight Server, Business Process Management (BPM), Content Management, and an API platform for enterprise connectivity to corporate back-end systems.

The steps and code samples in this tutorial will help you create and deploy a similar enterprise app.
“In building our Node.js app, we integrated various technologies within Bluemix and outside of Bluemix, but you can customize and use the same integration architecture for any enterprise scenario.”

About our supply chain app

The robust, scalable supply chain app lets sellers of products and services focus on customers rather than on back-office operations to support sales transactions. In the sales process, sellers use a mobile app or web app to:

- Request back-office support to validate customer information
- Create price quotes
- Create a bid to engage with the customer to win the sale and create a contract

The signed contract is then registered by the back-office support specialist in a System of Record (SoR) so that the business transaction can be recorded and processed further for cash collection.

What you'll need to build a similar application

Only the first four items below are required, but in a typical enterprise app scenario, it is common for core foundation services or the orchestration layer (for example, a Node.js app) to integrate with components for business process management, content management, mobile-optimized middleware, APIs, and so on.

- A Bluemix account, so you can use the Node.js runtime and the Business Rules service.
- A DevOps Services account to get and fork the sample code below.
- BLU Acceleration for fast and robust in-database analytics.
- JQuery or any other user interface (UI) library to build the UI and any LDAP directory for authentication.
- (Optional) IBM Business Process Manager (BPM), which is a full-featured, consumable business process management platform designed for faster time to deployment. It includes tooling and a runtime for process design, execution, monitoring, and optimization of business processes.
- (Optional) IBM FileNet Content Manager, which helps you manage enterprise content with greater speed, efficiency, and accuracy. It is a document management engine that combines enterprise content, security, and storage features.
- (Optional) IBM Worklight Server, which is mobile-optimized middleware that serves as a gateway between applications, back-end systems, and cloud-based services.
• (Optional) **IBM API Management**, which provides tools for creating, proxying, assembling, securing, scaling, and socializing web APIs.

**Get the code**

**About the code**

• The Bluemix Integration code is in the BluemixIntegration directory.
• The mobile app code is under the SRMobileUI directory.

**Integration scenario**

To support the sales process, the supply chain app needs to integrate various components on Bluemix and outside of Bluemix. For example, when a sales support request is initiated by the seller to the back office, the request is processed by assigning a support specialist who will then create and store content, such as pricing documents or contracts, in a content management (CM) system. The request for support is handled as workflow due to multiple hand-offs between seller, support specialists, approvers, and automated steps when applicable. (You might have scenarios where an approval hand-off must occur for special pricing. For brevity, we did not include this scenario in our implementation.)

Essentially:

• The supply chain app uses **Node.js on Bluemix** as the orchestration layer for both mobile and web workloads.
• **IBM Worklight Server** handles mobile requests. IBM Worklight Server calls the Node application to orchestrate the request.
• The Node application is integrated with the **Business Rules service on Bluemix** to determine the type of customer and validate customer information with customer master data.
• The Node application integrates with **IBM Business Process Manager (BPM)** to handle workflow.
• The Node application integrates with **IBM FileNet Content Manager** for content management via FileNet APIs.
• The Node application integrates with the **IBM API Management** platform to invoke APIs that connect to corporate back-end systems.
• Custom Bluemix services are used to derive prices for products and service offerings.
• **BLU Acceleration** provides data warehousing and real-time analytics.

In this implementation, we are using an on-premise Bluemix platform within an IBM firewall so we don’t need the Cloud Integration service on Bluemix. However, you might want to use the **Cloud Integration service** on Bluemix if you need to access corporate back-end systems behind firewalls.
Architecture

The mobile app and web app integration with Bluemix and enterprise back-end services involves a request-response pattern and various components and services:
1. The mobile integration adapter (Worklight Server) sends the mobile request directly to the Node app on Bluemix for orchestration. Web requests are sent directly to the Node app for orchestration.

2. The Node app calls the BPM API to determine where the request is in the workflow and invokes the next step in the flow.

