Redbooks

ibm.com/redbooks

Enhancing the IBM Power Systems Platform with IBM Watson Services

Ahmed Azraq

Soheel Chughtai

Ahmed (Mash) Mashhour

Duy V Nguyen

Reginaldo Marcelo Dos Santos

Cloud

Power Systems

IBM

Redbooks

International Technical Support Organization

Enhancing the IBM Power Systems Platform with IBM Watson Services

April 2018

SG24-8419-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xv.

First Edition (April 2018)

This edition applies to IBM Power Systems servers running AIX Version 7.2, IBM i Version 7.2, and Linux on Power RHEL 7.4 invoking IBM Watson services on IBM Cloud, IBM Watson Tone Analyzer Version 3 API Version 2017-09-21, IBM Watson Language Translator Version 2 Default API Version, and IBM Watson Natural Language Understanding Version 1 API Version 2017-02-27.

© Copyright International Business Machines Corporation 2018. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2018. All rights reserved.

Figures

© Copyright IBM Corp. 2018. All rights reserved.

Tables

© Copyright IBM Corp. 2018. All rights reserved.

Examples

© Copyright IBM Corp. 2018. All rights reserved.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

© Copyright IBM Corp. 2018. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

AIX®
Bluemix®
ClearCase®
Concert™
DB2®
IBM®
IBM Watson®
Integrated Language Environment®
Language Environment®
POWER®
Power Architecture®
Power Systems™
POWER5®
POWER6®
POWER6+™
POWER7®
POWER7+™
POWER8®
POWER9™
PowerHA®
PowerLinux™
PowerPC®
PowerSC™
PowerVM®
Rational®
Rational Team Concert™
Redbooks®
Redbooks (logo) ®
RS/6000®
System i®
Tivoli®
Watson™
Watson Health™
WebSphere®
z/Architecture®
z10™

The following terms are trademarks of other companies:

The Weather Company, and Wundersearch are trademarks or registered trademarks of TWC Product and Technology LLC, an IBM Company.

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbooks® publication provides an introduction to the IBM POWER® processor architecture. It describes the IBM POWER processor and IBM Power Systems™ servers, highlighting the advantages and benefits of IBM Power Systems servers, IBM AIX®, IBM i, and Linux on Power.

This publication showcases typical business scenarios that are powered by Power Systems servers. It provides an introduction to the artificial intelligence (AI) capabilities that IBM Watson® services enable, and how these AI capabilities can be augmented in existing applications by using an agile approach to embed intelligence into every operational process. For each use case, the business benefits of adding Watson services are detailed.

This publication gives an overview about each Watson service, and how each one is commonly used in real business scenarios. It gives an introduction to the Watson API explorer, which you can use to try the application programming interfaces (APIs) and their capabilities. The Watson services are positioned against the machine learning capabilities of IBM PowerAI.

In this publication, you have a guide about how to set up a development environment on Power Systems servers, a sample code implementation of one of the business cases, and a description of preferred practices to move any application that you develop into production.

This publication is intended for technical professionals who are interested in learning about or implementing IBM Watson services on AIX, IBM i, and Linux.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Ahmed Azraq is a Cloud Solution Leader in IBM Egypt. He joined IBM in 2012 and recently joined the Global IBM Cloud Services and Solutioning East Hub organization. His primary responsibility is to help clients across the Middle East and Africa (MEA) and Asia Pacific to adopt IBM Cloud and IBM Watson. Ahmed has several professional certifications, including IBM Certified Solution Advisor - Cloud Reference Architecture, and IBM Certified Application Developer - Cloud Platform. Ahmed has delivered training on IBM Cloud and Watson APIs to IBM clients, and university students around the world. Ahmed acquired his IBM Redbooks Platinum Author badge for authoring several IBM Redbooks publications.

Soheel Chughtai is an IBM Early Program Manager based at the IBM Hursley lab in the UK. He has over 30 years of programming experience, including assembly language, C/C++, Java, Python, and Node.js. Soheel is the owner and lead developer for the Watson nodes on Node-RED project. Soheel uses Node-RED and the Watson nodes as a teaching tool for developers and non-developers alike, for which he has developed several courses. As a consequence, he both attends and runs hackathons regularly. He has worked at IBM for 31 years. Over that time, he has worked hard to obtain and upgrade the skills, knowledge, and adaptability it takes to be hands-on.

© Copyright IBM Corp. 2018. All rights reserved.

Ahmed (Mash) Mashhour is a Power Systems Global subject matter expert (SME). He is IBM AIX, Linux, and IBM Tivoli® certified with 11 years of professional experience in IBM AIX and Linux systems. He is an IBM AIX back-end SME, working on supporting several customers in the US, Europe, and the Middle East. His core experiences are in IBM AIX, Linux systems, clustering management, and visualization tools for various Tivoli and database products. Mash has authored several publications inside IBM. He has hosted more than 60 classes worldwide.

Duy V Nguyen is an IBM Master Certified IT Architect with experience in IBM and open technologies. He is a Lead Architect/Cloud Engineer serving of the Cloud Engineering and Services team in the IBM CIO Office, Transformation and Operations group at IBM CHQ/US. His daily job is helping IBM to transform by using new technologies, specifically cloud, mobile, and other emerging technologies. He is focusing on the creation of cloud and mobile solutions for IBM employees, and provides his expertise in assisting IBM clients with enterprise mobile and cloud engagements as needed. His core experiences are in web, security, cloud, and mobile technologies.

Reginaldo Marcelo Dos Santos is a Senior IT Specialist with IBM Virtual Integrated Organization (VIO) in Brazil. He is a Certified Cloud Platform Application Developer with 5 years of professional experience in IBM Cloud technologies. He is a Cloud Application Developer, working on a high-performance dashboard for internal executives in North America. His core experiences are in application development, mobile first, and back-end and front-end development. Reginaldo has been a technical speaker for IBM Watson APIs at various developer conferences. He is a mentor in hackathons and in the IBM Cognitive community.

The project that created this publication was managed by:
Scott Vetter, PMP

Thanks to the following people for their contributions to this project:

Ryan Anderson, Faheem Altaf, Timothee Bouhour, Ingo Dimmer, Bo Fan, Jesse Gorzinski, Bimal Kumar Jha, Laksh Krishnamurthy, Daniel E. Laun, Nin Lei, Chris Parsons, Steven Roberson, Tim Rowe, Scott Soutter, Alise Spense, Javier Bazan Lazcano Teran
IBM

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Introduction to IBM Power Systems servers

This chapter explains IBM POWER processor-based systems and IBM Power Systems servers, including AIX, Linux on Power, and IBM i. It also describes the advantages and benefits of the IBM Power Systems architecture.

The chapter includes the following topics:

© Copyright IBM Corp. 2018. All rights reserved.

1.1 POWER processor-based architecture

In February 1990, the first computers from IBM to incorporate the POWER processor were the Reduce Instructions Set Computer (RISC) System/6000 or IBM RS/6000® servers. These RS/6000 computers were divided into two classes, workstations and servers, and were introduced as the POWER station and POWER server.

Note: POWER is Performance Optimized With Enhanced RISC.

The RS/6000 CPU had two configurations, which were named RIOS-1 and RIOS.9 (or more commonly, the POWER1 CPU). A RIOS-1 configuration had a total of 10 discrete chips, including an instruction cache chip, a fixed-point chip, a floating-point chip, four data L1 cache chips, a storage control chip, input/output chips, and a clock chip. It was the first microprocessor that used register renaming and out-of-order execution. A simplified and less powerful version of the 10-chip RIOS-1 was made in 1992 for lower-end RS/6000 servers. It used only one chip and was named RISC Single Chip (RSC).

Note: RIOS is Riverbed Optimization System. The lower-cost RIOS.9 configuration had eight discrete chips, including an instruction cache chip, a fixed-point chip, a floating-point chip, two data cache chips, a storage control chip, an input/output chip, and a clock chip.

1.1.1 IBM POWER5

The IBM POWER5® processors were built on the popular POWER4 processors, and incorporated simultaneous multithreading (SMT) into the design, which was a technology that was introduced in the IBM PowerPC® AS-based RS64-III processor, and on-die memory controllers. It was designed for multiprocessing on a massive scale and came in multi-chip modules with onboard large L3 cache chips.

POWER5 processors

The POWER5 processors have two configurations:

- POWER5: The iconic setup with four POWER5 chips and four L3 cache chips on a large multi-chip module.
- POWER5+: A faster POWER5 that is fabricated by a reduced process to reduce power consumption.

1.1.2 IBM POWER6

IBM POWER6® was developed by the eCLipz Project, joining the IBM i (AS/400), P (RS/6000), and Z (Mainframe) instruction sets under one common platform. IBM System i® and System p were already built with the POWER4, but the eCLipz effort failed to include the CISC-based IBM z/Architecture®, and the IBM z10™ processor became the POWER6 eCLipz sibling. z/Architecture remains a separate design track to this day that is not related to the IBM Power Architecture® instruction set in any way. Because of eCLipz, the POWER6 processor is an unusual design because it is aimed at high frequencies and sacrifices out-of-order execution, something that has been a feature for POWER and PowerPC processors since their inception. POWER6 also introduced the decimal floating point unit (FPU) to the Power ISA, which is something that it shares with z/Architecture.

Note: With the POWER6, in 2008 IBM merged the former System p and System i server and workstation families into one family that is called Power Systems. Power Systems machines can run different operating systems (OSs), such as AIX, IBM i, and Linux on Power.

POWER6 processors

The POWER6 processors have two configurations:

- POWER6: Reaches 5 GHz, comes in modules with a single chip on it, and has an MCM with two L3 cache chips.
- IBM POWER6+™: A minor update that is fabricated by the same process as POWER6.

1.1.3 IBM POWER7

The IBM POWER7® symmetric multiprocessor design was a substantial evolution from the POWER6 design, focusing more on power efficiency through multiple cores, SMT, and out-of-order execution. The eight-core chip can run 32 threads in parallel, and has a mode in which it can disable cores to reach higher frequencies in the ones that are left. It uses a new high-performance FPU that is called Vector Scalar Extension (VSX) that merges the functions of the traditional FPU with AltiVec. Even though the POWER7 chips run at lower frequencies than POWER6, each POWER7 core perform faster than its POWER6 counterpart.

POWER7 processors

The POWER7 processors have two configurations:

- POWER7: Comes in single-chip modules or in quad-chip MCM configurations for supercomputer applications.
- IBM POWER7+™: Uses a scaled-down fabrication process, and has an increased L3 cache and frequency.

1.1.4 IBM POWER8

IBM POWER8® is a 4 GHz, 1-core processor with eight hardware threads per core for a total of 96 threads of parallel execution. It uses 96 MB of eDRAM L3 cache on chip, a 128 MB off-chip L4 cache, and a new extension bus that is called Coherent Accelerator Processor Interface (CAPI) that runs on top of PCIe, replacing the older GX bus. The CAPI bus can be used to attach dedicated off-chip accelerator chips, such as graphics processing units (GPUs), application-specific integrated circuits (ASICs), and field-programmable gate arrays (FPGAs). IBM states that it is 2 - 3 times faster than its predecessor, the POWER7. It has been built by a 22-nm process since 2013 or early 2014. In December 2012, IBM began submitting patches to the 3.8 version of the Linux kernel to support new POWER8 features, including the VSX-2 instructions.

1.1.5 IBM POWER9

The IBM POWER9™ is the first to incorporate elements of the Power ISA version 3.0 that was released in December 2015, including the VSX-3 instructions, and will also incorporate support for the Nvidia NVLink bus technology.

The United States Department of Energy together with Oak Ridge National Laboratory and Lawrence Livermore National Laboratory have contracted IBM and Nvidia to build two supercomputers, the Summit and the Sierra, which will be based on POWER9 processors that are coupled with Nvidia Volta GPUs. POWER9 is manufactured by using a 14-nm FinFET process, and will come in at least two versions, which are two 24-core versions for scale-up and scale-out applications, and possibly more because the POWER9 architecture is open for licensing and modification by the OpenPOWER Foundation members.

Note: The focus is on a high core count and high-performance I/O. It is planned to be built on a 10-nm technology.

Continuous improvements are still being done to enhance POWER processor reliability and scalability.

1.2 Benefits of Power Systems servers

The IBM Power Enterprise Solution Editions are available with eligible solutions from the IBM software portfolio and from third-party independent software vendors (ISVs). IBM is working in partnership with the ISVs to select business applications that have a close affinity with the Power Systems server hardware, OSs, and IBM middleware to provide the finely tuned solutions that your business needs.

Performance management for Power Systems capabilities and benefits provides:

- A comprehensive, low-cost, secure, and easy to use systems management tool to help ensure that your systems are ready to meet today's and tomorrow's business challenges and opportunities.
- Easily understood reports at a system, processor pool, or partition level help identify potential resource constraints affecting performance and capacity, which helps you avoid surprises.
- Help for making it easier to understand the impact on your information technology (IT) environment of the sophisticated Power System functions. For example, dynamic processor allocation, rapid logical partition (LPAR) provisioning, processor pools, capped versus uncapped partitions, dedicated versus shared processors, active memory sharing, active memory expansion, Live Partition Mobility (LPM), Capacity on Demand (CoD) processors, and multiple OS types.
- A process to easily size your future system requirements based on the growth trends of your existing workload, which helps you more easily plan for the future.
- A simple methodology to help with understanding the impact of running your existing workloads in a virtualized environment by taking advantage of logical partitioning, server consolidation, IBM middleware, or the inclusion of another OS type in to your existing or future Power Systems server.
- A self-managing process to track CPU, memory, disk, and other key system measurements automatically, which helps reduce the need for dedicated performance management and capacity planning personnel.
- A systematic way to help with the proactive (as opposed to reactive) planning of your potential financial requirements to keep your system running at its peak efficiency and meet the growing uses of IT in your business.

Advantages of Power Systems servers

Power Systems servers have several advantages:

- Deliver services faster, with higher quality and superior economics.
- Design optimized systems for specific business services and workloads.
- Select the optimal operating environment from AIX, IBM i, and Linux on Power.
- Consolidate and reduce cost with IBM PowerVM® virtualization.
- Deliver resiliency without downtime with IBM PowerHA® and IBM Storage.
- Build systems on the foundation of POWER processor technology.

Everyone knows what “performance” meant for IT in the past. Built on the foundation of POWER processor technology, Power Systems servers continue to excel and extend industry leadership in traditional benchmarks of performance.

But the IT landscape is evolving rapidly. As processes become more interrelated and complex, IT must solve challenging new problems and implement new projects, both with higher service levels and in a more cost-effective manner. IBM has the systems, software, and expertise to help clients implement projects that make their IT an enabler of innovation, a catalyst for business change, and deliver services faster with higher quality and superior economics.

Today’s IT performance means delivering services faster, with higher quality and with superior economics. The emerging measures of IT performance are around agility and the ability to help the business capitalize on new opportunities. IT is measured on providing an infrastructure that can handle rapid growth and manage business risk while meeting higher required service levels. The new services must be delivered within tighter budget constraints, with IT expected to do more with less and find the lowest cost solutions possible.

Intelligence is increasingly being embedded in every operational process: supply chain management, human resources and payroll, and financial, security and risk management. As more of the world becomes instrumented, such as roadways, power grids, consumer goods, and food, businesses need the ability to analyze the big data coming from these sources in real time.

Optimized systems increasingly are designed to address the unique needs of specific workloads and big data analytics, combining servers and software with critical skills and domains.

Watson is emblematic of this new approach of optimized systems design in the era of smarter computing. It combines IBM Deep QA analytics software, POWER processor-based servers, and innovative skills from IBM Research. Watson was designed to answer Jeopardy! questions posed in natural language in less than 3 seconds, by using massively parallel POWER processor performance to run thousands of complex analytics tasks simultaneously. However, Watson is not a supercomputer. It uses commercially available IBM Power Systems servers that are also deployed by thousands of businesses today to run optimized systems for everything from complex analytics to transaction processing.

Watson represents an impressive leap forward in systems design and analytics. Its IBM DeepQA software pioneers a new kind of analytics for unstructured text and big data. Powered by IBM POWER processor-based technology, Watson is an example of the complex analytics systems that are becoming increasingly common and critical to business success and competitiveness in today’s data-intensive environment.

1.3 Power Systems operating systems

This section describes different IBM Power Systems OSs, including AIX, Linux on Power, and IBM i.

1.3.1 IBM AIX

IBM AIX is an enterprise-class UNIX OS for the POWER processor architecture that is found in IBM Power Systems servers. Today's global business relies on an infrastructure that is secure, highly available, and can adapt quickly to changing business needs. AIX delivers these capabilities and more, with the performance, reliability, and security that your mission-critical data demands. It has a 30-year history of consistently delivering a reliable, flexible, and high performing secure environment. AIX maintains a strong, long-standing security focus and reputation.

Security features include Trusted AIX to easily harden the security settings of the system, and Trusted Execution to control the integrity of the system.

Users can apply hot patches since AIX 6.1, and IBM Support delivers concurrent update interim fixes that enable kernel patches without a restart. Still, in some situations, fixes cannot be coded with this approach.

AIX Live Update now enables all interim fixes, service packs, and technology levels to be applied without the need to restart by using LPAR virtualization, AIX virtualization, and alt-disk installation technology. AIX Live Update can do the following tasks:

- Create an LPAR with an updated running kernel.
- Provide a live migration while running workloads to this new LPAR.
- Use Enterprise Pools to find/enable necessary resources.

