Create a natural language question answering system with IBM Watson and Bluemix services

Swami Chandrasekaran (swamchan@us.ibm.com)
Executive Architect at IBM Watson
IBM

Carmine DiMascio
Senior Software Engineer
IBM

29 September 2014
(First published 30 May 2014)

IBM® Watson™ represents a first step into cognitive systems, a new era of computing. Watson inspires many developers to dream up new and inventive ideas for applications that use cognitive elements to deliver a better and more personalized experience to the user. In this article, we will use Watson Question and Answer (Q&A) technology and the Q&A API exposed by Watson to build a simple demo application, Watson Films.

This article was written using the Bluemix classic interface. Given the rapid evolution of technology, some steps and illustrations may have changed. The Bluemix service names have also changed.

Cognition is in virtually everything that humans do, such as language understanding, perception, judgment, motor skills, learning, spatial processing and social behavior. Increasingly, we expect the machines that we use to exhibit the same cognitive behavior. Watson represents a first step into cognitive systems, a new era of computing. In addition to using programmatic computing, Watson has three capabilities that make it truly unique:

- Natural language processing
- Hypothesis generation and evaluation
- Dynamic learning

> *“ Watson is a natural language question answering system that does not use prepared answers, but determines its answers and associated confidence scores that are based on knowledge it has acquired. ”*

Watson brings these powerful capabilities together in a way that's never been done before, resulting in a fundamental change in the way businesses look at quickly solving problems. Watson

is a natural language question answering system that does not use prepared answers, but determines its answers and associated confidence scores that are based on knowledge it has acquired.

Watson has inspired many developers to dream up new and inventive ideas for applications that use cognitive elements to deliver a better and more personalized user experience. Applications that are powered by Watson can move beyond simple processing of data to finding correlations, creating hypotheses, and learning from outcomes. In this article, we will use Watson Question and Answer technology and the Question and Answer API (QAAPI) exposed by Watson to build a simple demo application, Watson Films.

Note: *To run the "Watson Films" application in this article end-to-end, you must have access to a Watson instance. To get access to a Watson instance, refer to Watson Developer Cloud Enterprise (PDF file) and / or apply to the Watson Ecosystem program. For you to understand the Watson input and output, we provide Watson QAAPI request and response JSON samples in the DevOps Services project under the watson_films_dw/samples folder. Thanks for understanding!*

The application will be hosted and managed on IBM Bluemix™, a **Cloud Foundry**-based **PaaS** for building, managing and running apps of all types — web, mobile, big data, and smart devices. The code for the application will be managed through DevOps Services (JazzHub).

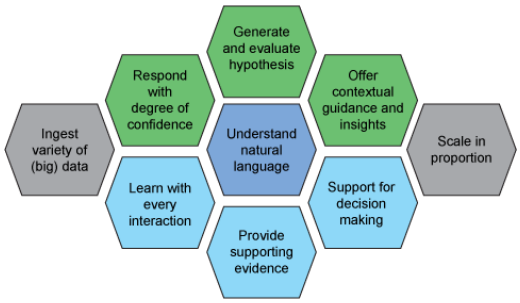
Characteristics of a cognitive system

Cognitive systems:

- Navigate the complexities of human language and understanding
- Ingest and process vast amounts of structured and unstructured (big) data
- Generate and evaluate countless possibilities
- Weigh and evaluate responses that are based only on relevant evidence
- Provide situation-specific advice, insights, and guidance
- Improve knowledge and learn with each iteration and interaction
- Enable decision making at the point of impact
- Scale in proportion to the task

These systems apply human-like characteristics to conveying and manipulating ideas. When combined with the inherent strengths of digital computing, they can solve problems with higher accuracy, more resilience, and on a more massive scale. Watson is an example of a Cognitive System. It can tease apart human language and identify inferences between text passages with human-like high accuracy at speeds far faster and scale far larger than any human. A rules-based approach would require a near-infinite number of rules to capture every case we might encounter in language.

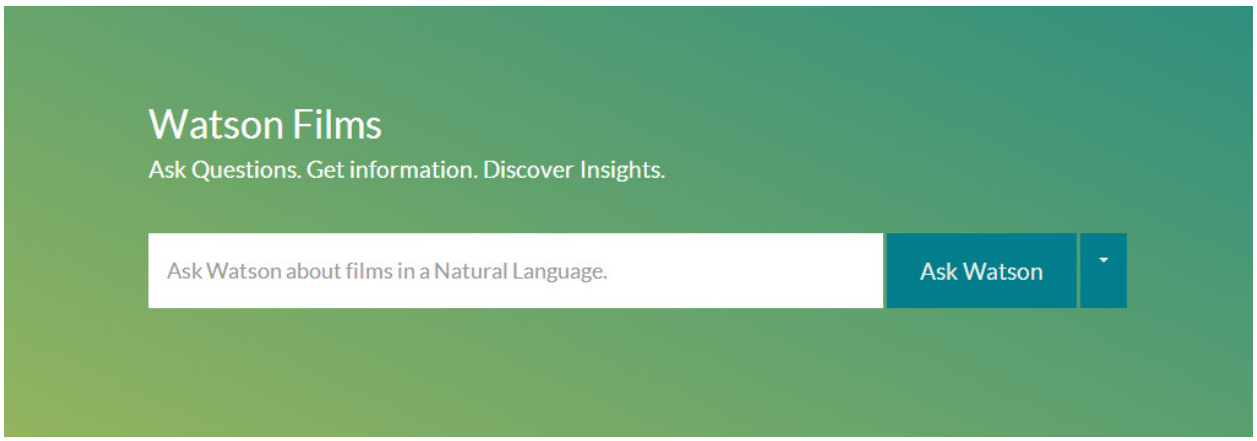
Figure 1. Key characteristics of a cognitive system



Preparing to build Watson Films

The Watson Films app is a simple demonstration of how to build an application that interacts with Watson by using the Watson QAAPI. This application allows users to ask questions about films, especially the American Film Institute top 100 American films, and about filmmaking in general.