3. The Node app calls the FileNet Content Management API to upload content to the FileNet repository.

4. The Node app calls Business Rules on Bluemix to run business rules and uses the output to process the next steps.

5. The Node app calls custom Bluemix services such as Pricing to calculate a price for the product or offering.

6. The Node app calls the BLU Acceleration to process the request type and customer information.

**Step 1. Build the application**

First, get a Bluemix Node.js project template, by creating a Node.js application on Bluemix, downloading the code from Bluemix to your local file system, and importing it to your IDE. A new Node.js application on Bluemix looks like this:
A Bluemix Node.js template will automatically create the following project structure:

- **app.js**: This file contains the server-side JavaScript code for your application written using the Node.js API.
- **views/**: This directory contains the views of the application. It is required by the Express framework and Jade template engine in this sample application.
- **public/**: This directory contains public resources of the application. It is required by the Express framework in this sample application.
- **package.json**: This file is required by the Node.js environment. It specifies this Node.js project name, dependencies, and other configurations of your Node.js application.

We added more source code to implement our integration project. For the UI, we used HTML, JQuery, and Express. The mobile UI runs on IBM Worklight Server. The UI is served by the Node.js app running on Bluemix.

Our steps show Bluemix integration with multiple technology components such as BPM, Content Management, API Platform, and Business Rules. Based on our application design, we chose to integrate using RESTful services with these technology components. However, you can use any integration method that is suitable for your project.

A key design decision was to use an app written in Node.js as an orchestration layer for our application due to its non-blocking and asynchronous architecture. Any activities that take a long time to finish, such as file upload or network communication, are requested and put aside until the results are ready and returned via a callback function.

**Project file structure**

The file structure for the project is:
To get all the project files, go to our BlueMixEnterpriseApp DevOps Services project.

- **app.js**: Declares multiple Node.js modules, such as Express, ejs, opp_rest, etc., that are used in our project
- **config.js**: All application configuration parameters were externalized into this file so all users can change them if needed.
- **db_helper.js**: Shows how to connect to the database, run queries, and get the results back. It uses some common .js files and the ibm_db module.
- **Helper.js**: Has common functions, such as get content by http/https request.
- **opp_dao.js**: For integration with external systems to retrieve the Sales Opportunity list.
- **Service.js**: Handles all the http/https calls to call external APIs, BPM, and Bluemix services.

**Step 2. Add external system calls**

The following sections have code snippets for external systems calls from Bluemix. (For all the code, go to our BlueMixEnterpriseApp DevOps Services project.) If you want your own copy of the code to edit, you can fork the project in DevOps Services.

**READ:** Developing an IBM Bluemix application in Node.js with the Web IDE (for details on how to fork and deploy a forked project)

**Authentication using LDAP**

The user will log in with their credentials and Node.js will validate them against the LDAP system using the ldapjs module. Here is the authentication method that was used:

```javascript
var authenticate = function (username, password, callback) {
    if (!user_config.isCheckUser) {
        getSuccessResult(username, password, null, callback);
    } else {
        var opts = {
            filter: '(mail=' + username + '),' + 'scope: ' + 'sub'
        };
        var client = ldap.createClient({
            url: ldap_config.scheme + '://' + ldap_config.ldapHost + ':' + ldap_config.ldapPort
        });
        var isFoundEntry = false;
        client.search(ldap_config.base, opts, function(err, res) {
            res.on('searchEntry', function(entry) {
                isFoundEntry = true;
                // found the user entry
            })
            if (isFoundEntry) {
                authenticatePassword(username, password, callback);
            } else {
                callback('missing login information', null);
            }
        });
    }
}
```
if(user_config.isCheckPassword){
  client.bind(entry.dn, password, function(err) {//bind successfully
    if (err)
      callback(helper.getMessageObject(false,"Your password is not correct.");
    else{
      getSuccessResult(username,password,entry,callback);
    }
  });
} else{
  getSuccessResult(username,password,entry,callback);
}