1.3.2 IBM i

IBM i is an integrated operating environment with a reputation for robust architecture, exceptional security, and business resilience for over 25 years. Running applications based on IBM i has helped companies over many years to focus on innovation and deliver new value to their business. IBM i uses innovative technology to run multiple applications in a single environment, virtualizes storage with its single-level storage, and integrates security, IBM DB2® for i, and other middleware. The combination of IBM i and IBM PowerVM deliver powerful virtualization technologies, and together with IBM Power Systems servers offer an integrated solution for managing unpredictable growth.

It can also provide comprehensive solution capabilities for a highly secure system environment. Sophisticated technologies combine to minimize the potential risk that is posed by security threats, while better enabling you to rapidly adapt and respond to changing security policy requirements.

IBM i offers an integrated Language Environment® (ILE) that supports a broad range of open application options, with best in class IBM Rational® development tools and a wide range of options from IBM tools partners. The ILE for IBM i enables companies to use existing application assets, while taking advantage of new business opportunities with a range of open technologies. Various technologies can be used to implement a service-oriented architecture (SOA), integrating traditional transaction processing language environments with today's open source technologies.

1.3.3 Linux on Power

Linux on Power can be Red Hat, SUSE, or Ubuntu Linux. With IBM PowerLinux™ processor compatibility modes, you can run OS versions that use all the standard features of a previous generation of POWER processor. A processor compatibility mode is a value that is assigned to an LPAR by the hypervisor. It specifies the processor environment on which the LPAR can successfully operate.

You can run several versions of Power Systems servers. Sometimes, older versions of these operating environments do not support the capabilities that are available with new processors, thus limiting your flexibility to move LPARs between servers that have different processor types. Support for new processors is typically added only to new Linux distribution releases.

You can move LPARs between servers that have different processor types without upgrading the operating environments that are installed in the LPARs. When you move an LPAR to a destination server that has a different processor type from the source server, the processor compatibility mode enables that LPAR to run in a processor environment on the destination server. The processor compatibility mode enables the destination server to provide the LPAR with a subset of processor capabilities that are supported by the operating environment that is installed in the LPAR.

You can determine processor compatibility mode in advance and at run time. In addition, you can optimize applications for performance and to benefit from processing features available only on some processors.

2

Introduction to IBM Watson services and the Representational State Transfer architecture

The chapter includes the following topics:

© Copyright IBM Corp. 2018. All rights reserved.

2.1 Introducing IBM Watson

IBM Watson (its logo is shown on Figure 2-1) is the AI platform from IBM that enables a new partnership between people and computers.

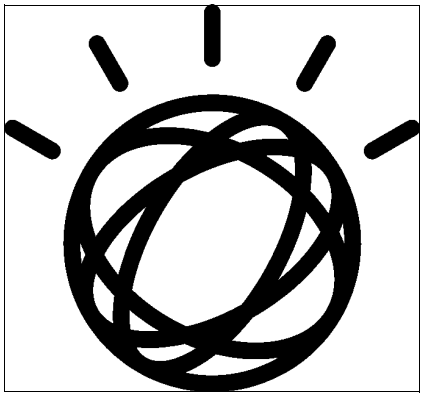

Figure 2-1 IBM Watson

Watson combines six core capabilities:

- Accelerate research and discovery by conducting rigorous, domain-specific research faster, and discovering insights and new opportunities by crawling through diverse data sources and extracting information.
- Enrich interaction by understanding and communicating with customers and employees around the clock, responding to their needs with tailored dialogue and personalized, adaptive experiences.
- Anticipate and preempt disruptions by monitoring the condition of your systems and equipment always, enabling you to detect potential issues before they lead to bigger, more expensive problems down the road.
- Recommend with confidence by making more confident, targeted recommendations by drawing from a broad set of information and understanding the nuances of your current context.
- Scale expertise and learning by elevating the expertise of every employee by collecting individual know-how from your organization and combining it with the current lessons that are learned in your industry, creating a deep source of knowledge that is available to every employee on-demand.
- Detect liabilities and mitigate risk by understanding the written language of threat reports, regulations, and new findings, keeping your business up to date and shielding it from risk on all sides.

IBM Watson is at the forefront of a new era of computing: cognitive computing. In summary, Watson can understand all forms of data, interact naturally with people, and learn and reason, at scale.

Data, information, and expertise create the foundation for working with Watson. Figure 2-2 shows examples of data and information that Watson can analyze and learn from, and derive new insights that were never discovered before.

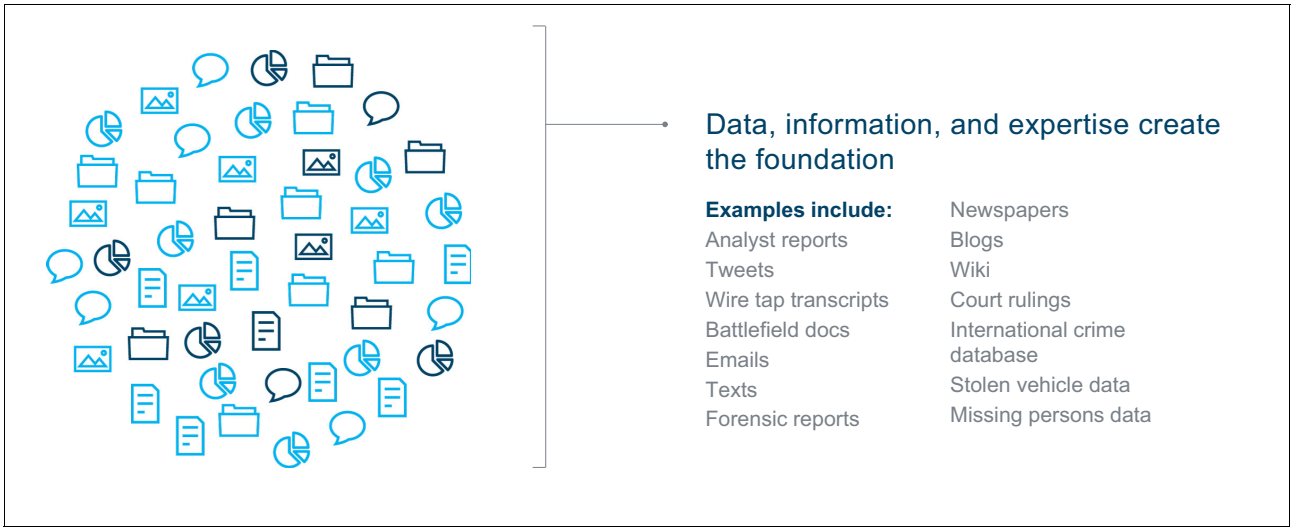

Figure 2-2 Watson relies on collections of data and information

Watson is available as a set of open application programming interfaces (APIs) and software as a service (SaaS) industry solutions. This book focuses on the Watson APIs, which are made available as Watson services through IBM Cloud.

2.1.1 Watson services overview

IBM Cloud provides a cloud-hosted marketplace where application providers of all sizes and industries can tap into resources for developing applications. Developers can combine the Watson services with other services that are available in IBM Cloud and other providers. They can combine these services with extra logic and components on Power Systems servers to build AI applications (Figure 2-3). This solution enables a worldwide community of software application providers to build a generation of applications that are infused with Watson AI capabilities.

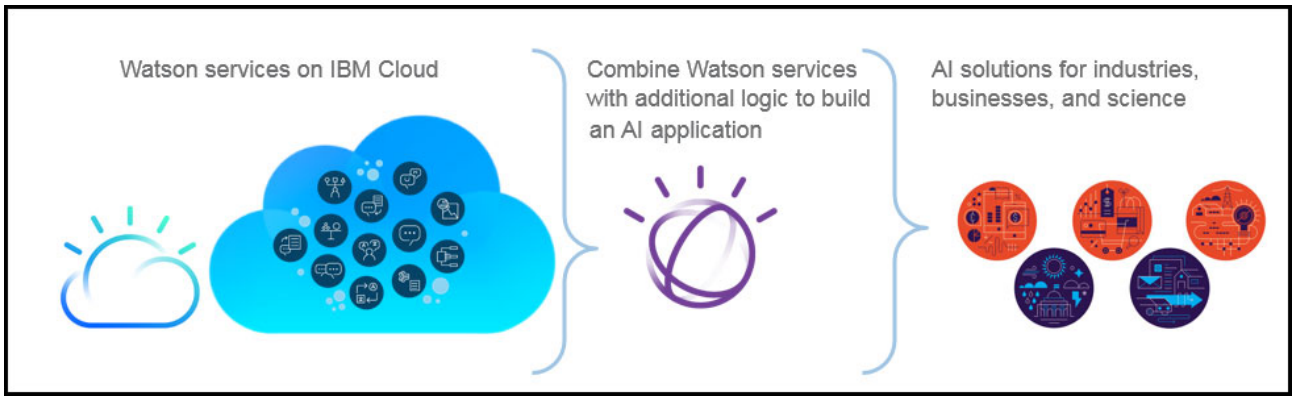

Figure 2-3 Building AI solutions with IBM Watson services on IBM Cloud

Note: IBM Cloud is an open cloud computing platform that combines platform as a service (PaaS) with infrastructure as a service (IaaS), and includes a catalog of diverse cloud services, which can be used to rapidly build and deploy business applications or infrastructure.

As a PaaS, it provides developers access to create applications by using a set of run times such as Node.js, Python, and Java. It also provides services for integration, Watson, data, security, and other key functions.

The application types can range from web, mobile, big data, and smart devices to the Internet of Things (IoT).

As an IaaS, it allows developers control over the infrastructure on which their applications are deployed. Developers can select bare metal servers, virtual servers, containers, and cloud storage, in IBM Cloud data center locations around the world.

To see what you can accomplish with IBM Cloud, see the following video.

The following Watson services are available on IBM Cloud at the time of writing:

Watson Assistant	Adds a natural language interface to your application.
Watson Discovery	Adds a cognitive search and content analytics engine.
Watson Language Translator	Translates text from one language to another for specific domains.
Watson Natural Language Classifier	Performs natural language classification on question texts.
Watson Natural Language Understanding (NLU)	Analyzes text to extra metadata from content.
Watson Personality Insights	Derives insights to identify psychological traits.
Watson Speech to Text	Low-latency, streaming transcription.
Watson Text to Speech	Synthesizes natural-sounding speech from text.
Watson Tone Analyzer	Uses linguistic analysis to detect tones from communications.
Watson Visual Recognition	Analyzes images for scenes, objects, faces, and other content.

For an updated list, see the Watson products and services page.

Watson services that are available on IBM Cloud are exposed as Representational State Transfer (REST) APIs. You learn more about REST APIs in 2.2.1, “Representational State Transfer Architecture” on page 37.

The next sections provide an overview for each Watson service. Take your time to play with the demonstrations, watch the videos, and learn from a generalized use case scenario for each service. The generalized use cases help stimulate you into thinking about scenarios that are related to your business that can use Watson services on Power Systems servers.

2.1.2 Watson Assistant

The Watson Assistant service enables learning to respond to customers in a way that simulates a conversation between humans. Common applications include virtual agents and chatbots that can integrate and communicate through any channel or device, including mobile devices, messaging platforms, and robots. You can train the Watson Assistant service through an easy-to-use web interface that is designed so that you can quickly build natural conversation flows between your apps and users, and deploy scalable, cost-effective solutions.

For an overview of the Watson Assistant service, see this YouTube video.

Demonstration

Figure 2-4 shows a demonstration application that uses the Watson Assistant service where Watson Assistant is trained on a specific set of car capabilities. Watson can understand your entries and respond correctly. Try asking “Where is the nearest restaurant” or “Turn on the lights” to see how Watson understands your commands.

You can access the demonstration application at IBM Bluemix®.

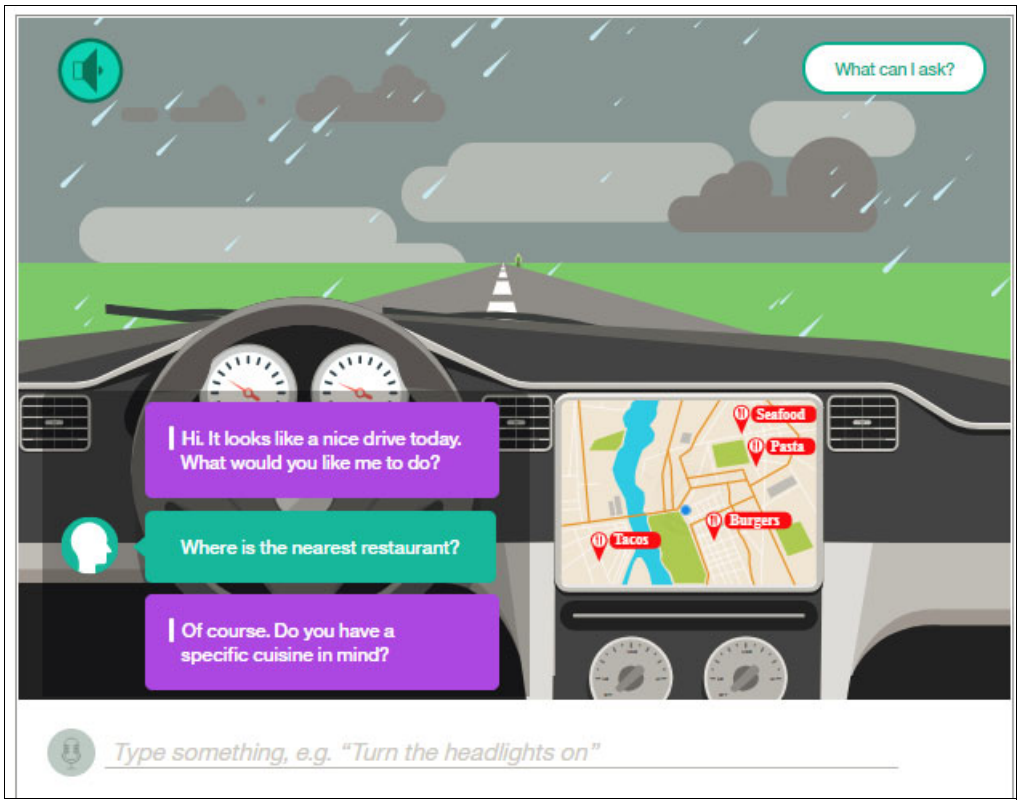

Figure 2-4 Watson Assistant demonstration

Use case

A bank enables its customers to input their text in natural language and uses Watson Assistant to recognize the intent. The application logic uses this intent to determine which back-end core banking service for integration responds to the request. For example, the user inputs “I would like to know my account” balance, and then the application logic integrates with the account balance and returns “Your account balance is 1000” to the user.

2.1.3 Watson Discovery

Watson Discovery enables you to rapidly build AI and cloud-based exploration applications that unlock actionable insights that are hidden in unstructured data. It takes you only a few steps to prepare your unstructured data so that you can create a query that pinpoints the information that you need, and then integrate those insights into your new application or existing solution. You can securely unify structured and unstructured data with pre-enriched content, and use a simplified query language to eliminate the need for manual filtering of results. Common applications include allowing AI search over legal documents, scientific research, or analysis of news.

For an overview of the Watson Discovery service, see this YouTube video.

Demonstration

Figure 2-5 shows a demonstration application that uses the Watson Discovery service. Several news articles are imported into Watson Discovery, and it quickly finds insights in recent news articles.

You can access the demonstration application at IBM Bluemix.

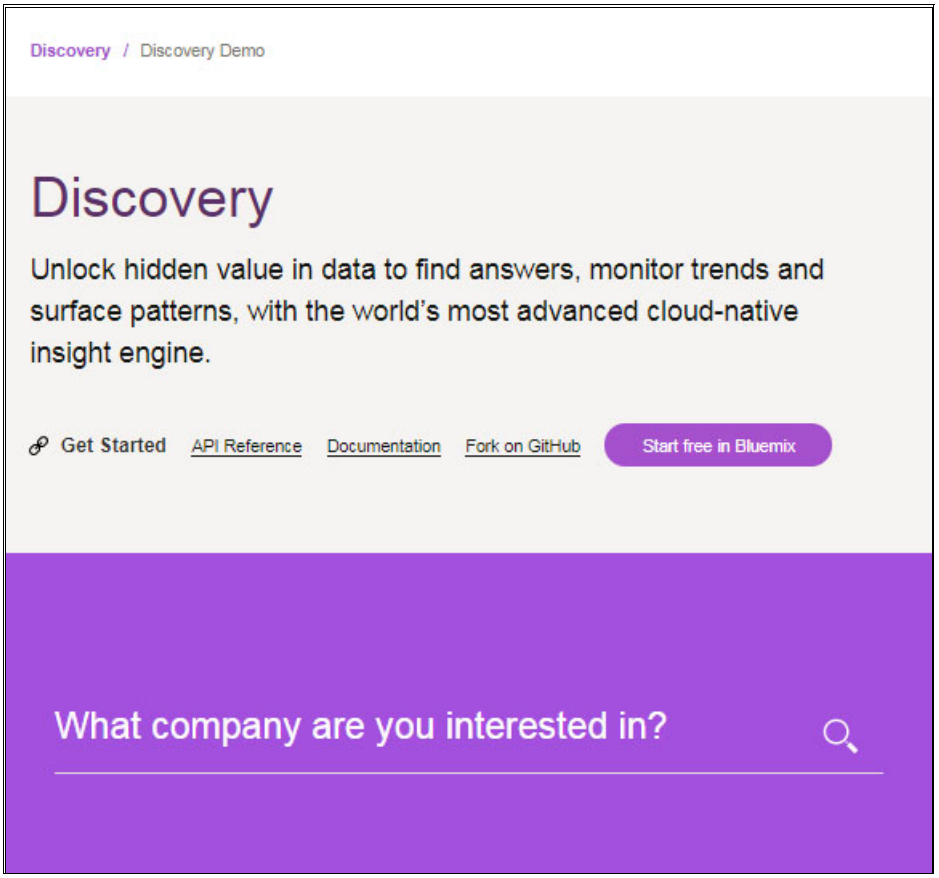

Figure 2-5 Watson Discovery demonstration

Write the name of any company and then press Enter. The demonstration application uses the Watson Discovery service to output the top stories and top entities, as shown in Figure 2-6.