Figure 2. Watson Films interface



Before Watson can answer user questions, it must have a knowledge base that contains information from which it can formulate correct answers. For this application, we use content from Wikipedia about the list of the top 100 greatest American films of all time according to the American Film Institute. This content from Wikipedia is in HTML format. We also use content from *The Essential Reference Guide for Filmmakers* from Kodak, which is in PDF format.

Get or view the code

WATCH: Webcast: Watson Can Power Your Application. What Will You Do With It? http://ibm.biz/WatsonBlmx-dw

WATCH: Watson Films (video demo) http://youtu.be/r-HNSGt53wg

READ: Watson Films transcript (PDF file)

WATCH: See IBM Bluemix in action http://www.ibm.com/developerworks/cloud/library/cl-bluemix-dbarnes-ny/index.html

Get the Watson Films code

- Try it first! Check out the Watson Films demo (read PDF transcript file).

- Get or view the Watson Films code by clicking the preceding **Get or view the code** button, or by using IBM DevOps Services with the Web IDE. Because this project contains JavaScript, HTML, and CSS code, the Web IDE is probably all you need to view or edit after you fork the project.
- To download and set up this project in your local Eclipse IDE (with the Nodeclipse plug-in installed), follow these instructions.
- You can also fork the "Watson Films" project and create your own DevOps Services project that is based on the contents of this project.

About the Watson Films code

The Watson Films demo application is built on Node.js with Express. We also use Bootstrap and Slick Carousel to construct and style the user interface.

The following is a brief summary of the technologies used:

- Watson Services for Bluemix – rapidly prototype and build powerful cognitive apps in the cloud
- Node.js - a lightweight platform that is built on Chrome's JavaScript run time for easily building fast, scalable, server-side network applications
- Express.js - a web application framework for Node
- Bootstrap.js – a simple and responsive front-end web development framework
- jQuery - a fast, small, feature-rich JavaScript library
- Ladda – a responsive button toolkit
- Slick Carousel – a JavaScript carousel. Yes, Bootstrap also has a carousel, but we wanted to try out Slick!

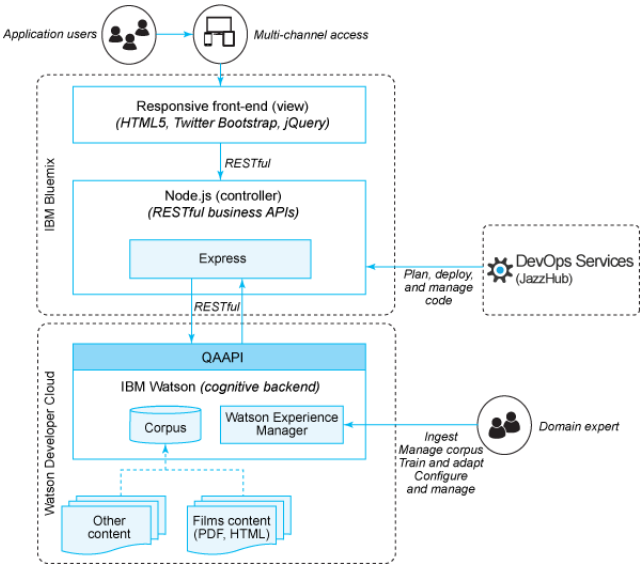
About the Watson Films architecture

The Watson Films architecture is made up of the following layers and associated components:

- **View**: A responsive front-end user interface primarily built by using HTML5, CSS3, Bootstrap, jQuery and other JavaScript client-side plug-ins. The *view* component in this layer allows the user with an entry point to ask questions of Watson.
- **Controller**: A Node.js and Express-based middleware and controller tier that exposes RESTful business APIs to the *view* components. This REST API interacts with Watson by invoking the Watson QAAPI and handling the responses. It can be extended to perform other business operations, including integrating with other cognitive services and third-party services.
- **Cognitive Backend**: Watson ingested with content about films and filmmaking. Watson provides a RESTful interface, the QAAPI. This API enables developers to ask Watson questions about films and filmmaking and receive answers. Answers include various information such as the answer text, Watson's confidence level, and evidence that supports the answer.
- **Platform and DevOps**: The Watson Films application will be deployed on Bluemix. It also uses DevOps Services for managing the code. DevOps Services is responsible for management of the Watson Films source code and its deployment to Bluemix. Bluemix hosts

both the responsive front-end user interface and the Node.js middle tier, while Watson and its QAAPI are hosted in the Watson developer cloud.

Figure 3. Watson Films Architecture



Mapping the code to architecture

When you review the WatsonFilms code, you will see a small set of files and directories under the `watson_films_dw` project root. Those files and folders directly relate to the architectural components.

- **public/** – This directory contains all Watson Films client-side HTML, CSS, and JavaScript. It is served up by Node.js.
- **watson/** – This directory contains logic that is used by our REST API. It interacts directly with Watson by using the Watson QAAPI.
- **app.js** – This file starts the web server, which hosts our REST API and all of our client-side content.