//do action if err happened when search run
res.on('error', function(err) {
  callback(helper.getErrorMessage(err.toString()));
});
//do action if end such as release client
res.on('end', function(result) {
  client.unbind(function(err) {
    if (err!==null)
      callback(helper.getErrorMessage(err.toString()));
  });
  if (!isFoundEntry)
    callback(helper.getMessageObject(false,"Your intranet id is not correct.");
});

REST call to BPM
In opt_resst.js:

exports.startProcess=function(header,callback){
  helper.httpsRequest(config.host,config.port,'/rest/bpm/wle/v1/process?action=start&bpdId=' + config.bpdId + '
&processAppId='+ config.processAppId,
  null,true,callback,'POST',header,false);
};

We used RESTful calls to integrate with BPM activities. In the snippet above, the Node.js app on Bluemix triggers the BPM process to start a new service request using a REST call. Because the httpsRequest method is a common method, it is in the helper.js.

REST call to API to get customer information
The code on Bluemix calls an external API to retrieve customer data from Systems of Record. You could use this code to call any external APIs.

exports.retrieveCustName=function(country,custno,header,callback){
  var pathStr =
  config.valid_service.path.replace("[countryNo]",encodeURIComponent(country)).replace("[custNo]",encodeURIComponent(custno));
  helper.httpsRequest(config.valid_service.host,config.valid_service.port,pathStr,null,true,callback,'GET',header,false);
};

Call Bluemix rules
Using the Business Rules service on Bluemix, we externalized the business rules for a simple scenario to illustrate the integration with Business Rules on Bluemix.

Build an enterprise supply chain app on Bluemix
The snippet below was used to call the Business Rules service on Bluemix. If a business analyst decides to change the rules, it can be done without having to change the Node.js app. This approach provides business agility.

```javascript
exports.compareWithODM = function(requestorCust, retrieveCust, callback) {
  var header = {"Content-Type": "application/json"};
  var data = {"Customer": {
    "firstName": requestorCust!=null?requestorCust.trim():requestorCust,
    "secondName": retrieveCust!=null?retrieveCust.trim():retrieveCust
  }};
  helper.httpsRequest(config.ilogUrl, config.ilogPort, config.ilogPath, data, false, callback, "POST", header, false);
};
```

### Call Content Management (CM) system

It is critical to store content for an enterprise application in a reliable and scalable CM system — especially for audit purposes. We decided to store all documents in a CM system that let us access the documents using APIs or directly from Content Navigator, which is the UI for CM systems that support Content Management Interoperability Standards (CMIS).

Using the CM API, the Node.js app will upload the contract document to the CM system and get the document ID, which BPM will use to update the records:

```javascript
var filePath = path.dirname(req.files.fileupload.path) + " / " + req.files.fileupload.originalFilename;
fs.rename(req.files.fileupload.path, filePath, function() {
  form.append("fileupload", fs.createReadStream(filePath));
  var beginTime = new Date();
  var endTime = null;
  postRequest(form, function(err, response, body) {
    endTime = new Date();
    fs.unlink(filePath);
  });
});
```

### Step 3. Create Bluemix custom services (user-provided services)

A user-provided service on Bluemix calculates the prices for a sales contract. We decided to host the pricing service on Bluemix to make the service available to other Bluemix applications.

To create a service instance in interactive mode, use the `-p` option with a comma-separated list of parameter names. `cf` then prompts you for each parameter in turn, as shown below.

You will need to put in your own parameters based on your host, port, path, and so on.

Bluemix will create a new user-provided service.