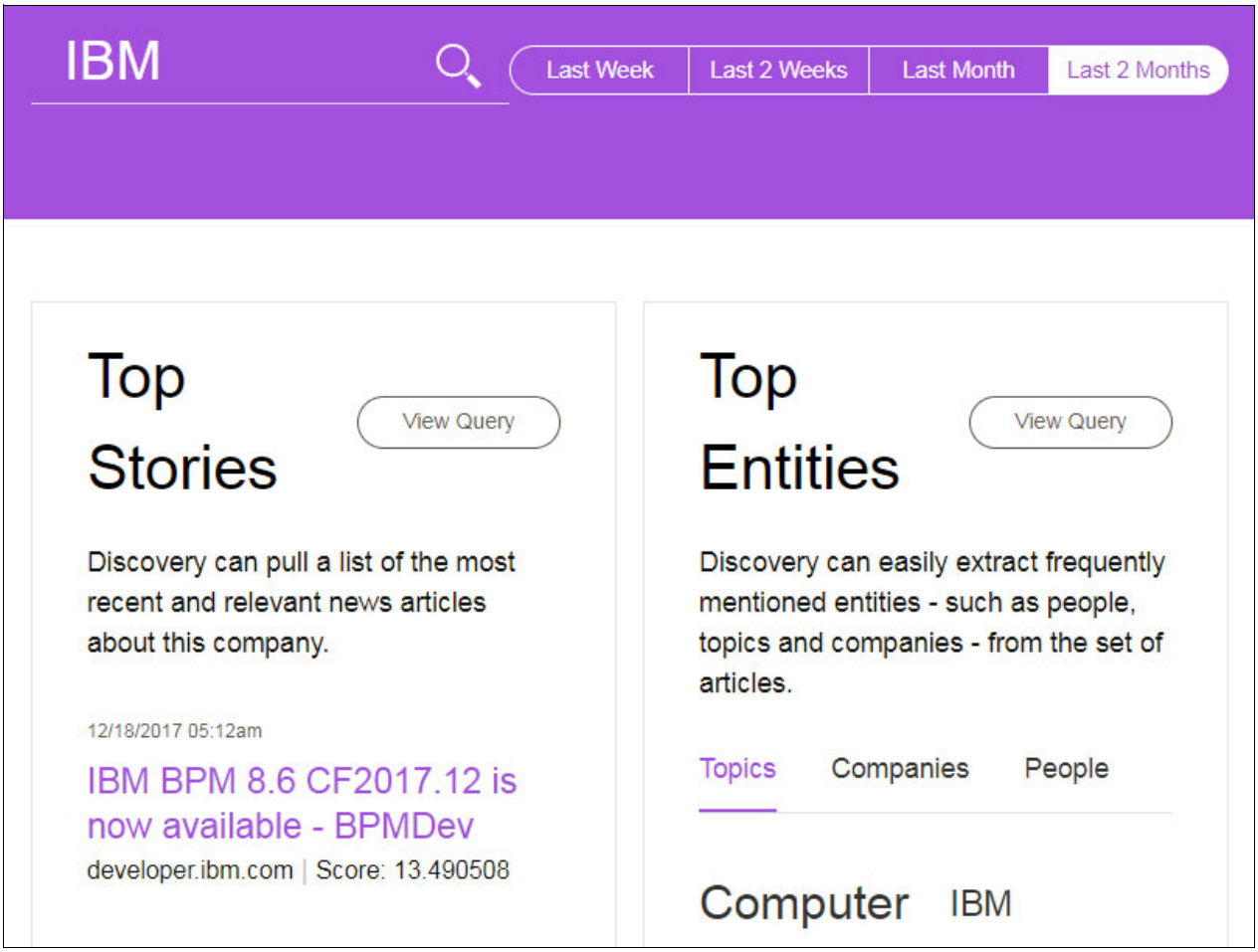

Figure 2-6 Top stories and top entities in the demonstration application

The demonstration application also outputs the sentiment analysis for all the news articles about the company grouped by each source, as shown in Figure 2-7.

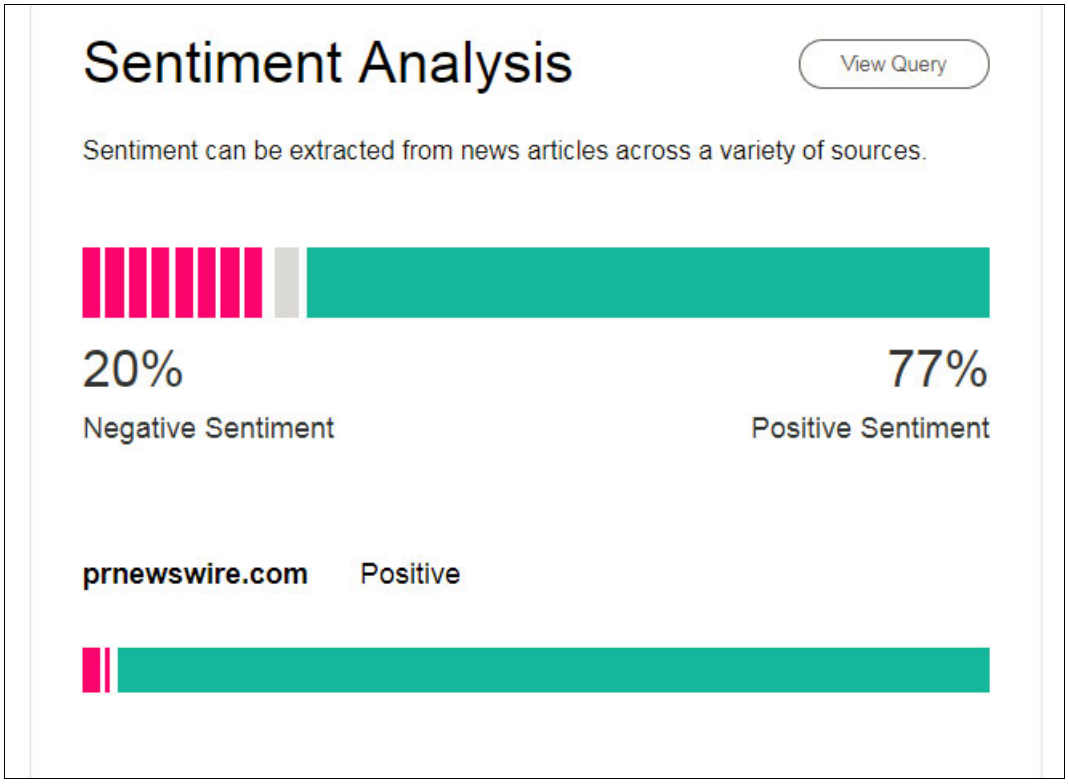

Figure 2-7 Sentiment Analysis

It detects days with an unusually high number of mentions. For example, Figure 2-8 shows that October 17 has a high number of mentions for IBM because of the major US stock indexes.

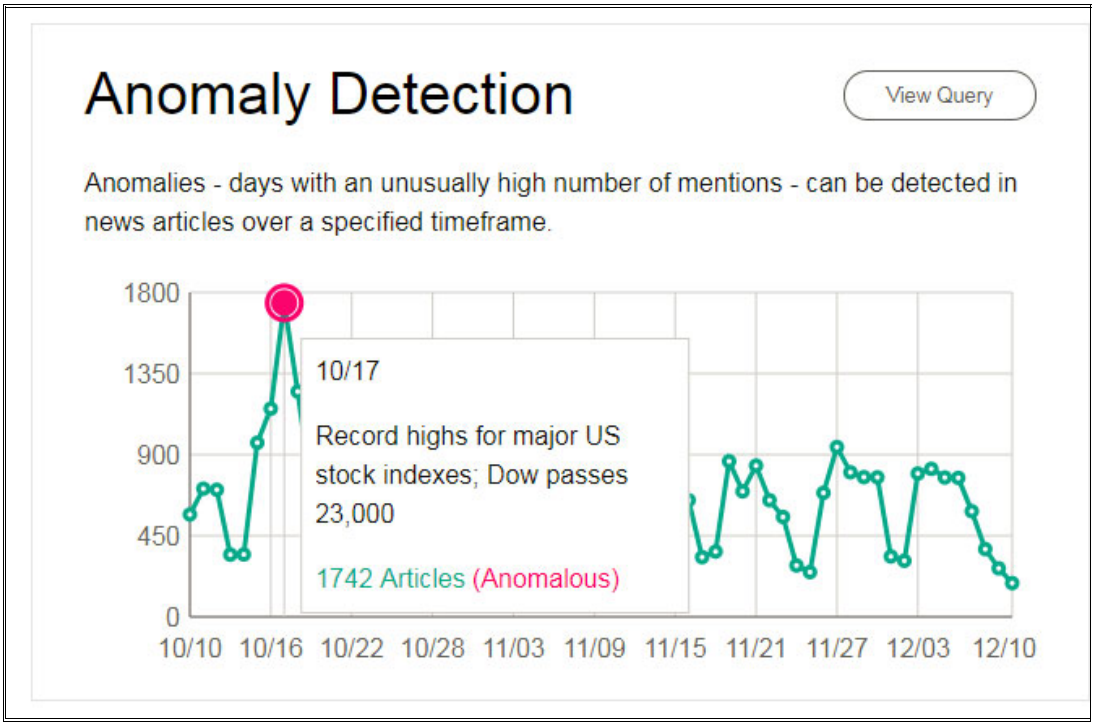

Figure 2-8 Anomaly Detection

Watson Discovery can identify frequently co-mentioned entities and follow trends in sentiment. For example, Figure 2-9 shows that IBM and Microsoft are frequently mentioned together with a positive sentiment.

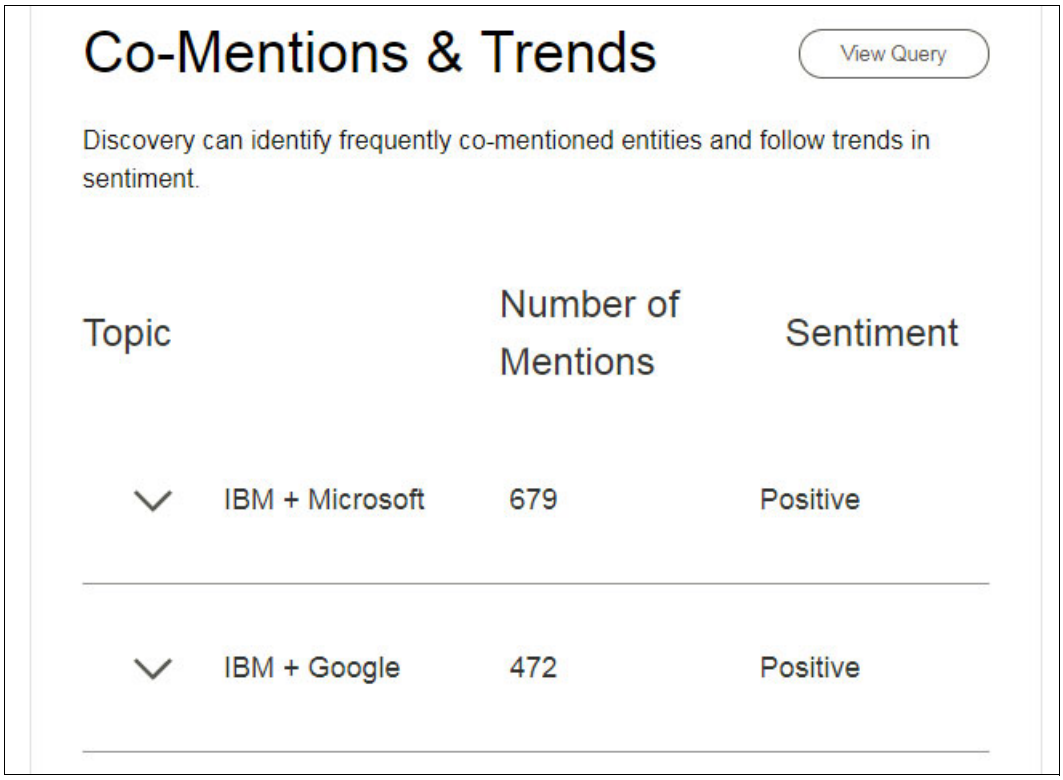

Figure 2-9 Watson Discovery Co-Mentions & Trends

Use case

As a prerequisite for large companies before submitting patents, they should first make sure that no similar ideas of their patents were submitted. To accomplish this task, they can import all their previous patents on to the Watson Discovery service. For new patents, they can use the Watson Discovery service to check whether there is any possible match.

2.1.4 Watson Language Translator

The Watson Language Translator service enables you to programmatically translate text. Common applications include translating text from one language to another, taking news from across the globe and presenting it in your language, publishing content instantly in multiple languages, or communicating with your customers in their own language. For the list of supported languages, see IBM Watson Translator.

For an overview of the Watson Language Translator service, see this YouTube video.

Demonstration

Figure 2-10 shows a demonstration application that uses the Watson Language Translation service.

Note: The demonstration application enables you to view the equivalent REST API call and JavaScript Object Notation (JSON) output. You learn more details about REST and JSON in 2.2.1, “Representational State Transfer Architecture” on page 37 and 2.2.2, “JavaScript Object Notation” on page 39.

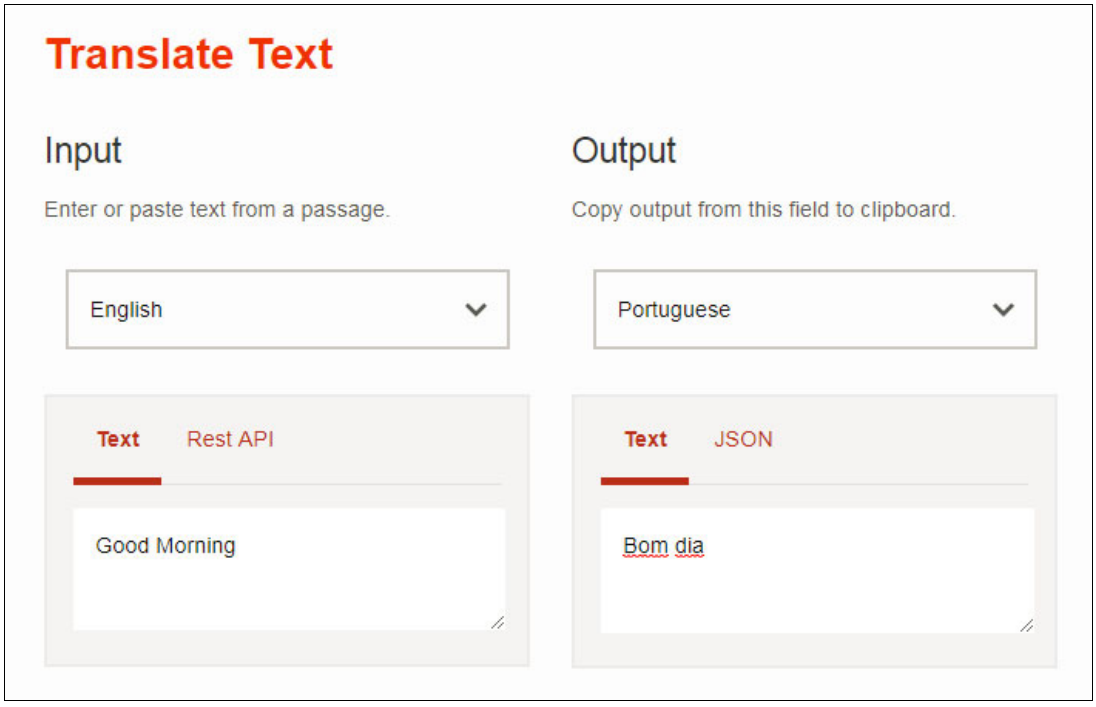

Figure 2-10 Watson Language Translation demonstration application

You can access the demonstration application at IBM Bluemix.

Use case

A multi-national blogging company has authors from across the globe, and must maintain the English version of the blogs. For example, when an Arabic speaking author submits an Arabic blog, it is sent to the Watson Language Translator service for conversion into English, and then an editor reviews the English content and publishes it.

2.1.5 Watson Natural Language Classifier

The Watson Natural Language Classifier service enables you to build applications that understand the language of short texts and make predictions about how to handle them. A classifier learns from your example data and then can return information for texts that it is not trained on. It applies machine learning techniques to return the best matching classes for a sentence or phrase. You create a classifier instance by providing a set of representative sentences and one or more correct classes for each set. After training, the new classifier can accept questions or phrases and return the top matches with a confidence level for each match. Common applications include redirecting a user to a set of features in a web portal depending on the natural language inquiry.