Ingesting relevant content into Watson by using Watson Experience Manager

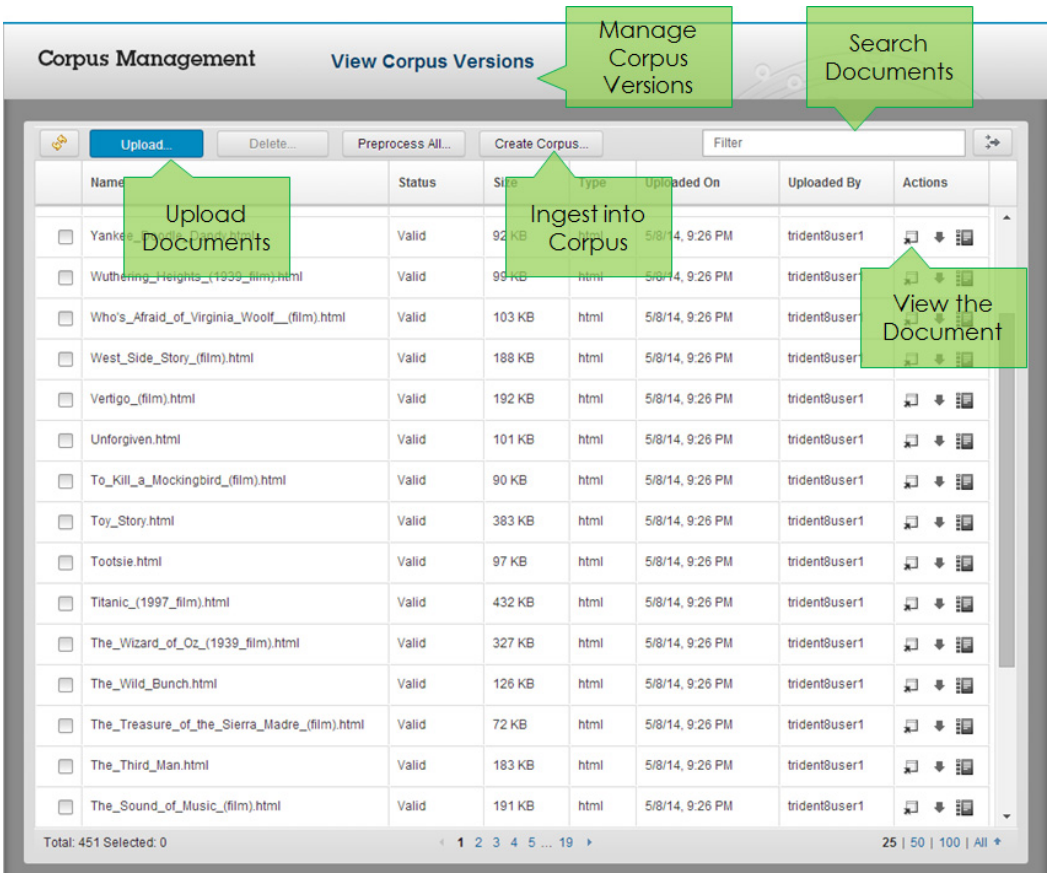
Watson provides a web-based set of tools that are called Watson Experience Manager. With the tools, you can:

- Upload and manage documents
- Create and manage a knowledge base
- Create training data to help teach Watson about your domain
- Test Watson by using an out-of-the-box UI by submitting questions and viewing returned answers or responses
- Monitor and view usage reports

Figure 4. Watson Experience Manager

Watson can ingest various content formats, including DOCX, DOC, PDF, HTML, and text. These formats represent most of unstructured content available. For the Watson Films application, we will ingest HTML content from Wikipedia about the 100 best American movies according to the American Film Institute. We will also ingest a PDF document from *The Essential Reference Guide for Filmmakers*. We will upload and ingest these documents with Watson Experience Manager.

Figure 5. Watson Experience Manager - Corpus Management



Build and run Watson Films

An IBM DevOps Services account is required to build and run the Watson Films demo locally. If you do not already have an DevOps Services account, you can create an account.

The Watson Films demo can be built either with Eclipse or without using Eclipse. Both methods are described next.

After you download the project and set it up locally, you need access to a Watson instance to run it end-to-end. You also need to change the URL and credentials in the `watson.js` file to reflect your own Watson instance. We provide a number of Watson QAAPI request and response JSON samples, which you can find under the `watson_films_dw/samples` folder.

For more information on Watson QAAPI JSON, see Inside IBM Watson.

Build and run with Eclipse/Nodeclipse

Step 1. Install Eclipse

1. Download and install Eclipse (v4.3.2 or greater).
2. Install Nodeclipse – Drag the **Install** button to the Eclipse toolbar.

Step 2. Get the code

The Watson Films code is contained in a Git repository hosted on DevOps Services. To get the code:

1. Open the Git perspective: **Window->Open Perspective->Git**.
2. Clone the repository: The source code is on the **origin/master** branch.

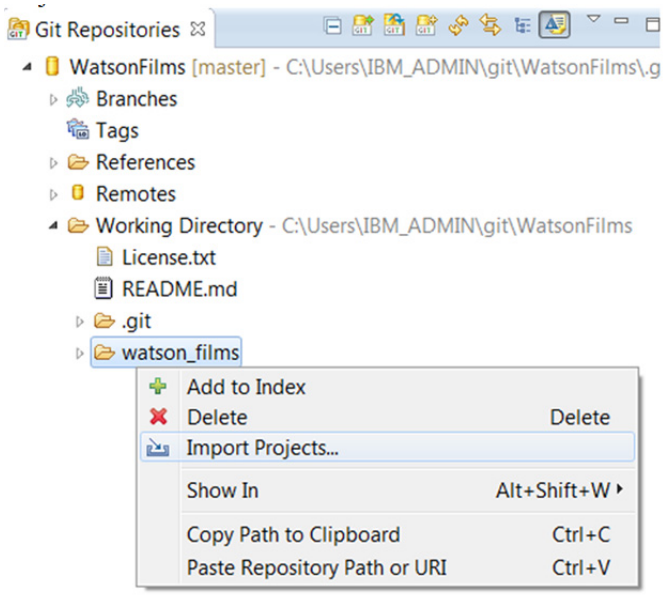
Figure 6. Clone the repository

3. Enter the following Git repository URL: **https://hub.jazz.net/project/dimascio/WatsonFilmsDW**.
4. Enter your DevOps Services credentials and complete the wizard.

Step 3. Import the WatsonFilm project into Eclipse

From the Git perspective, open the **Working Directory** folder and select **Import Projects**.