Next, you will need to bind your service to your application. There are two ways to bind user-provided service instances to applications:

- With a manifest, when you push the application
• With the `cf` bind-service command, after you push the application

We will use the `cf` command to bind the service, as follows:

```bash
$\text{cf bind-service myAppName myServiceName}
```

For the binding to take effect, you must push your application a second time.

```bash
cf push your AppName
```

**READ: Creating user-provided service instances**

### Step 4. Connect to customer information

We also stored some customer information using Blu Acceleration. Here's a code sample to connect to Blu Acceleration from Node.js:

```javascript
exports.getCustInfo=function(custNo,callback){
    var sql="select * from CUSTOMER where CUST_NO='"+custNo+"';
    db_helper.sqlQuery(sql,true,function(data){
        if(data.isSuccess===false){
            callback(data);
        }else{
            if(data.list.length>0){
                callback({'CustNo':custNo,'CustRegion':data.list[0].CUST_REGION,'AccountSize':data.list[0].ACCOUNT_SIZE});
            }else{
                callback({'CustNo':custNo,'CustRegion':'Not Avaliable','AccountSize':'Not Avaliable'});
            }
        }
    });
    exports.sqlQuery=function (sql,isList,callback){
        this.runQuery(sql,isList,db2_bm,callback,true);
    };
};
```

### Step 5. Create the mobile app

We created a mobile application and hosted it on IBM Worklight Server. Node.js on Bluemix is the middle layer for mobile requests and the web browser, which allowed us to use the same methods for both UIs.

![SMBuild](https://developer.ibm.comulatory/examples/smbuild/)

The mobile application has two adaptors.
• AuthAdapter authenticates the user against an LDAP system:

```javascript
function submitAuthentication(username, password) {
    var input = {
        method: 'post',
        returnedContentType: 'json',
        path: 'service/sign',
        parameters: {
            j_username: username,
            j_password: password,
            is_mobile: 'true'
        }
    };
    var result = WL.Server.invokeHttp(input);
    if (result && result.isSuccess) {
        var userIdentity = {
            userId: username,
            displayName: username,
            loginName: username,
            attributes: {token: result.token, role: result.role}
        };
        WL.Server.setActiveUser(REALM_NAME, userIdentity);
        return { type: 'auth', authRequired: false };
    } else {
        return onAuthRequired(null, 'User name or password is wrong.);
    }
}
```

• RestServiceAdapter handles all REST calls to Bluemix to retrieve a Sales Opportunities list and Service Request (SR) list, create a new SR, and validate an SR. RestServiceAdapter.xml shows the methods that the adaptor is calling.

```xml
<procedure name="getSRList" securityTest="AuthenticationAdapter-securityTest"/>
<procedure name="getSRDetails" securityTest="AuthenticationAdapter-securityTest"/>
<procedure name="createSR" securityTest="AuthenticationAdapter-securityTest"/>
<procedure name="validateSR" securityTest="AuthenticationAdapter-securityTest"/>
<procedure name="getOppList" securityTest="AuthenticationAdapter-securityTest"/>
```

**Conclusion**

You've now seen the general steps and source code for building an enterprise application on Bluemix and integrating with corporate back-end systems. You can use the code provided to create and customize enterprise apps for your own scenarios.

**Acknowledgments**

The authors wish to acknowledge:

- Milton Bonilla from the CIO Lab for his contribution to make Bluemix available in the CIO Lab.
- Zhi Yong Mao, Jian H Zhang, and Yi WH Zhou from the IBM China development team for their contributions to this Bluemix project.
- Marco Marquez and Robert Kirby of IBM Integrated Supply Chain for their contributions to the BPM component.
- Sam Thompson of IBM Software Group for feedback and suggestions to improve the article.

Node.js runtime [https://ace.ng.bluemix.net/#/store/cloudOEPaneId=store&appTemplateGuid=nodejsHelloWorld&fromCatalog=true] helps you develop, deploy, and scale server-side JavaScript apps with ease. Business Rules service [https://ace.ng.bluemix.net/#/store/cloudOEPaneId=store&fromCatalog=true&serviceOfferingGuid=b13b9cf8-fff8-4e3a-b794-acf3558b91f9] minimizes your code changes by keeping business logic separate from application...
logic.dashDB service is a data warehousing and analytics solution powered by IBM BLU Acceleration.

RELATED TOPICS: Node.js Business Process Management

© Copyright IBM Corporation 2014
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)