For an overview of the Watson Natural Language Classifier service, see this YouTube video.

Demonstration

Figure 2-11 shows a demonstration application that uses the Natural Language Classifier service. The classifier is trained to determine whether the question is related to temperature or conditions. The output includes the top classification and a confidence score. For example, Watson Natural Language Classifier categories the sample phrase “is it cold?” as temperature with a confidence rate of 100%. This categorization is based on the training that can be done by the Natural Language Classifier service.

You can access the demonstration app at IBM Bluemix.

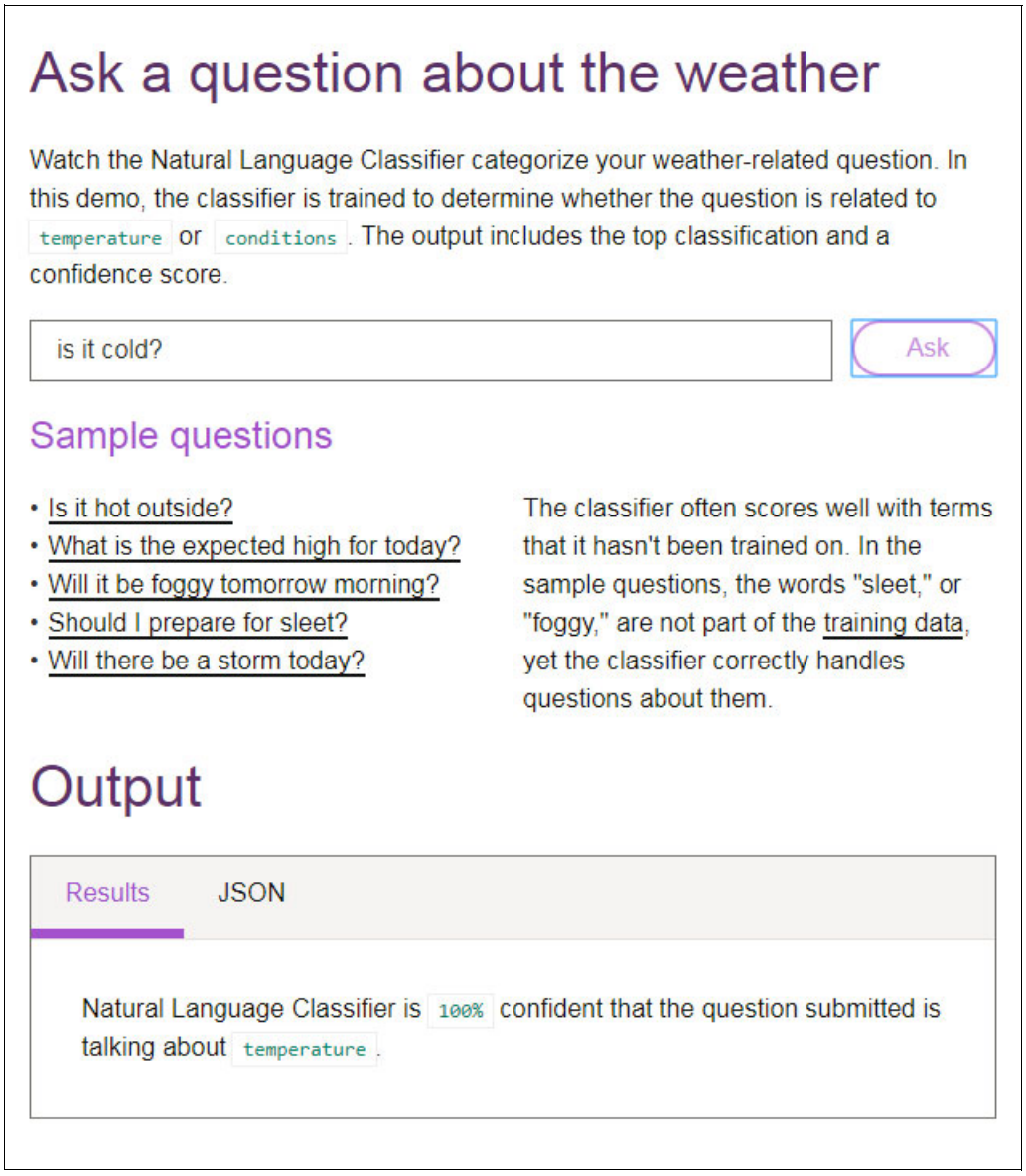

Figure 2-11 Watson Natural Language Classifier demonstration app

Use case

A company has a web portal. On the portal home page, there is a question about what the user is looking for. The page prompts “What do you want to do today?”. For example, the user can input “I want to submit my expenses”. Based on the user intent as determined by Watson Natural Language Classifier, the application logic redirects the user to the correct page, which in this case is the Expenses application.

2.1.6 Watson Natural Language Understanding

The NLU service enables you to build applications that use natural language processing for advanced text analysis. You can analyze text to extract metadata from content such as concepts, entities, keywords, categories, sentiment, emotion, relations, and semantic roles. Common use cases include providing text analysis for customer feedback that helps improve customer satisfaction.

For an overview about the NLU service, see this YouTube video.

Demonstration

Figure 2-12 shows a demonstration application that uses NLU. It takes text or a Uniform Resource Locator (URL), and extracts the sentiment, emotion, important keywords, entities, or concepts, classifies content into a hierarchy, and parses the sentences.

You can access the demonstration app at IBM Bluemix.

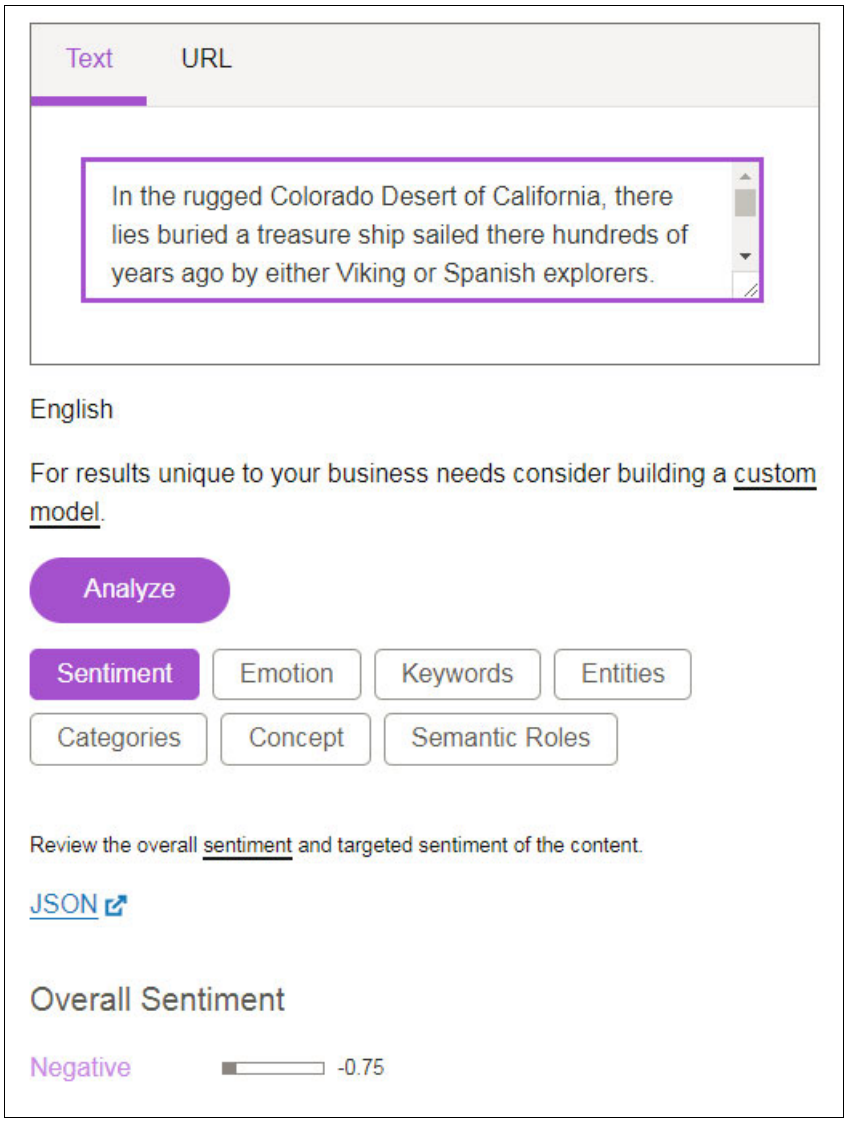

Figure 2-12 Watson NLU demonstration app

Use case

A large company that provides Level 3 support asks their customers for their feedback. They use the keyword feature with sentiment enabled in NLU to determine which parts of the support that the clients are happy about and which parts the clients have issues with. The company uses this information to enhance the level of the services that are provided. This data can also be considered in the yearly performance evaluation of support employees.

2.1.7 Watson Knowledge Studio

The Watson Knowledge Studio enables you to teach Watson the language of your domain. You can create a machine learning model that understands the linguistic nuances, meaning, and relationships that are specific to your industry. You can also create a rules-based model that finds entities in documents based on rules that you define. You can facilitate the task of training Watson with Watson Knowledge Studio to become a subject matter expert (SME) in a given industry or domain.

Common applications include integrating a custom model from Watson Knowledge Studio with the Watson Discovery service to provide custom entity and relations enrichments. This solution gives you the flexibility to apply the Watson Discovery service's document-enhancing capabilities with information that is specific to areas of particular focus, such as industry or scientific discipline. You can also extend NLU with custom models that identify custom entities and relations unique to your domain from the Watson Knowledge Studio.

For an overview of the Watson Knowledge Studio service, see this YouTube video.

Use case

A software support company uses Watson Knowledge Studio to develop a model that understands the language of support requests in their specific domain. The company integrates the model with the NLU service. Their application on Power Systems servers uses the entities that are extracted from NLU on the specific domain that is trained by Watson Knowledge Studio to assign support tickets to the company's engineers.

2.1.8 Watson Personality Insights

The Watson Personality Insights service enables you to build a solution that can predict personality characteristics, needs, and values through written text. You can understand your customers' habits and preferences on an individual level. The service uses linguistic analytics to infer individuals' natural personality characteristics from digital communications, such as email, text messages, tweets, and forum posts. The service infers, from potentially noisy social media, portraits of individuals that reflect their personality characteristics. Common applications include determining individuals' consumption preferences, which indicate their likelihood to prefer various products, services, and activities.

For an overview of the Watson Personality Insights service, see this YouTube video.

Demonstration

Figure 2-13 shows a demonstration application that uses the Watson Personality Insights service. It enables you to analyze the personality of individuals.

You can access the demonstration application at IBM Bluemix.

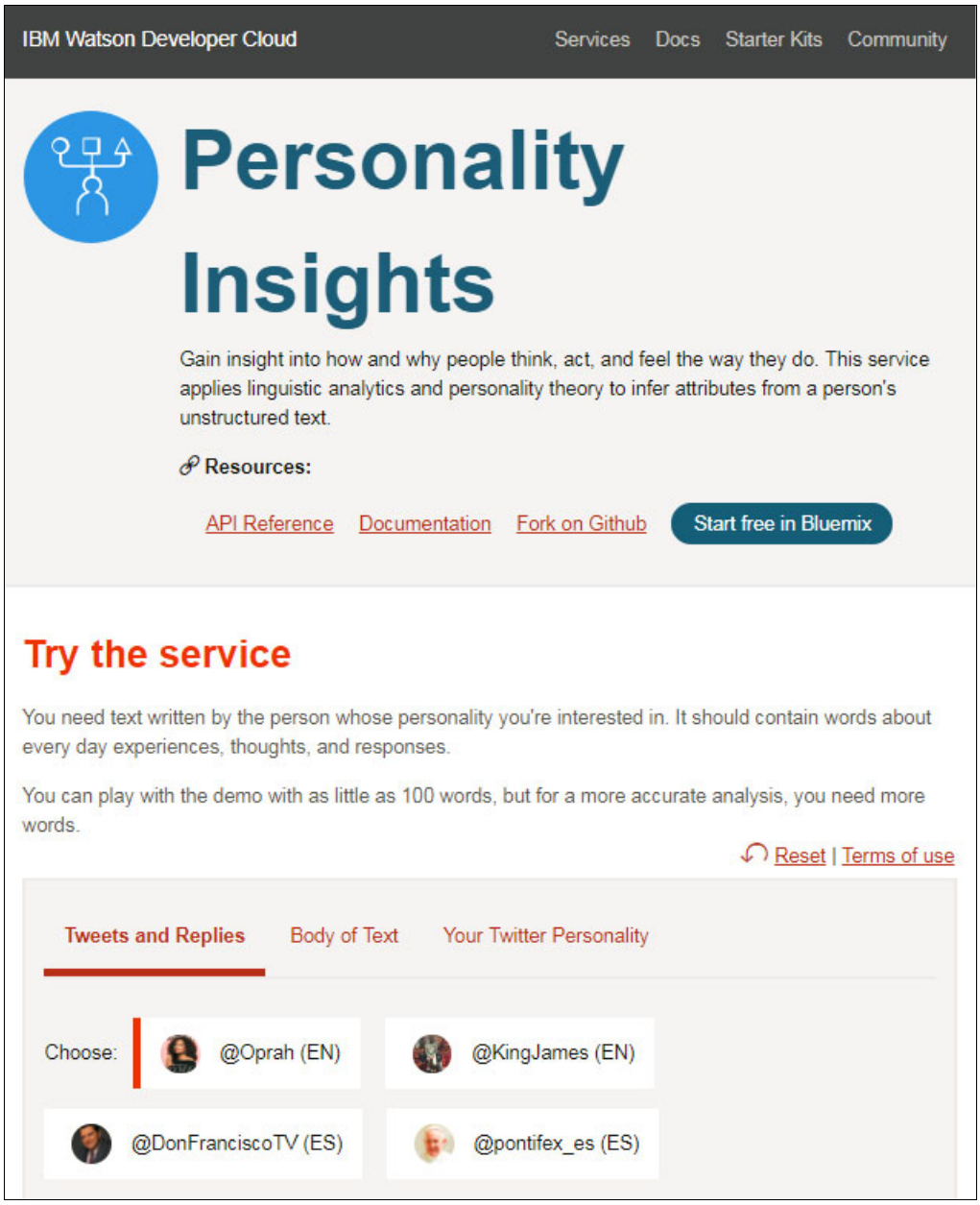

Figure 2-13 Watson Personality Insights demonstration app

Figure 2-14 shows a sample analysis for Tweets and Replies.

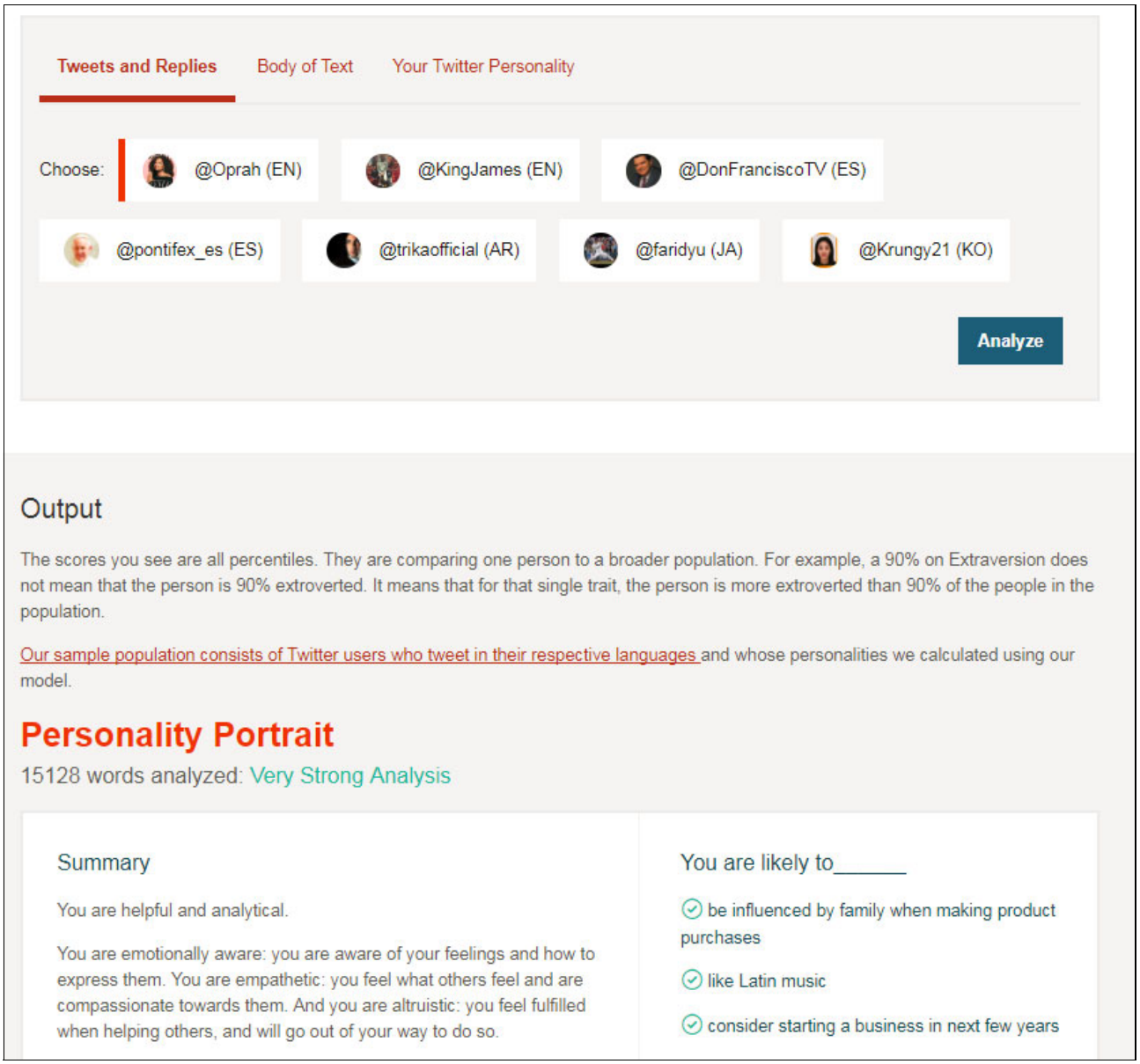

Figure 2-14 Personality Analysis for Tweets and Replies

Figure 2-15 shows a sample analysis for a text.