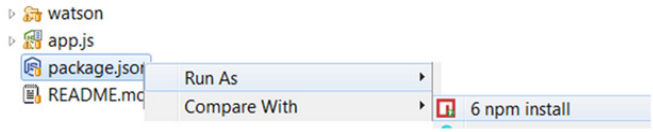
Figure 7. Import projects



Step 4. Install Dependencies

Select `package.json` and run `npm install` to install the dependencies, for example, Express. Installing dependencies is optional. All dependences are included in the repository.

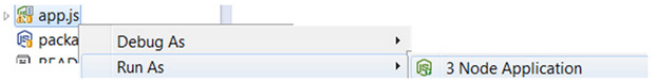
Figure 8. Install dependencies



Step 5. Run the code

From the Node perspective, select `app.js` and **Run As->Node Application**.

Figure 9. Run the code



Watson Films is now running on port 3001. Point your browser to http://localhost:3001 to play with the demo.

Build and run without Eclipse/Nodeclipse

1. Git clone https://*username*:*password*@hub.jazz.net/git/dimascio/WatsonFilmsDW, where *username* and *password* refer to your DevOps Services account.
2. Install Node.js.
3. Run `npm install express`.
4. Run `npm install request`.
5. Navigate into your cloned repository and the `WatsonFilmsDW/watson_films_dw` folder.
6. Run `node app.js`.

Watson Films is now running on port 3001. Point your browser to http://localhost:3001 to play with the demo.

Create a REST API with Node.js

To build our Watson Films REST API with Node.js, we will use the following two modules: Express.js and Request. We previously described how to install these modules by using `npm install`.

Step 1. Expose the REST API

Our REST API is simple. It contains a single resource, `/question`. When a POST request is made to `/question`, the business logic invokes Watson via the QAAPI.

With a few simple lines of Node code, the following snippet from `app.js` defines our question resource and creates and starts a web server.

Listing 1. Define question resource

```
// Get access to our Watson module
var watson = require('./watson/watson');
// Set up RESTful resources
// POST requests to /question are handled by 'watson.question'
app.post('/question', watson.question);

// Start the http server
http.createServer(app).listen(app.get('port'), function() {
  console.log('Express server listening on port ' + app.get('port'));
```

Step 2. Invoke the Watson QAAPI

In the previous section, we set a handler for all POST requests to `/question.` This handler is defined in a simple node module called `watson`. The file, `watson/watson.js`, is responsible for invoking Watson via the QAAPI.

Listing 2. Invoking Watson via the QAAPI

```
// Describe the Watson Endpoint
// Specify the information and credentials pertinent to your Watson instance
var endpoint = {
  // enter watson host name; for example: 'http://www.myhost.com'
  host : '',
   // enter watson instance name; for example: '/deepqa/v1/question'
  instance : '',
   // enter auth info; for example: 'Basic c29tZXVzZXJpZDpzb21lcGFzc3dvcmQ='
  auth : ''
};


// Handler for /question POST requests
// Submits a question to Watson via the IBM Watson QAAPI
// and returns the QAAPI response.
exports.question = function(req, res) {
 if (!endpoint.host) {
  res.send(404, 'Watson host information not supplied.');
  }
 var uri = endpoint.host + endpoint.instance;
 var request = require("request");
   // Form a proper Watson QAAPI request
 var questionEntity = {
  "question" : {
  "evidenceRequest" : { // Ask Watson to return evidence
  "items" : 5 // Ask for five answers with evidence
  },
  "questionText" : req.body.question // The question
  }
  };

  console.log('Ask Watson: ' + req.body.question + ' @ ' + uri);

  // Invoke the IBM Watson QAAPI Synchronously
  // POST the questionEntity and handle the QAAPI response
  request({
  'uri' : uri,
  'method' : "POST",
  'headers' : {
  'Content-Type' : 'application/json;charset=utf-8',
  'X-SyncTimeout' : 30,
  'Authorization' : endpoint.auth
  },
  'json' : questionEntity,
```

```
  }, function(error, response, body) {
  // Return the QAAPI response in the entity body
  res.json(body);
  });
}
```

So, what does this code do? First it exports the `question` function, our handler, so that it can be accessed from `app.js`. The `question` function takes two parameters, the request, `req`, and the response, `res`.

The request, `req`, is expected to be the string that contains the question text. The `question` string is then used to create a proper Watson QAAPI question request that conforms to the QAAPI JSON Request and Response model.

Finally, the `question` function invokes the Watson QAAPI synchronously by using the `request` module and returns the response "as-is."

Create the user interface

Now that we have a REST API capable of communicating with Watson, let's use this API to build a simple user interface for Watson Films.

All Watson Films user interface code can be found in the `watson_films_dw/public` directory. As one might expect, `index.html` and `movies.css` define the appearance, while `movies.js` is responsible for invoking our REST API and rendering the response.

Step 1. Invoke the Node.js REST API by using JQuery

To invoke our REST API, we make an Ajax POST request to the `/question` resource via JQuery.

The following snippet is from `movies.js`,

Listing 3. Invoke the REST API

```
  // Ask a question.
  // Invoke the Node.js REST service. The Node.js
  // service, in turn, invokes the IBM Watson QAAPI
  // and returns to us the QAAPI response
 var ask = function(question) {  var searchTerm = $("#searchTerm");
 var samples = $('.dropDownSampleQuestion');
  // Create a Ladda reference object
 var l = Ladda.create(document.querySelector('button'));
   ......
  l.start();
   // Form a question request to send to the Node.js REST service
 var questionEntity = {
  'question' : question
  };

  // POST the question request to the Node.js REST service
  $.ajax({
  type : 'POST',
  data : questionEntity,
  dataType : "json",
  url : '/question',
  success : function(r, msg) {
  // Enable search and stop the progress indicator
  searchTerm.removeAttr("disabled");
```

```
 samples.removeAttr("disabled");
  l.stop();
   // Display answers or error
 if (r.question !== undefined) {
  displayAnswers(r);
  } else {
  alert(r);
  }
  },
  error : function(r, msg, e) {
  // Enable search and stop progress indicator
  searchTerm.removeAttr("disabled");
  samples.removeAttr("disabled");
  l.stop();
   // Display error
 if (r.responseText) {
   alert(e+' '+r.responseText);   } else {
   alert(e);
  }
   }
  });
  };
```

Step 2. Render answers with Slick Carousel

Slick Carousel is a super cool carousel library. The Watson Films demo uses Slick to display answers that are returned by Watson. Each answer is rendered in a slide within the carousel. The following code snippets describe how to do it.