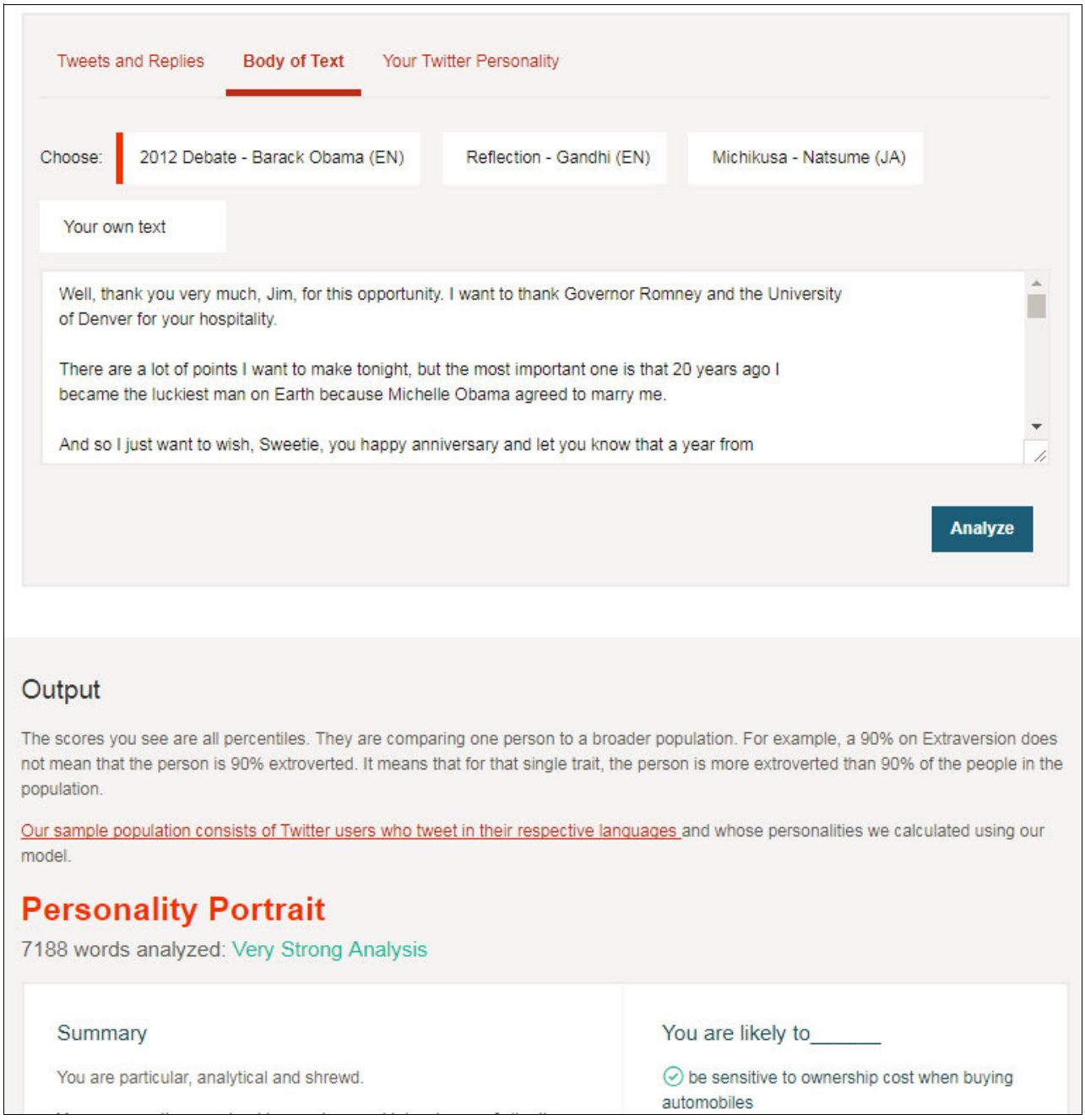

Figure 2-15 Watson Personality Insights for Body of Text

Use case

An outdoor activities company analyzes its customers tweets with Watson Personality Insights and uses this analysis to group their customers into different market segments. The company then provides a tailored marketing campaign for each segment.

2.1.9 Watson Speech to Text

The Watson Speech to Text service enables you to build solutions that convert audio and voice into written text for quick understanding of content. You can use it to add speech transcription capabilities to your applications. To transcribe the human voice accurately, the service uses machine intelligence to combine information about grammar and language structure with knowledge of the composition of the audio signal. The service continuously returns and retroactively updates a transcription as more speech is heard. Common applications include voice control of applications, embedded devices, and vehicle accessories, transcribing meetings and conference calls, and dictating email messages and notes.

For more information about the languages that are support by the service, see IBM Bluemix.

For an overview of the Watson Speech to Text service, see this YouTube video.

Demonstration

Figure 2-16 shows a demonstration application that uses the Watson Speech to Text service. You record audio, and it returns the equivalent text.

You can access the demonstration application at IBM Bluemix.

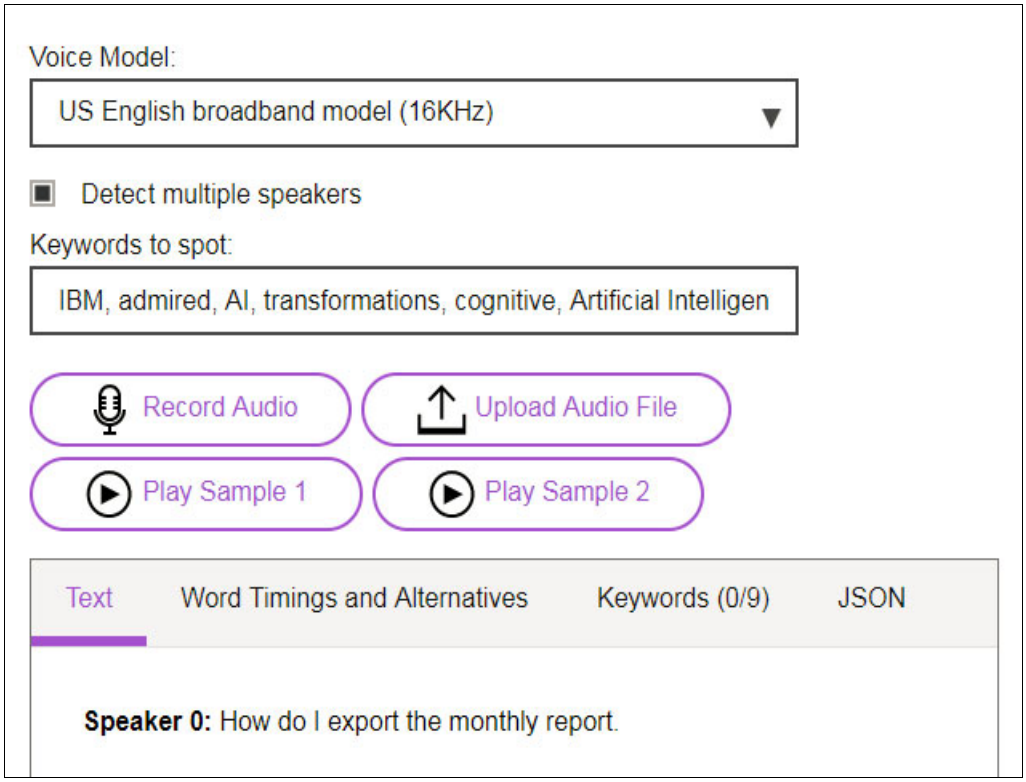

Figure 2-16 Watson Speech to Text demonstration application

Speaker labels let you identify which individuals spoke which words in a multi-participant exchange. You can use the information to develop a person-by-person transcript of an audio stream, such as contact to a call center, or to animate an exchange with a conversational robot or avatar. Figure 2-17 shows a sample conversation from the demonstration application that uses speaker labels.

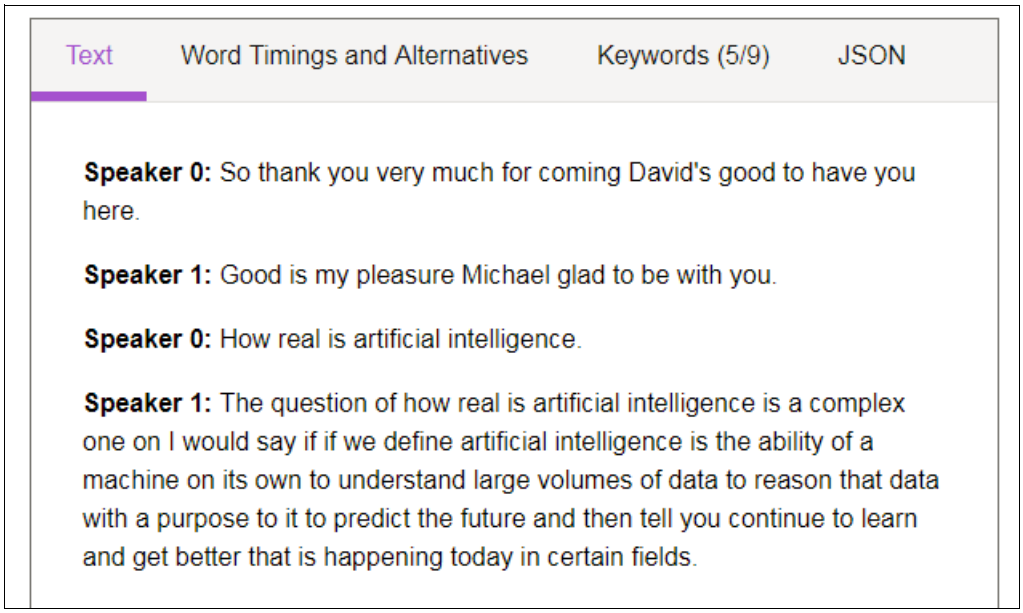

Figure 2-17 Speakers labels on the demonstration application

Use case

A multi-national call center company has recordings for their agents with their customers that they gathered for quality purposes. They use Watson Speech to Text to analyze the text by combining the results with other Watson services.

2.1.10 Watson Text to Speech

The Watson Text to Speech service enables you to convert written text into natural-sounding audio in various languages and voices. It processes text and natural language to generate synthesized audio output complete with appropriate cadence and intonation. Common applications include a speech interface to a hotel concierge to enable the service to respond in a spoken natural language to customer's inquires.

The service is available with the voices and languages that are listed at IBM Bluemix.

Demonstration

Figure 2-18 shows a demonstration application that uses the Watson Text to Speech service. You can input the text, and it outputs the speech in several voices.

You can access the demonstration application at IBM Bluemix.

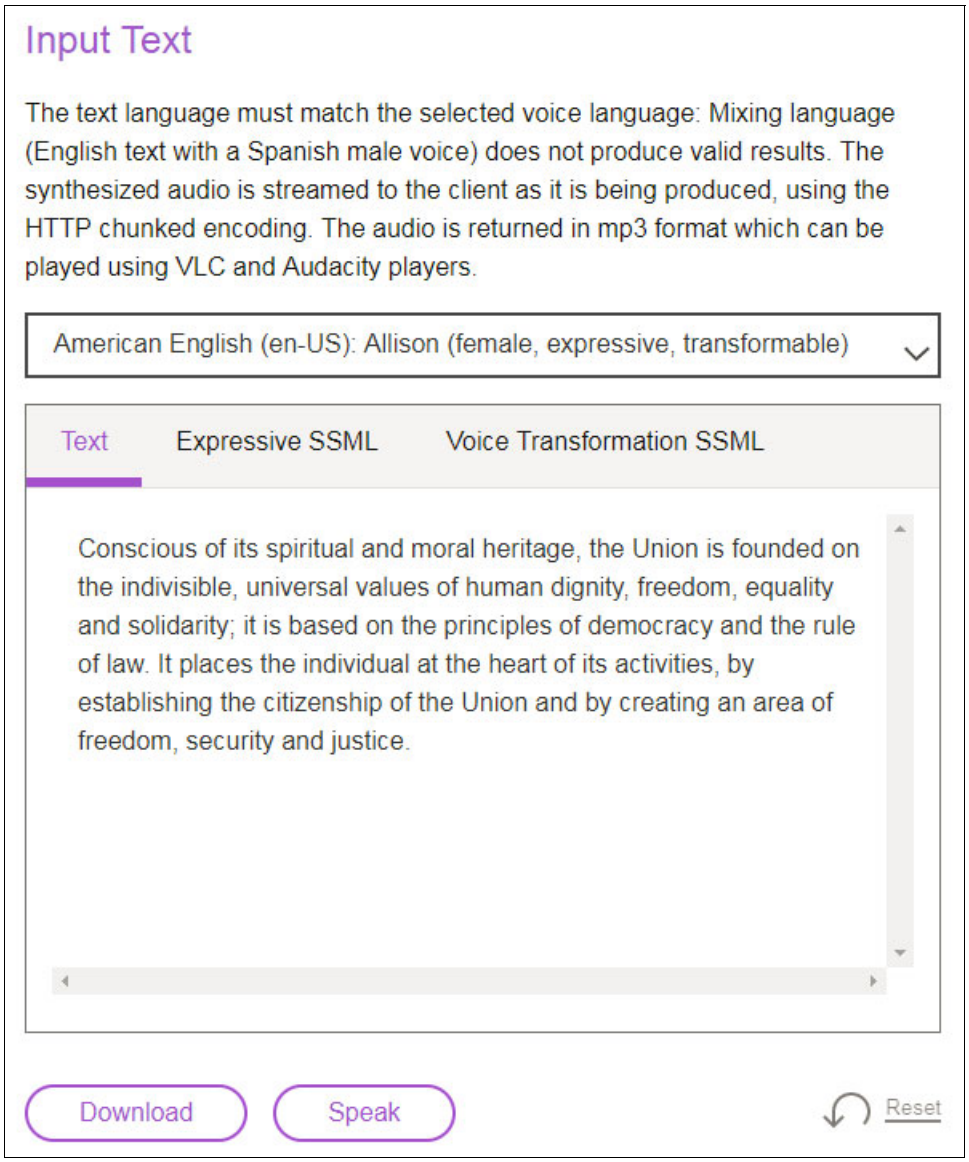

Figure 2-18 Watson Text to Speech demonstration application

By default, the Watson Text to Speech service synthesizes text in a neutral declarative style. The service uses the `<express-as>` element in Speech Synthesis Markup Language (SSML), as shown in Figure 2-19. SSML is an XML-based markup language that provides annotations of text for speech-synthesis applications. The `<express-as>` element lets you indicate expressiveness by converting text to synthesized speech in the following speaking styles:

- GoodNews to express a positive message.
- Apology to express a message of regret.
- Uncertainty to convey an uncertain, interrogative message.

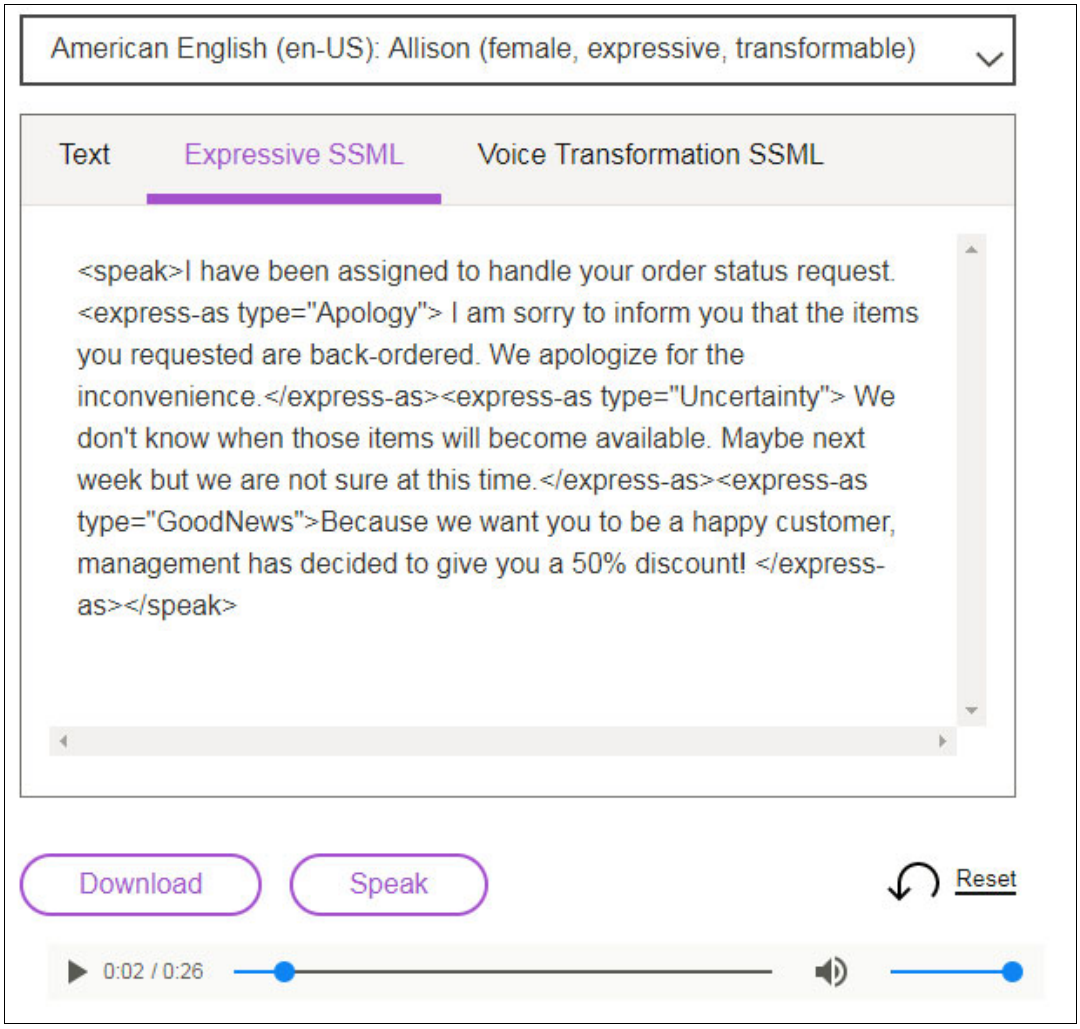

Figure 2-19 Watson Text to Speech: Expressive SSML

To expand the range of possible voices, the service extends SSML with a <voice-transformation> element that lets you realize different virtual voices by controlling aspects of a default voice, as shown in Figure 2-20.

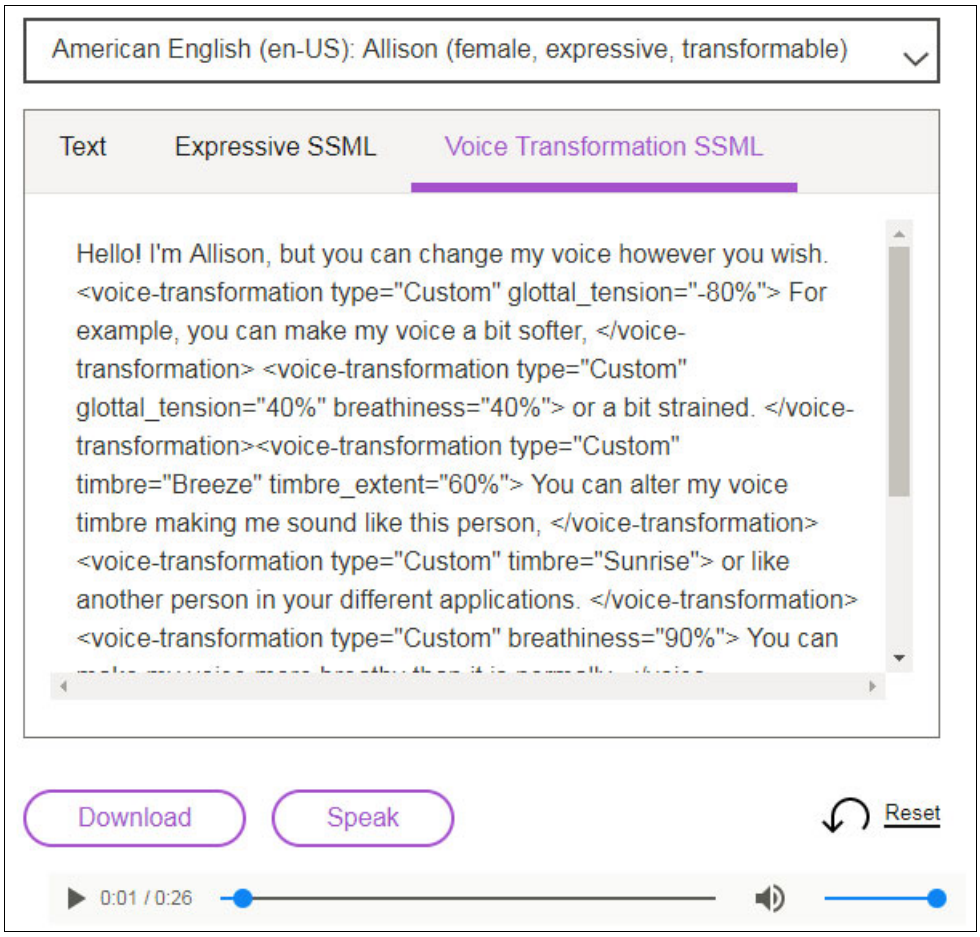

Figure 2-20 Voice Transformation SSML

Use case

A multi-national hotel concierge provides services for guests, which include reserving tables at exclusive restaurants or purchasing theater tickets. For example, the guest says in their own language “I would like to reserve a table for two at a popular restaurant downtown”. The application uses Watson Text to Speech to respond to the guest in a natural voice in the same language, “Table is reserved, Enjoy your meal”.

2.1.11 Watson Tone Analyzer

The Watson Tone Analyzer service enables you to build a solution that understands emotions and communication style in text. It uses linguistic analysis to detect emotional and language tones. The service can analyze tone at both the document and sentence levels. It detects types of tones, including emotion (anger, fear, joy, and sadness), and language styles (analytical, confident, and tentative) from text. You can use the service to understand how your written communications are perceived and then to improve the tone of your communications. Common applications include using the service to detect the tone of your customers' communications and to respond to each customer appropriately, or to understand and improve your customer conversations in general.

For an overview of the Watson Tone Analyzer service, see the following YouTube video.

Demonstration

Figure 2-21 shows a demonstration application that uses the Watson Tone Analyzer service. It detects joy, fear, sadness, anger, analytical, confident, and tentative tones that are found in tweets, online reviews, email messages, product reviews in French, or your own text.

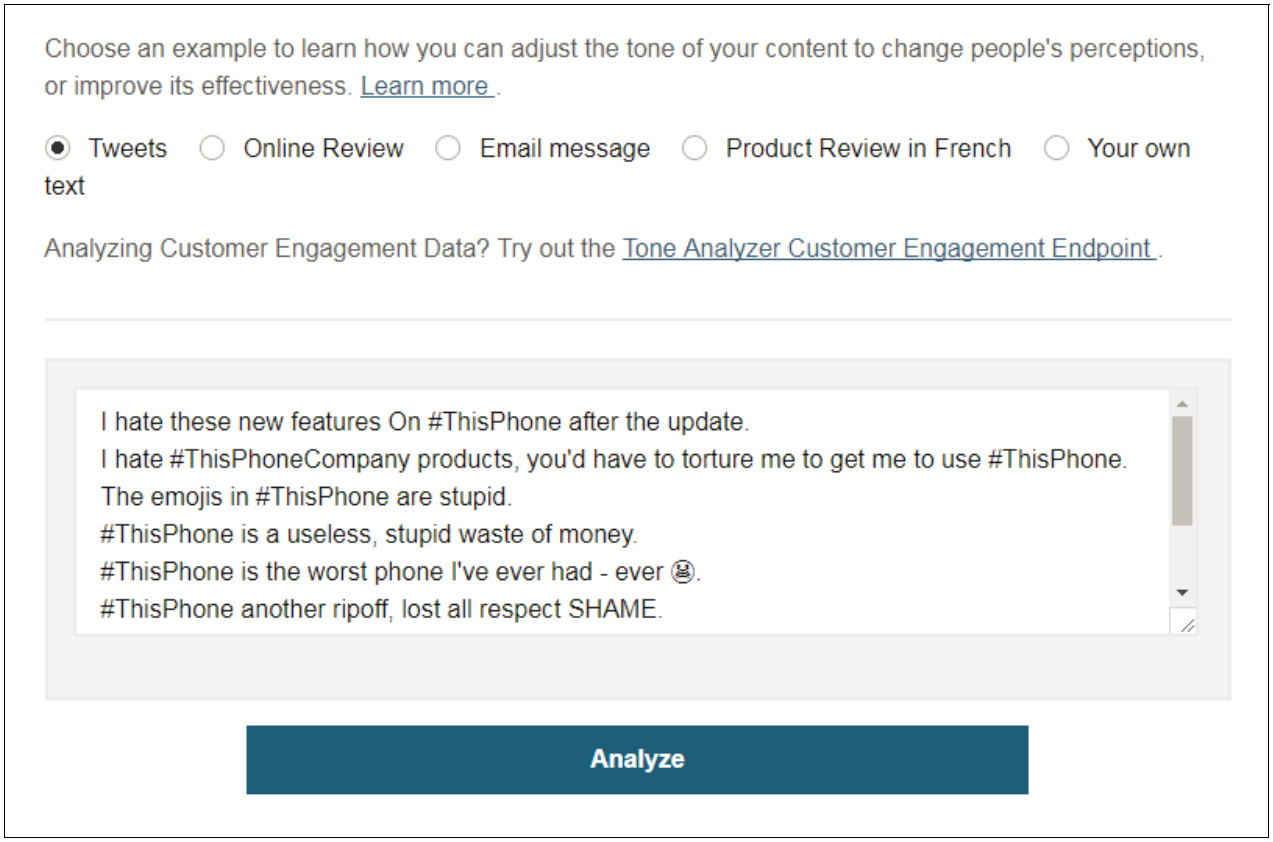

Figure 2-21 Watson Tone Analyzer demonstration application

You can access the demonstration application at IBM Bluemix.

The Watson Tone Analyzer Service analyzes text at the document level and the sentence level. Use the document level analysis to get a sense of the overall tone of the document, and use the sentence level analysis to identify specific areas of your content where tones are the strongest. Figure 2-22 shows an example of document-level and sentence-level analysis.

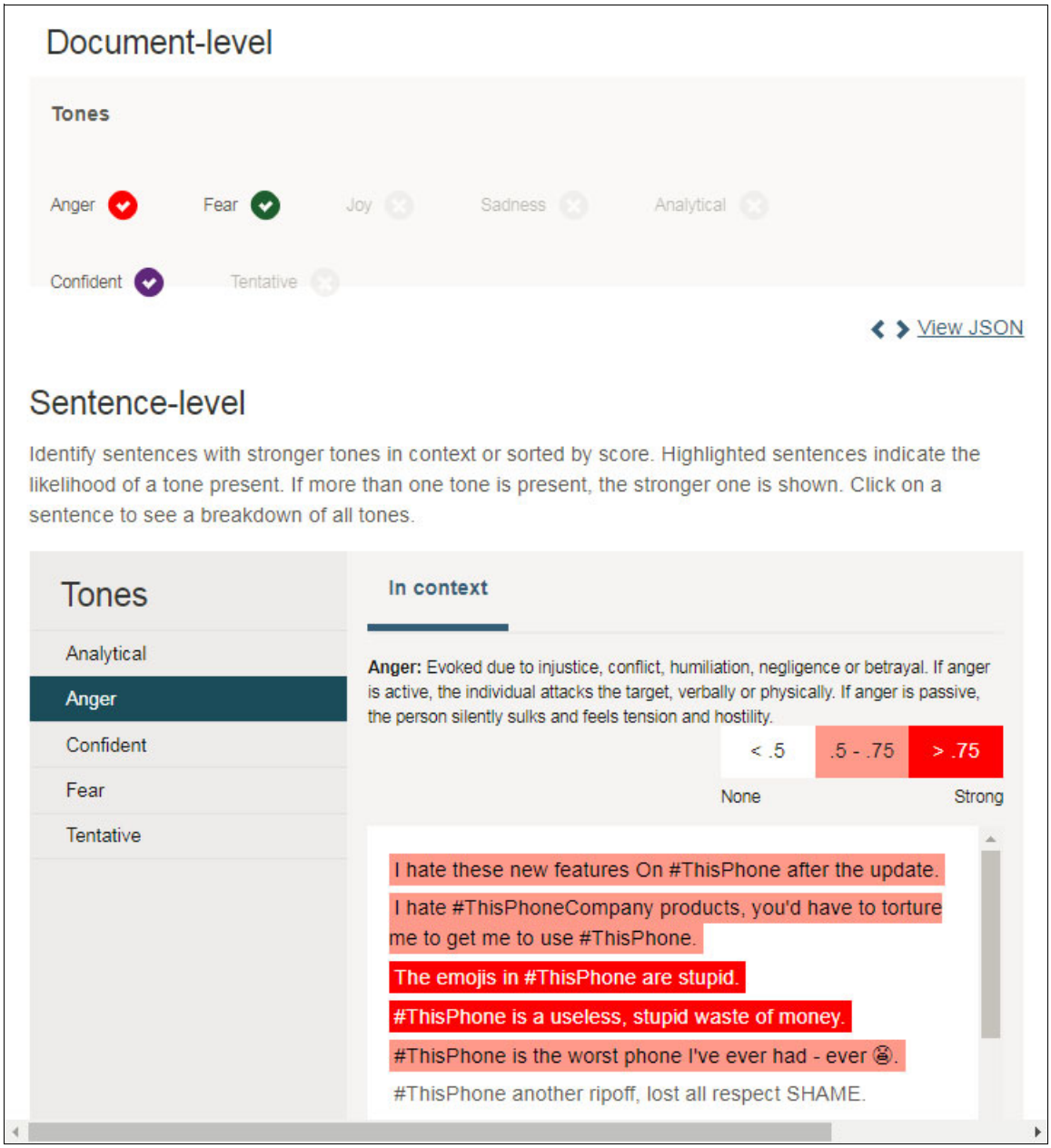

Figure 2-22 Watson Tone Analyzer output

Use case

A customer help desk has a chatbot that interacts with customers in natural language. It uses Watson Tone Analyzer to detect the tone of what the customers write. When the tone starts to become negative feelings such as anger, it switches to a human agent.

2.1.12 Watson Visual Recognition

The Watson Visual Recognition service enables you to build a solution that quickly and accurately tags, classifies, and trains visual content by using machine learning. It uses deep learning algorithms to analyze images for scenes, objects, faces, and other content. The response includes keywords that provide information about the content. There is a set of built-in models that provide highly accurate results without training, and you can also train custom models to create specialized classes. Common applications include detecting diseases on plants, intruder detection, and detecting that workers are wearing the approved safety clothing.

For an overview of the Watson Visual Recognition service, see this YouTube video.

Demonstration

Figure 2-23 shows a demonstration application that uses the Watson Visual Recognition service. Select any photo, and then the service analyzes the contents of the photo in forms of classes and the degree of confidence for each class.

You can access the demonstration application at IBM Bluemix.

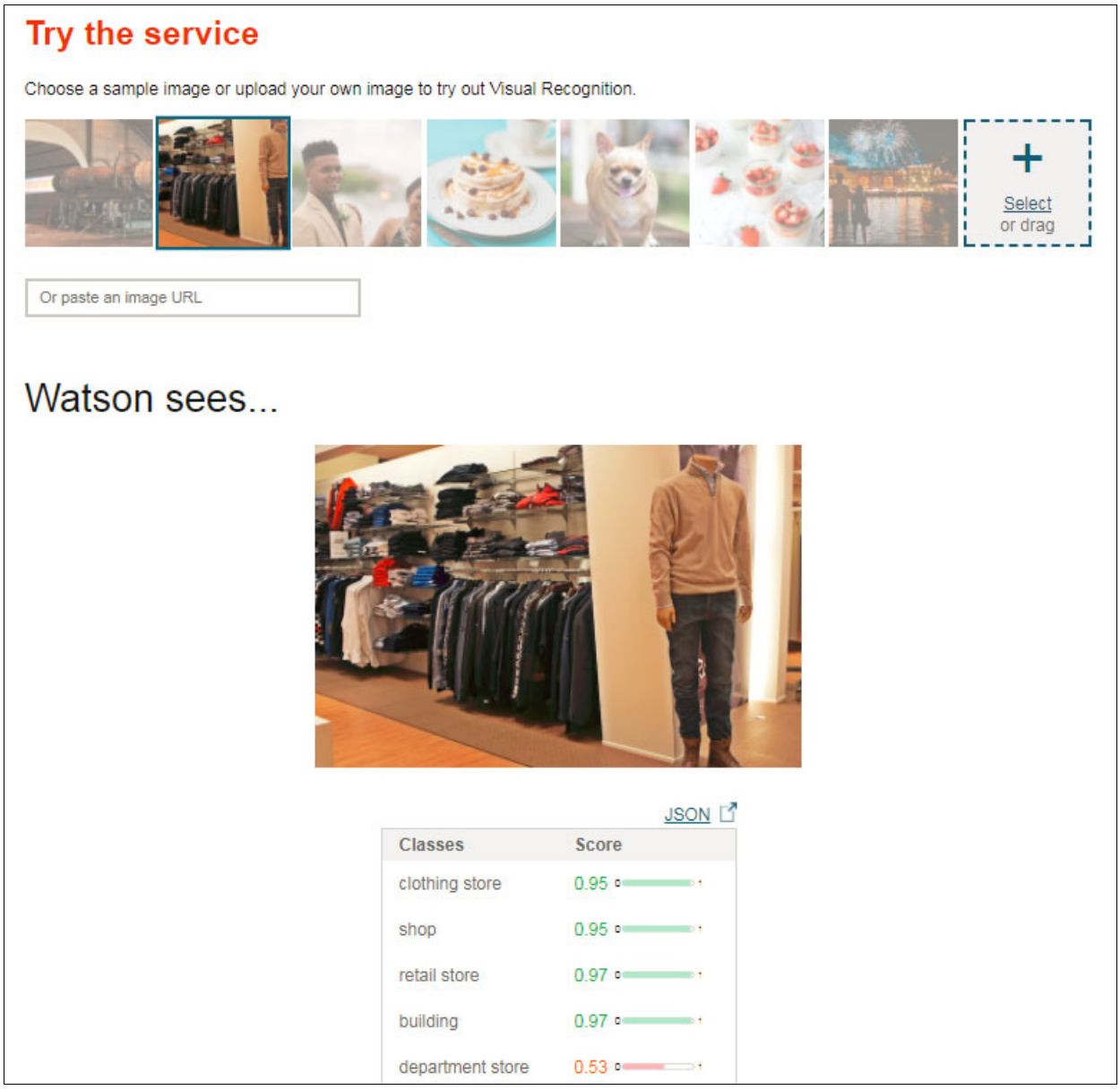

Figure 2-23 Watson Visual Recognition application demonstration

Click **Train** to train your classifier into different classes. Figure 2-24 shows a demonstration that creates a temporary classifier for Dog Breeds. It shows training images for four classes: Golden Retriever, Husky, Dalmation, and Beagle. It also shows training the classifier against a negative class by using pictures of animals other than dogs.