The following snippet is from `index.html:`

Listing 4. Render answers with Slick Carousel

```
<!-- placeholder for the slick carousel -->
<div class="col-lg-12">
   <div id="answerCarousel" class="single-item answer" style="margin-top: 10px;"></div>
  <iframe name="form-iframe" src="about:blank" class="hidden"></iframe>
</div>
```

And this snippet is from `movies.js:`

Listing 5. Snippet is from `movies.js`

```
 var displayAnswers = function(r) {
 var answerCarousel = $("#answerCarousel");
 var answerText = "Hmm. I'm not sure.";
  slickIndex = 0;

 if (r.question.answers[0] !== undefined) {
  answerText = r.question.answers[0].text
  console.log('answer: ' + answerText);
  slickIndex = r.question.answers.length;
  }

  answerCarousel.show();

 for (var i = 0; i < slickIndex; i++) {
  $('#panswer' + i).remove();
  answerCarousel.slickAdd(createAnswerSlide(i, r));
  }

  answerCarousel.slickGoTo(0);
  };
```

IBM DevOps Services and Bluemix

DevOps Services and Bluemix play an important role in creating the Watson Films application. DevOps Services provide a comprehensive set of easy-to-use services that helps you to develop, track, plan, and deploy your application simply and efficiently, such as:

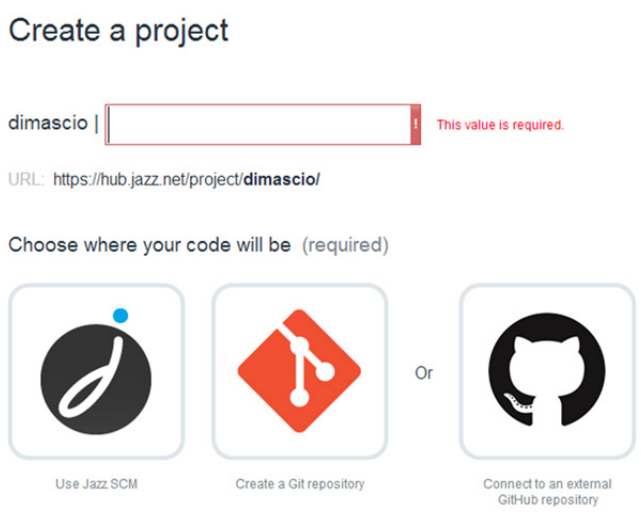
- A simple user interface that helps you quickly create your project
- Built-in source control management through Git or Jazz SCM
- Automatic build and deployment mechanisms that help you to publish your app with Bluemix
- integrated services complete with dashboards that help you to track and plan project activities

For a more detailed list of features, visit https://hub.jazz.net/features.

Create a project by using DevOps Services

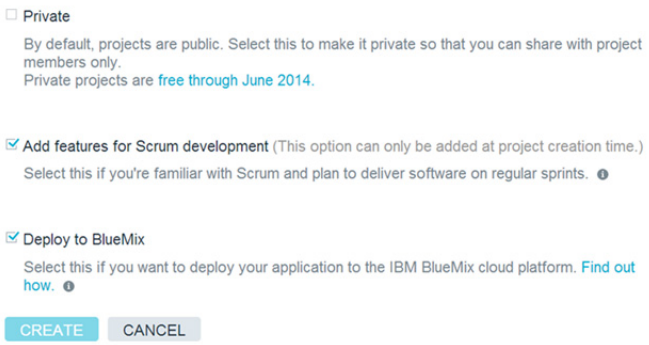
To create a new project, go to https://hub.jazz.net and click **Create Project**. Here you supply your project name, choose your source control management system, and set your planning and deployment preferences. For example:

Figure 10. Create project on DevOps Services



For Watson Films, we named our project, "Watson Films" and chose to create a new Git repository. Next, we set our project preferences. Is this project private? No. Do we want to use Scrum Development for tracking and planning? Sure. Do we want to deploy the application to Bluemix? Of course!

Figure 11. DevOps Services project settings

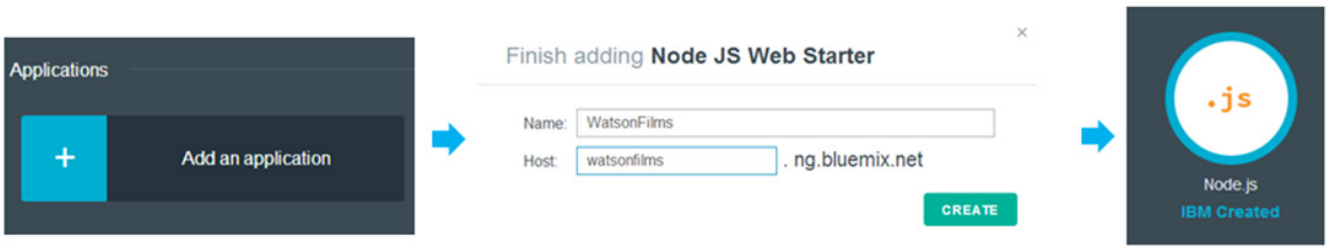


Create an application on Bluemix

Bluemix is an IBM open platform-as-a-service (PaaS) for developing and deploying mobile and web applications. Bluemix contains a vast array of pre-built services that make it easy for developers to build, enhance, and deploy applications. Bluemix allows developers to concentrate on what they do best—develop innovative, top-quality applications. You can get access to Bluemix by signing up here. Here's how to create an application in Bluemix:

1. Log in to Bluemix.
2. Click **Add an application** and select **Node.js**.
3. Name the application.

Figure 12. Creating Node.js based Watson Films application in Bluemix



Build and deploy from DevOps Services to Bluemix

The Watson Films source code is stored in DevOps Services. Using the "Build and Deploy" facility in DevOps Services, we configure a simple, one-time Grunt build to compile and "mini-fy" the project. We then create a one-time Deployer that deploys the resulting application on Bluemix.