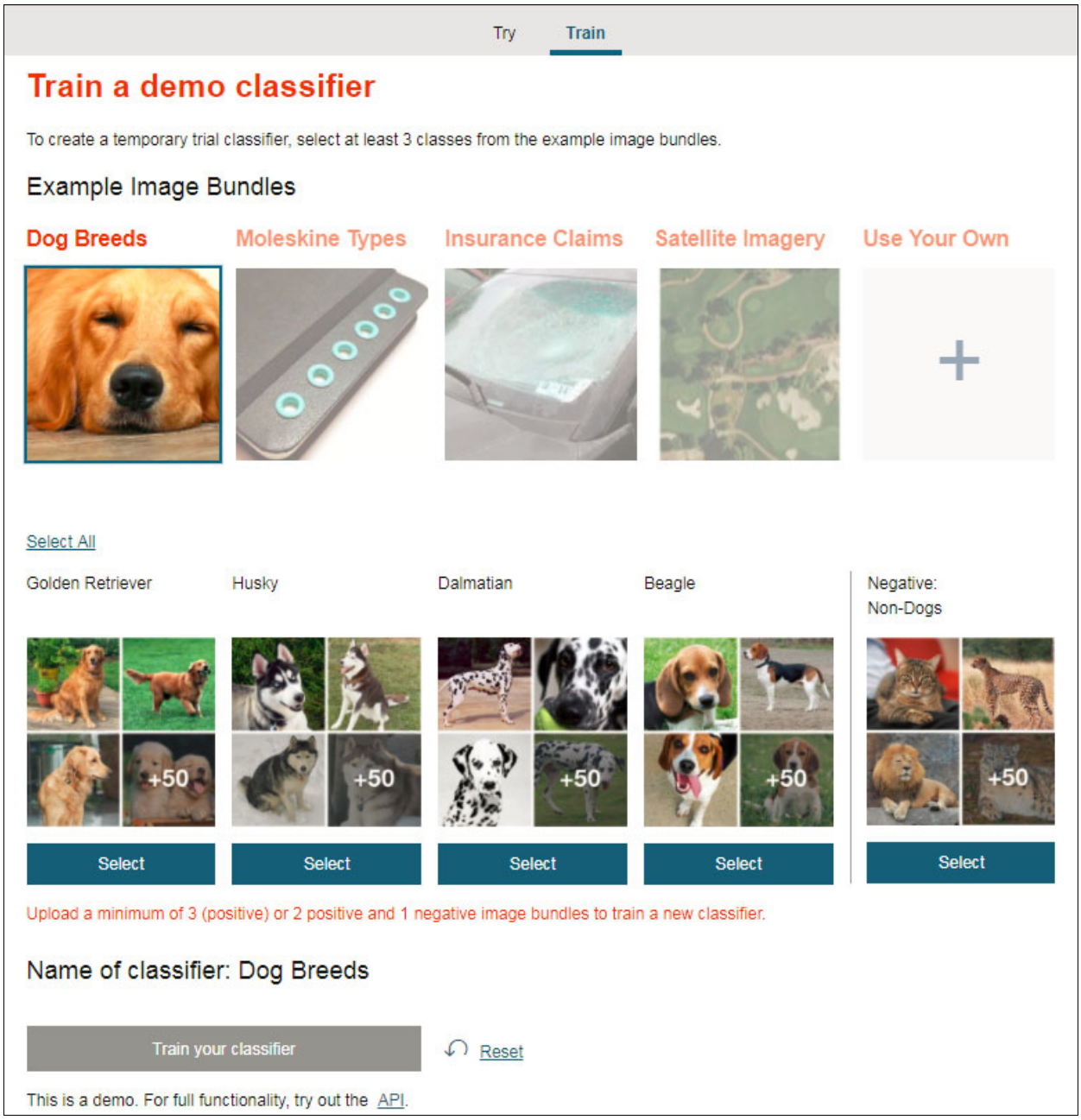

Figure 2-24 Training Watson Visual Recognition service

Use case

An agricultural company that grows crops for commercial uses Watson Visual Recognition to detect disease in the plants and determines which treatment to apply. Watson Visual Recognition is trained on the different diseases by providing several images that represent each disease and other images that show disease-free plants.

2.2 Representational State Transfer architecture

In 2.1, “Introducing IBM Watson” on page 10, you learned about the different Watson services that are available on IBM Cloud and that all Watson services are exposed through REST APIs.

In this section, you learn about REST APIs, and how to call Watson services through REST APIs.

2.2.1 Representational State Transfer Architecture

REST is an architecture style for building resources on the web. Examples of resources include HTML documents, images, and script file, as shown in Figure 2-25.

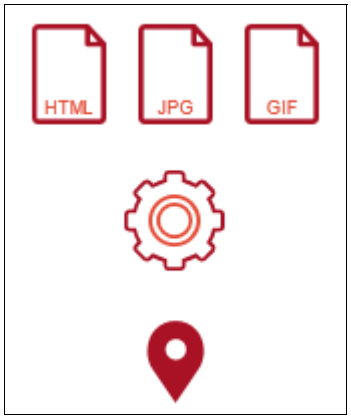

Figure 2-25 Example of resources that can be accessed by REST

To retrieve or update a resource, perform an action through HTTP methods. REST uses a Uniform Resource Identifier (URI) to describe the network location of the resource and to identify which resource to retrieve or update.

REST provides the following HTTP methods:

- **GET**
- **POST**
- **DELETE**
- **PUT**
- **OPTIONS**
- **HEAD**
- **TRACE**
- **CONNECT**

The **GET** method is used to retrieve information from the server. When you use your browser to go to any URI, you use the **GET** method to get the HTML of that website. The query string that contains the parameters that are needed for the request are sent to the URL by placing a question mark (?) at the end of the URI, and then writing the parameters.

Each parameter is represented as a name-value pair. The parameters are separated by “&”. The URI for a **GET** request can be formatted as shown in the following examples:

- `http://example.com/personDetail?firstName=Ahmed&age=28`
- `http://example.com/personDetail/Ahmed/28`

The **POST** method is used to post data to the server. In this case, the parameters are posted in the body of the request, not in the URI.

The **DELETE** method is used to delete a resource from the server.

In a more general sense, web resources represent a source of information. For example, HTML documents define the structure of a web page. Cascading stylesheet (CSS) documents define the presentation of a web page, and image files provide a visual representation of information. With REST services, you treat server applications as web resources.

A REST service is now an entry point to an application on the server. It provides information from the server application. To call a REST service, use HTTP method verbs, such as **GET**, **PUT**, and **POST**. To specify which REST service to call, use a URI to describe the location of the resource on the server.

Calling Representational State Transfer APIs from Power Systems servers

Figure 2-26 shows a sample application that is developed in the Power platform that calls the Watson Translation service on IBM Cloud through REST APIs.

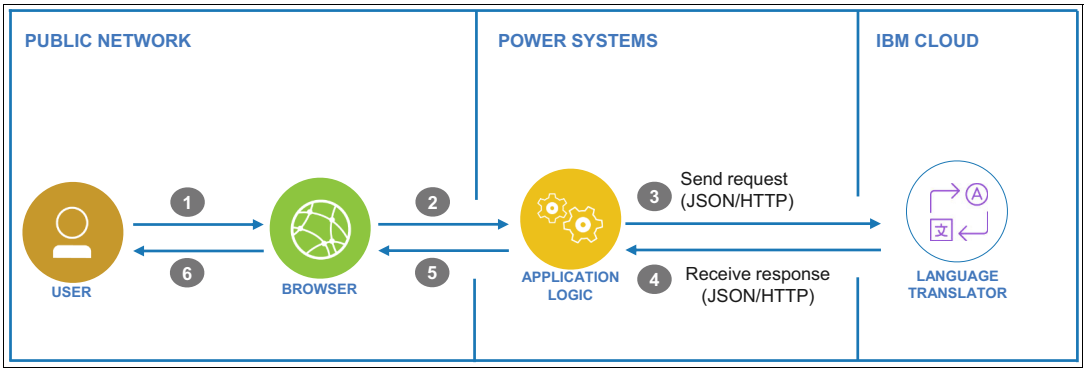

Figure 2-26 Sample application on the Power platform calling Watson services

The following steps explain the sequence of interactions between the components that are shown in Figure 2-26:

1. In a web browser, suppose that the user inputs “Good Morning” and wants to translate it from English to German.
2. The request is passed from the web browser to the application that is deployed on IBM Power Systems.
3. The application logic on Power System performs the following steps:
   a. It formats the request in JSON, as shown in Example 2-1. The JSON request contains two fields, `model_id` and `text`. `model_id`, which show that you want to translate from English to German, and the text shows the text that you want to translate.

Example 2-1 JSON format of the request

```
{
  "model_id": "en-de",
  "text": "Good Morning\n"
}
```

b. It sends a REST request as an HTTP **POST** in JSON format to the Watson Language Translator service.

4. The Watson Language Translator service returns a REST response as a JSON format to the application. Example 2-2 shows the JSON response that is returned from the Watson Language Translator service. The response contains the `translation`, `word_count`, and `character_count`.

Example 2-2 JSON response that is returned from the Watson Language Translator service

```
{
  "translations": [
    {
      "translation": "Guten Morgen"
    }
  ],
  "word_count": 2,
  "character_count": 12
}
```

5. The application that is deployed on the Power Systems server parses the JSON response that is received from the Watson Language Translator service and sends a response (“Guten Morgen”) to the web browser.
6. The user sees the response on the web browser.

Note: Watson has its services available as REST APIs. The application on the Power Systems server can be developed by using Node.js, Report Program Generator (RPG), Java, Python, or any other programming language that provides HTTP or REST clients. It calls these REST APIs by using one of the HTTP methods. The request and the response can be JSON or XML over the HTTP protocol.

2.2.2 JavaScript Object Notation

JSON is a text format for structured data. Its syntax is derived from the object literals of JavaScript, according to the ECMA-262 ECMAScript Language Standard, which is a scripting language standard.

JSON is a platform-neutral and language-neutral data format. The main design goal of JSON is to provide a minimal, portable, and textual data interchange format.

JSON is not a markup language. Unlike XML, it does not use descriptive tags to encapsulate its data. For example, XML is a markup language because it uses tags, such as `<title></title>`, to declare the title of the page.

JSON is built on two structures: A collection of name-value pairs that are known as *objects*, and a list of values known as *arrays*.

JSON features the following data types:

- A *string* is a sequence of zero or more Unicode characters, such as `"Hello world!\n"`.
- A *number* includes an integer part and a fraction. Numbers can be prefixed by a positive or negative sign, and can also include an exponent, such as -12.5435.

- There are two data types to hold a group of values:
  - An *object*: An unordered collection of zero or more name-value pairs. Objects are denoted by curly brackets such as `"person":{ "name":"John","preferredColor":"Blue"}`.
  - An *array*: An ordered sequence of zero or more values. Use square brackets to denote arrays, such as `["a",432,"c"]`.
- A *Boolean* is a literal value of `true` or `false`.
- The keyword `null` represents a null value.

JSON values must be an object, array, number, or string, or one of the three literal names (false, true, or null).

2.3 Exploring Watson services with Watson API Explorer

In this section, you explore some Watson services with Watson API Explorer to become familiar with the Watson services.

Watson API Explorer, which is shown in Figure 2-27, is a public portal that contains documentation for the different Watson REST APIs. It enables the user to test these REST APIs from the portal. You call the Watson API Explorer, which acts as a proxy to the real services, in much the same way that any application that you create acts as a proxy to the services.

You can call these REST services from your Power Systems server by using the Watson service endpoint and parameters.

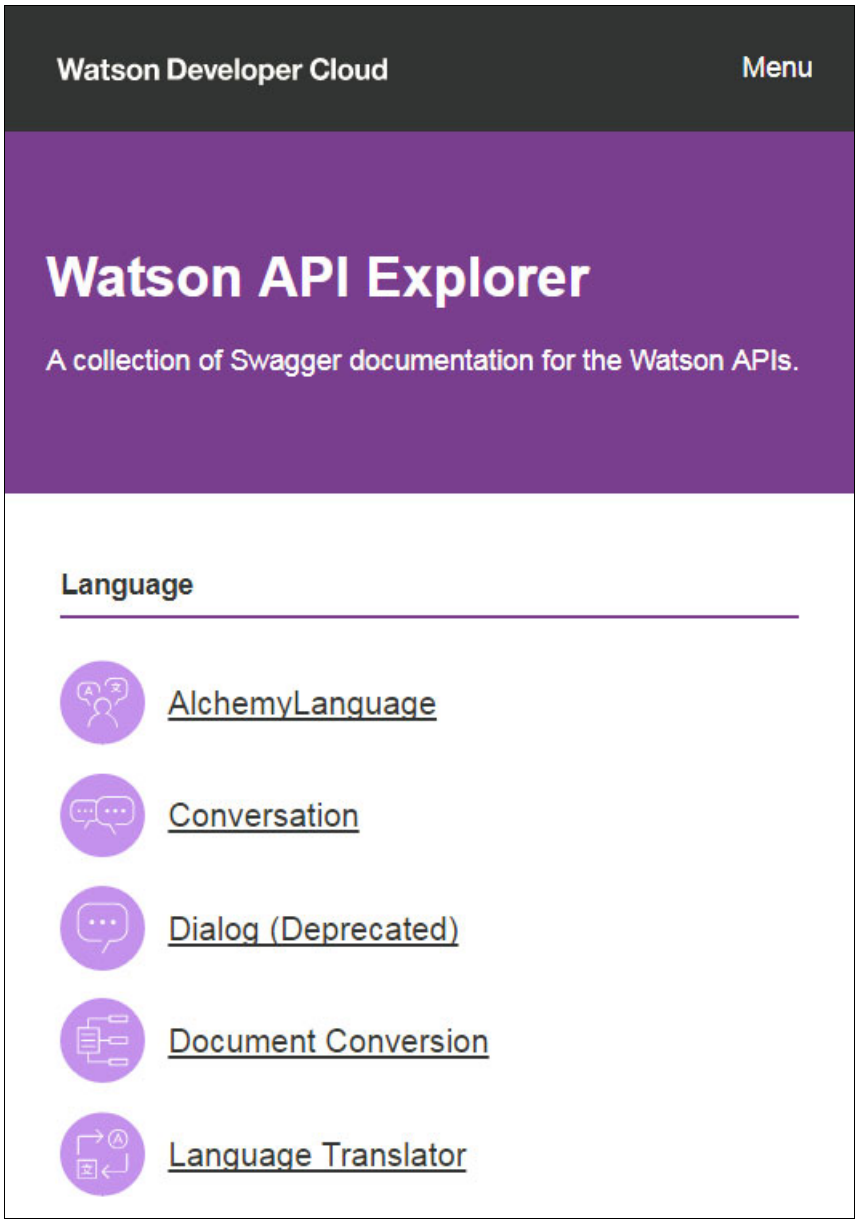

Figure 2-27 Watson API Explorer

Watson API Explorer is based on Swagger, which is used to document REST APIs.

You can access Watson API Explorer at IBM Bluemix.

2.3.1 Watson Natural Language Understanding

In this section, you are going to use NLU.

Complete the following steps:

1. Access NLU on Watson API Explorer.
2. Click **GET /v1/analyze**, as shown in Figure 2-28. You use the `analyze` REST API in the NLU service to extract the author of an article.