1. Log in to DevOps Services.
2. Click **Build and Deploy**.
3. Enable **Simple**.
4. Click **Edit Code**.
5. Create the YAML file `manifest.yml` in the DevOps Services project root. (See the following information for details.)

Figure 13. Code structure in IBM DevOps Services

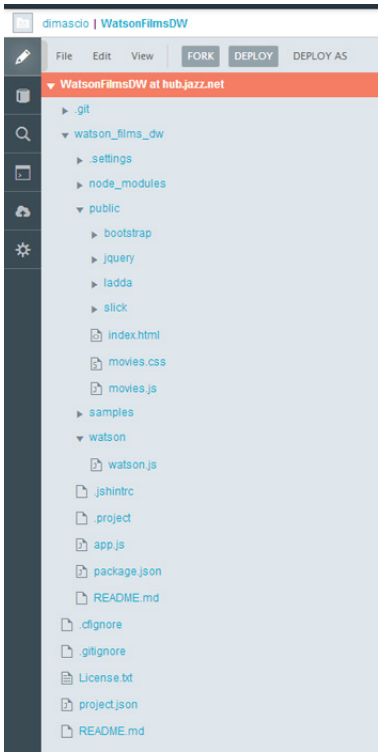


Figure 14. Manifest YAML File

```
---
applications:
- name: WatsonFilms
  mem: 512M #The maximum memory to allocate to each application instance
  instances: 1 #The number of instances of the application to start
  host: watsonfilms #Hostname for app routing.
  domain: mybluemix.net #Bluemix Parent Domain Name
  path: ./watson_films #Path to the application to be pushed
  command: node app.js #Node command, and application to start
```

The `manifest.yml` file provides the application's deployment configuration.

- **name** – the application name.
- **mem** – the amount of memory that is allocated to the application.
- **instances** – the number of cluster nodes that service the application. When the value of instances is greater than 1, Bluemix automatically handles load balancing.
- **command** – the command to start the application.
- **host** – the host name.
- **domain** – the domain name.
- **path** – the path to the working directory. The command runs from this directory.

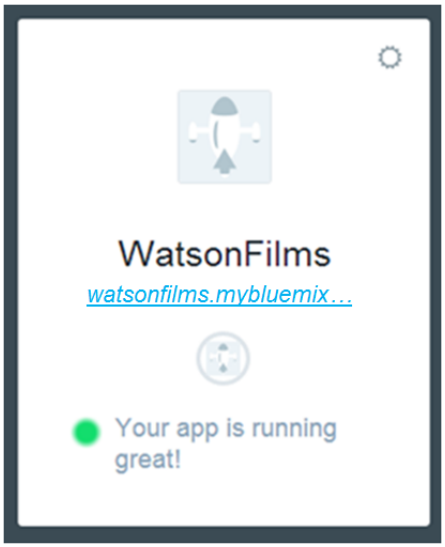
Deploy new changes to Bluemix

- Click **Edit Code** and press **Deploy**, OR
- Deliver to the branch that contains built and deployed source code!

Review the health of the application

Log in to Bluemix and click the application widget.

Figure 15. Application running in Bluemix



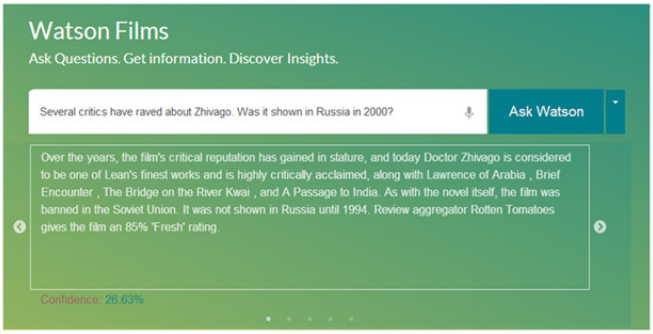
Watson Films in Action

The Watson Films application that is powered by Watson is now complete. Make sure that the application is started locally on Node.js or started in Bluemix. Start your favorite browser on your laptop or mobile device and point it to the application URL, for example, http://watsonfilmsdw.mybluemix.net/ or http://localhost:3001. The cognitive app you just wrote will appear. Click the drop-down icon to the right of the "Ask Watson" button for sample questions to try.

As you can see from Figure 16, the sample question "Several critics have raved about Zhivago. Was it shown in Russia in 2000?" was posed. Watson understood the question at a deeper level than just the individual words. If you look closely at the question, it is fairly ambiguous. One can easily mistake Zhivago for a restaurant in Chicago, or even a book. But it is about the movie *Dr. Zhivago*. By itself, "2000" in the question might appear as a number, but in reality it refers to a year. And Russia, is a geographic reference. As humans when we read a question or are asked a question, we automatically perform of this kind of linguistic analysis.

The answer reflects how Watson understood the question at a fairly deep level. Looking at the answer, we see how Watson knew how to apply context, fill in the gaps, resolve ambiguity and uncertainty, and accurately interpret language like a human. It understood Zhivago was indeed a movie and that 2000 refers to a year. It then came back with an answer *(…it was not shown in Russia until 1994…)* that was contained in the unstructured document. Watson understood the features of language as used by humans. From that was able to determine whether one text passage (call it the "question") infers another text passage (call it the "answer") with incredible accuracy under changing circumstances. To top it off, Watson provided insight into how it reached its conclusions by providing evidence.

Figure 16. Watson Films in action



Conclusion: Putting it all together

We crafted a basic cognitive application that demonstrates how to integrate Watson Q&A cognitive technology into an application, and hence a software project.

Watson is an incredibly complex and powerful platform. We explored the IBM Watson QAAPI, which provides a simple RESTful interface that enables a developer to harness the power of Watson. While we created Watson Films, we not only used Watson, but also used a number of cutting edge technologies, including Node.js and Bootstrap.js. Now that we've seen how easy it is to use the power of Watson, we can't wait to see the Watson-powered applications that you all dream up!

We explored IBM DevOps Services and saw how it can virtually eliminate the need to set up and maintain production servers for services like source control management, planning, and deployment. We saw how incredibly easy it is to create, manage, and deploy a software project by using IBM DevOps Services and Bluemix.

This Watson Films app that you created is a good starting point that can be expanded and enhanced with more cognitive capabilities, plus social and e-commerce capabilities. Some ideas include:

- Perform *named entity extraction* from the question and answer to automatically fetch movies from a movie streaming service catalog.
- Perform *sentiment analysis* on the question to understand the mood of the user, to offer relevant market products that are related to user preferences, such as movie tickets, rentals, and live shows.
- Perform *psycho-graphic analysis* from the user's linguistic footprint and automatically derive individual personality traits.
- Extract the *device location* information and offer location-specific information and insights.
- Recommend movies the user might like based on attributes from psycho-graphic analysis.
- Integrate with third-party services, including social media services, and transaction and e-commerce sites.

Cognitive systems like Watson and a PaaS like Bluemix make now a great time to be a developer. You can build great apps faster and more simply, and seamlessly infuse cognitive capabilities. Welcome to the future my fellow developers. Let your imaginations run wild!

Inside IBM Watson

How does Watson Q&A Service work?

Organizations today need to be able to deal with:

- The speed (velocity) of the incoming and changing big data (structured and unstructured)
- Their growing size (volume)
- Various inconsistent and often unpredictable forms of data (variety)
- The growing awareness that much of our data is uncertain (veracity)

Gathering insight and knowledge from data in this environment is becoming increasingly challenging.

Watson Q&A technology allows for users or other systems to interact with it using questions or inquires in natural language. It then understands the natural language, generates hypotheses, and provides answers and insights with supporting evidence, and a degree of confidence. Watson over time "learns and gets smarter" by:

1. Being taught by its users,
2. Learning from prior interactions
3. Being ingested with new information

Watson QAAPI JSON Request

The Watson Question and Answer API (QAAPI) is a Representational State Transfer (REST1) service interface that allows applications to interact with Watson. Using this API, one can pose questions to Watson, retrieve responses, and submit feedback on those responses. In addition to simple questions and responses, Watson can provide transparency into how it reached its conclusions through the REST services. Other functions, like ingesting content in the Watson platform, are exposed as tools.

To use the RESTful Watson QAAPI, POST a question with a JSON payload that looks like Listing 6:

Listing 6. Watson QAAPI Request JSON

```
{
    "question": {
        "questionText": "Several critics have raved about Zhivago. Was it shown in Russia in 2000?",
        "formattedAnswer": false,
        "evidenceRequest": {
            "items": "2"
        }
    }
}
```

The POST must include the HTTP header, including the following,

- `Accept: application/json`
- `Content-Type: application/json`
- `X-SyncTimeout: 30`

A value of 30 seconds exists for the X-SyncTimeout header element. This time represents how long the server waits after the question is submitted until it times out. The value does not represent how long the client waits for a response from the server.

Watson QAAPI JSON Response

After you submit a question, the response includes an HTTP status code. For synchronous mode, the successful HTTP status code is `200 Created`. The successful response also includes a complete question status and the answers. The answers section contains answers that are ranked by confidence. The value for confidence is a decimal percentage that represents the confidence Watson has in this answer. Higher values represent higher confidence. The text section contains the text of each answer.

The response also provides more natural language processing (NLP) information in the response, including details about the question's class, focus, lexical answer type, synonym list and more.

Listing 7. Watson QAAPI Response JSON

```
{
    "question": {
    "qclasslist": [
    {
        "value": "FACTOID"
    },
        {
        "value": "DESCRIPTIVE"
    }
    ],
    "focuslist": [
        {
        "value": "it"
        }
    ],
    "latlist": [
        {
        "value": "it"
        }
    ],
    "evidencelist": [
        {
            "value": "0.25249403715133667",
            "text": "Over the years, the film's critical reputation has gained in stature,
and today Doctor Zhivago is considered to be one of Lean's finest works and is highly
critically acclaimed, along with Lawrence of Arabia, Brief Encounter,
The Bridge on the River Kwai, and A Passage to India. As with the novel itself,
the film was banned in the Soviet Union. It was not shown in Russia until 1994.
Review aggregator Rotten Tomatoes gives the film an 85% 'Fresh' rating.",
            "id": "PB_FC6CC778B1703419C687B86A739779FF",
            "title": "Doctor Zhivago (film) - Wikipedia, the free encyclopedia : Doctor Zhivago (film) :
 Reception[edit]",
            "document": "/instance/8/deepqa/v1/question/document/
PB_FC6CC778B1703419C687B86A739779FF/3103/3563",
            "copyright": "No copyright specified for this supporting passage or document.",
            "termsOfUse": "No license specified for this supporting passage or document."
        },
        {
  "value": "0.10348163545131683",
  "text": "Many critics believed that the film's focus on the love story between Zhivago and Lara trivialized
the events of the Russian Revolution and the resulting civil war. The sweeping multi-plotted story form used
 by Pasternak
```

```
had a distinguished pedigree in Russian letters. The author of War and Peace, Leo Tolstoy, had used
 characters as symbols
of classes and historical events in describing the events in the Russia of Napoleonic times. Pasternak's
 father,
who was a painter, had produced illustrations for War and Peace.",
  "id": "PB_FC6CC778B170341996480AF5F45652821",
  "title": "Doctor Zhivago (film) - Wikipedia, the free encyclopedia",
  "document": "/instance/8/deepqa/v1/question/document/PB_FC6CC778B170341996480AF5F45652821/11749/12270",
  "copyright": "No copyright specified for this supporting passage or document.",
  "termsOfUse": "No license specified for this supporting passage or document."
  }
    ],
    "synonymList": [
  {
  "partOfSpeech": "verb",
  "value": "shown",
  "lemma": "show",
  "synSet": [
  {
  "name": "Wordnet_show-verb-1",
  "synonym": [
  {
  "isChosen": true,
  "value": "present",
  "weight": 1
  },
  {
  "isChosen": true,
  "value": "demo",
  "weight": 1
  },
  {
  "isChosen": true,
  "value": "demonstrate",
  "weight": 1
  },
  {
  "isChosen": true,
  "value": "exhibit",
  "weight": 1
  }
  ]
  },
  {
  "name": "Lemma_Expansion",
                   "synonym": [
  {
  "isChosen": true,
  "value": "show",
  "weight": 1
  }
      ]
  }
  ]
  },
  {
  "partOfSpeech": "verb",
  "value": "raved",
  "lemma": "rave",
  "synSet": [
  {
  "name": "Lemma_Expansion",
  "synonym": [
  {
  "isChosen": true,
  "value": "rave",
  "weight": 1
```

```
}
]
}
]
},
{
"partOfSpeech": "noun",
          "value": "russia",
          "lemma": "russia",
          "synSet": [
              {
"name": "Wordnet_soviet+union-noun-1",
"synonym": [
                      {
"isChosen": true,
"value": "ussr",
"weight": 1
}
]
              }
          ]
},
{
"partOfSpeech": "verb",
"value": "was",
"lemma": "be",
"synSet": [
{
"name": "Lemma_Expansion",
"synonym": [
{
"isChosen": true,
                        "value": "be",
"weight": 1
}
]
}
]
},
{
"partOfSpeech": "noun",
        "value": "critics",
"lemma": "critic",
"synSet": [
{
"name": "Lemma_Expansion",
"synonym": [
{
"isChosen": true,
"value": "critic",
"weight": 1
}
]
}
]
},
{
"partOfSpeech": "verb",
"value": "have",
"lemma": "have",
"synSet": [
{
"name": "Wordnet_have-verb-1",
"synonym": [
{
"isChosen": true,
"value": "hold",
```

```
  "weight": 1
  }
  ]
  }
  ]
  }
  ],
  "pipelineid": "1635768506",
  "formattedAnswer": false,
  "category": "",
  "items": 2,
  "status": "Complete",
  "id": "4A242FC03922427EBAB78669CC8DC4A3",
  "questionText": "Several critics have raved about Zhivago. Was it shown in Russia in 2000?",
  "evidenceRequest": {
  "items": 1,
  "profile": "Yes"
  },
  "answers": [
  {
  "id": 0,
            "text": "Over the years,
the film's critical reputation has gained in stature, and today Doctor Zhivago is considered to be one of
 Lean's
finest works and is highly critically acclaimed, along with Lawrence of Arabia, Brief Encounter,
The Bridge on the River Kwai, and A Passage to India. As with the novel itself, the film was banned in the
 Soviet Union.
It was not shown in Russia until 1994. Review aggregator Rotten Tomatoes gives the film an 85% 'Fresh'
 rating.",
  "pipeline": "Descriptive",
  "confidence": 0.25249,
  "evidenceProfile": [
  {
  "name": "FeatureGroup_Linguistic",
                    "value": "0.23145645026735637"
  },
                {
  "name": "FeatureGroup_Type",
  "value": "0.2690169077609411"
  },
  {
  "name": "FeatureGroup_Spatio-temporal",
   "value": "0.24976332098585127"
  },
  {
  "name": "FeatureGroup_Relevance",
  "value": "0.24976332098585127"
  }
  ]
   },
  {
  "id": 1,
  "text": "Many critics believed that the film's focus on the love story between Zhivago and Lara trivialized
the events of the Russian Revolution and the resulting civil war. The sweeping multi-plotted story form used
 by Pasternak
had a distinguished pedigree in Russian letters. The author of War and Peace, Leo Tolstoy, had used
 characters as symbols
of classes and historical events in describing the events in the Russia of Napoleonic times. Pasternak's
 father,
who was a painter, had produced illustrations for War and Peace."
  "pipeline": "Descriptive",
            "confidence": 0.10348,
            "evidenceProfile": [
  {
  "name": "FeatureGroup_Linguistic",
  "value": "0.21214725301293394"
  },
```

```
  {
  "name": "FeatureGroup_Type",
  "value": "0.2757757967510807"
  },
           {
  "name": "FeatureGroup_Spatio-temporal",
  "value": "0.25603847511799266"
  },
  {
  "name": "FeatureGroup_Relevance",
       "value": "0.25603847511799266"
  }
   ]
         }
  ],
  "errorNotifications": [],
  "passthru": ""
  }
}
```


The Question and Answer servicehttp://www.ibm.com/developerworks/topics/question and answer serviceinterprets and answers user questions directly based on primary data sources that have been selected and gathered into a body of data.

RELATED TOPICS: The Era of Cognitive Systems: An Inside Look at IBM Watson and How it Works (PDF) Creating Cognitive Applications Powered by Watson: Getting started with the API (PDF) Watson Developer Q&A Forum

About the authors

Swami Chandrasekaran

@swamichandra on Twitter

Carmine DiMascio

@CarmineDiMascio on Twitter