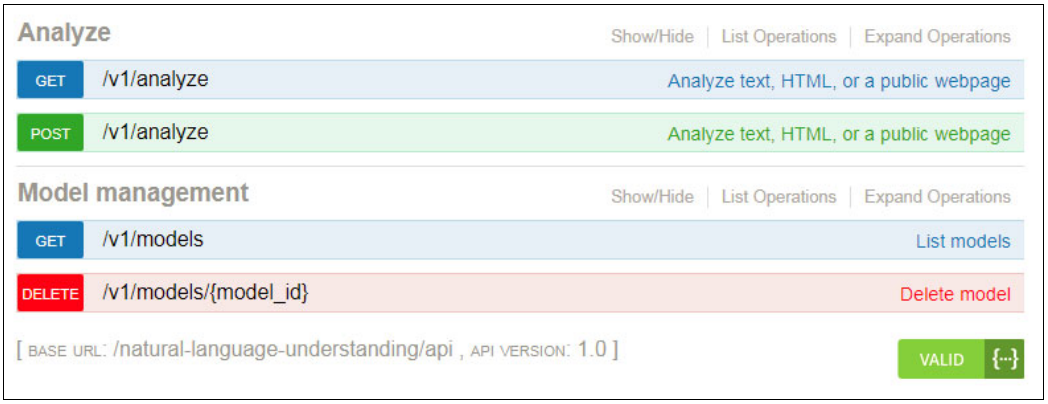

Figure 2-28 Watson API Explorer: NLU

3. Perform the following steps to input the parameters that are needed for the service:
   a. url: Input a URL of your choosing. For example:

      ```
      https://www.forbes.com/sites/alexkonrad/2016/01/29/new-ibm-Watson-chief-david-kenny-talks-his-plans-for-ai-as-a-service-and-the-weather-company-sale/#77c0c4bf6693
      ```

   b. features: NLU provides several features for analyzing the text. This example shows the use of the `metadata` feature, which extracts author information, the publication date, and the title of your text/HTML content.

The service parameters should look similar to Figure 2-29.

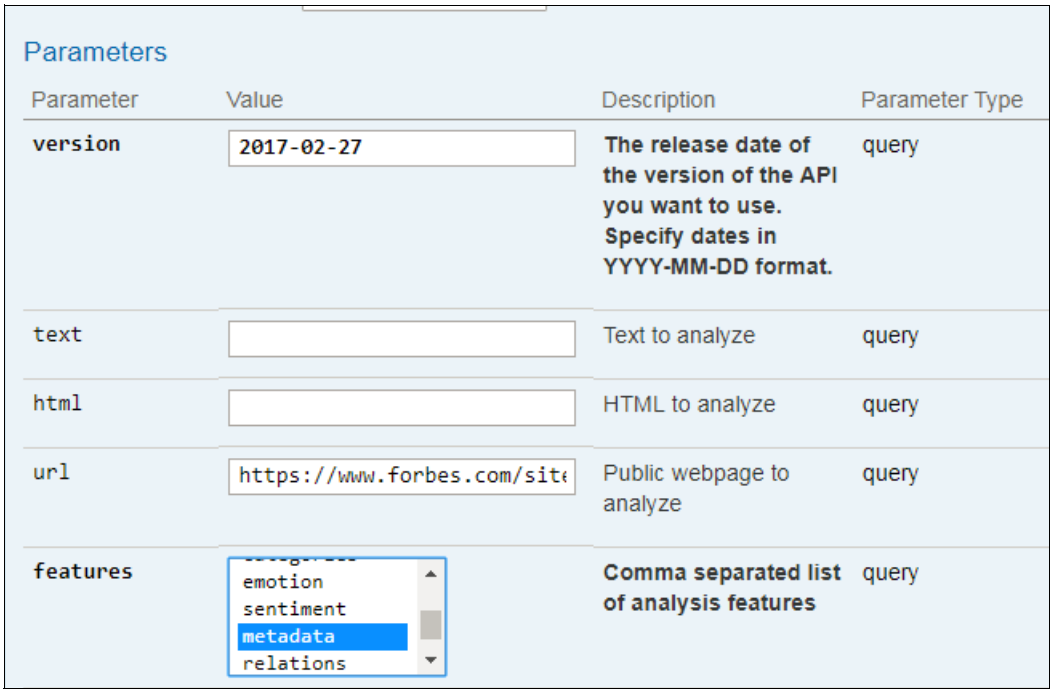

Figure 2-29 Service parameters

4. Click **Try it out!**. Watson API Explorer builds the Request URI, as shown in the following example:

```
https://Watson-api-explorer.mybluemix.net/natural-language-understanding/api/v1
/analyze?version=2017-02-27&url=https%3A%2F%2Fwww.forbes.com%2Fsites%2Falexkonr
ad%2F2016%2F01%2F29%2Fnew-ibm-Watson-chief-david-kenny-talks-his-plans-for-ai-a
s-a-service-and-the-weather-company-sale%2F%2377c0c4bf6693&features=metadata&re
turn_analyzed_text=false&clean=true&fallback_to_raw=true&concepts.limit=8&emoti
on.document=true&entities.limit=50&entities.mentions=false&entities.emotion=fal
se&entities.sentiment=false&keywords.limit=50&keywords.emotion=false&keywords.s
entiment=false&relations.model=en-news&semantic_roles.limit=50&semantic_roles.e
ntities=false&semantic_roles.keywords=false&sentiment.document=true
```

The Watson API Explorer URL that is acting as proxy for the NLU service is `https://Watson-api-explorer.mybluemix.net/natural-language-understanding/api/`.

The resource path is `/v1/analyze`, and all the parameters are passed to the query string of the URL because it is a **`GET`** method. Also, because it is a **`GET`** method, you can try it directly from your browser.

Example 2-3 shows the JSON response that is returned from Watson.

Example 2-3 JSON response that is returned from Watson NLU service

```
{
  "usage": {
    "text_units": 1,
    "text_characters": 4485,
    "features": 1
  },
  "retrieved_url":
"https://www.forbes.com/sites/alexkonrad/2016/01/29/new-ibm-Watson-chief-david-
```

```
kenny-talks-his-plans-for-ai-as-a-service-and-the-weather-company-sale/#77c0c4b
f6693",
  "Metadata": {
    "title": "New IBM Watson Chief David Kenny Talks His Plans For 'AI As A
Service' And The Weather Company Sale",
    "publication_date": "2016-01-29T00:00:00",
    "image":
"https://thumbor.forbes.com/thumbor/600x300/smart/https%3A%2F%2Fblogs-images.fo
rbes.com%2Falexkonrad%2Ffiles%2F2016%2F01%2Fkenny-e1454108067796.jpg",
    "feeds": [
      {
        "link": "/mailto:feedback@forbes.com"
      }
    ],
    "authors": [
      {
        "name": "Alex Konrad"
      }
    ]
  },
  "language": "en"
}
```


The JSON response is a JSON object with a collection of name-value pairs with the keys `usage`, `retrieved_url`, `metadata`, and `language`. The `metadata` key is another nested JSON object that contains `title`, `publication_date`, `image`, `feeds`, and `authors`. The `authors` key is a JSON array that contains the names of the authors of the article.

2.3.2 Watson Visual Recognition

To start the Watson Visual Recognition service on Watson API Explorer, complete the following steps:

1. Access Watson Visual Recognition on Watson API Explorer (Figure 2-30).

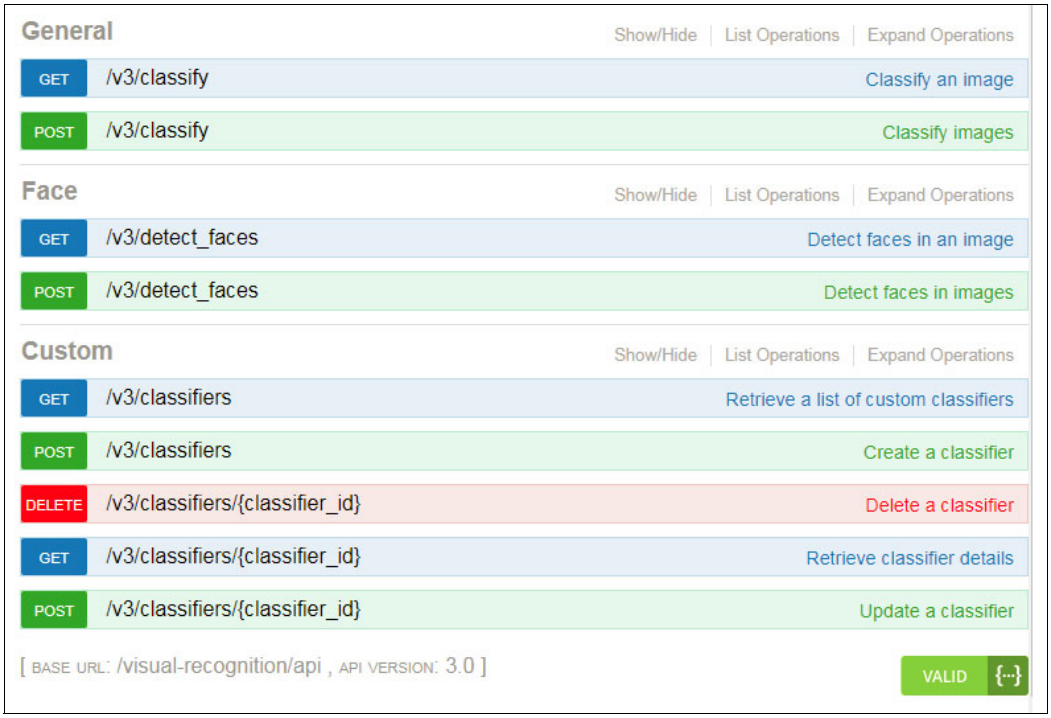

Figure 2-30 Watson Visual Recognition on Watson API Explorer

2. Click **GET /v3/detect_faces**.
3. Complete the url parameter with a URL of an image with faces. For example, you can analyze a picture of the former US president Obama by entering the following URL:

   `https://Watson-developer-cloud.github.io/doc-tutorial-downloads/visual-recognition/prez.jpg`

The parameters should be similar to Figure 2-31.

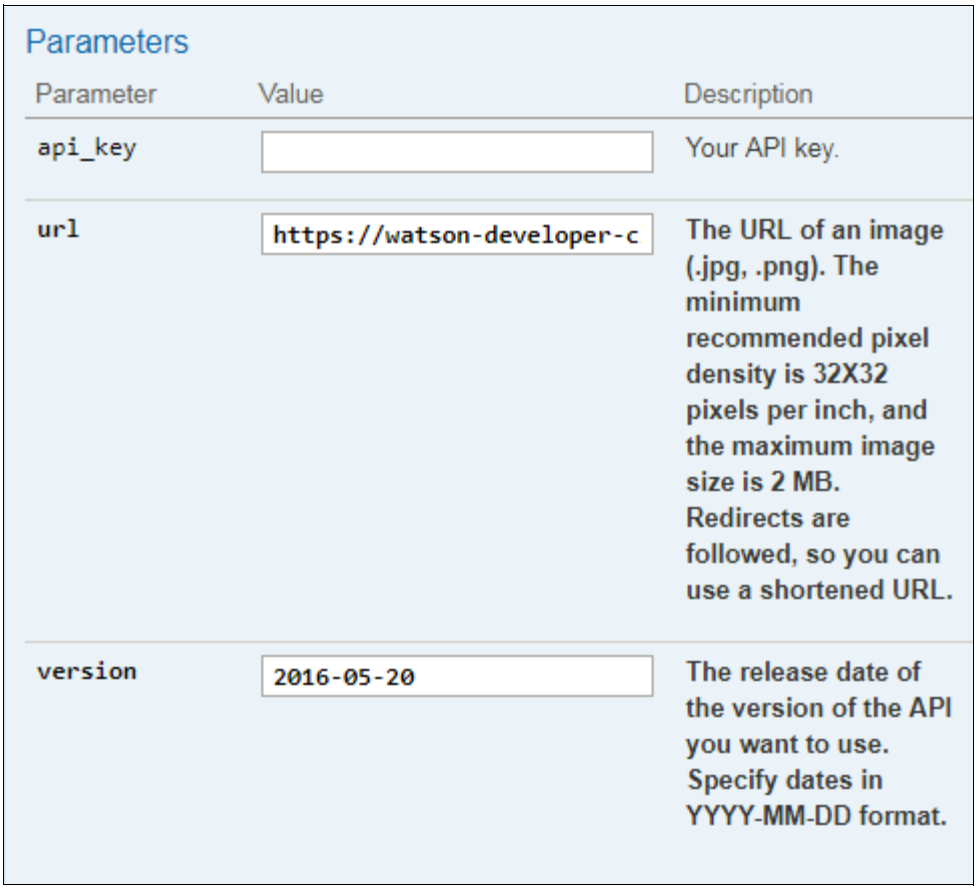

Figure 2-31 Watson Visual Recognition parameters

4. Click **Try it out!**. Watson API Explorer builds the Request URI, as shown in the following example:

   `https://Watson-api-explorer.mybluemix.net/visual-recognition/api/v3/detect_faces?url=https%3A%2F%2FWatson-developer-cloud.github.io%2Fdoc-tutorial-downloads%2Fvisual-recognition%2Fprez.jpg&version=2016-05-20`

   The Watson API Explorer URL that is acting as the proxy for the Watson Visual Recognition service is `https://Watson-api-explorer.mybluemix.net/visual-recognition/api/`.

   The resource path is `/v3/detect_faces`, and all the parameters are passed in the query string of the URL because it is a **`GET`** method.

   Example 2-4 shows the JSON response that is returned from Watson.

Example 2-4 JSON response from Watson Visual Recognition

```
{
  "images": [
    {
      "faces": [
        {
          "age": {
            "max": 44,
            "min": 35,
```

```
            "score": 0.446989
          },
          "face_location": {
            "height": 159,
            "left": 256,
            "top": 64,
            "width": 92
          },
          "gender": {
            "gender": "MALE",
            "score": 0.99593
          },
          "identity": {
            "name": "Barack Obama",
            "score": 0.970688,
            "type_hierarchy": "/people/politicians/democrats/barack obama"
          }
        }
      ],
      "resolved_url":
"https://Watson-developer-cloud.github.io/doc-tutorial-downloads/visual-recogni
tion/prez.jpg",
      "source_url":
"https://Watson-developer-cloud.github.io/doc-tutorial-downloads/visual-recogni
tion/prez.jpg"
    }
  ],
  "images_processed": 1
}
```


Notice the following in the returned JSON:

- `age`: Watson identified the age of the face in the picture to be 35 - 44 with a confidence score of 44.7%.
- `face_location`: The location of the face in the picture.
- `gender`: Watson recognized the gender as Male with a confidence score of 99.6%.
- `identity`: Watson recognized the identity of the face in the picture as Barack Obama with a confidence score of 97%.

2.3.3 Watson Tone Analyzer

To start the Watson Tone Analyzer service on Watson API Explorer, complete the following steps:

1. Access Watson Tone Analyzer on Watson API Explorer (Figure 2-32).

Figure 2-32 Watson Tone Analyzer on Watson API Explorer

2. Click **POST /v3/tone** to analyze the tone of the input content. The method accepts content in JSON, plain text, or HTML format.
3. To input the parameters that are needed for the service, complete the following steps:
   a. `tone_input`: Because this task is a **`POST`** HTTP method, input the parameters in the request body. Complete `tone_input` by using the JSON input that is shown in Example 2-5.

   Example 2-5 The tone_input JSON request

   ```
   {
     "text": "Team, I know that times are tough! Product sales have been
   disappointing for the past three quarters. We have a competitive product,
   but we need to do a better job of selling it!"
   }
   ```

   b. version: Input the version of the Watson service, which in this case is `2017-09-21`.
4. Click **Try it out!**. Watson API Explorer builds the Request URI, as shown in the following example:

   ```
   https://Watson-api-explorer.mybluemix.net/tone-analyzer/api/v3/tone?version=201
   7-09-21&sentences=true
   ```

   In the body, the JSON request that is shown in Example 2-5 is sent to the service.

   Example 2-6 shows the JSON response that is returned from Watson.

   Example 2-6 Example JSON response from Watson Tone Analyzer

   ```
   {
     "document_tone": {
       "tones": [
         {
           "score": 0.6165,
           "tone_id": "sadness",
           "tone_name": "Sadness"
         },
         {
           "score": 0.829888,
           "tone_id": "analytical",
   ```

```
        "tone_name": "Analytical"
      }
    ]
  },
  "sentences_tone": [
    {
      "sentence_id": 0,
      "text": "Team, I know that times are tough!",
      "tones": [
        {
          "score": 0.801827,
          "tone_id": "analytical",
          "tone_name": "Analytical"
        }
      ]
    },
    {
      "sentence_id": 1,
      "text": "Product sales have been disappointing for the past three
quarters.",
      "tones": [
        {
          "score": 0.771241,
          "tone_id": "sadness",
          "tone_name": "Sadness"
        },
        {
          "score": 0.687768,
          "tone_id": "analytical",
          "tone_name": "Analytical"
        }
      ]
    },
    {
      "sentence_id": 2,
      "text": "We have a competitive product, but we need to do a better job of
selling it!",
      "tones": [
        {
          "score": 0.506763,
          "tone_id": "analytical",
          "tone_name": "Analytical"
        }
      ]
    }
  ]
}
```


Notice the following items in the returned JSON:

- The tones `analytical` and `sadness` are the most dominant tones in the document, with confidence scores 83%, and 62% respectively.
- It also shows the tone analysis for each sentence.

2.4 IBM PowerAI

This section introduces the positioning of Watson services and PowerAI in terms of use cases, infrastructure, data, and targeted users.

Watson is an offering that is available on IBM Cloud, made of many different components, and based on multiple technologies. PowerAI is an infrastructure that deployed on-premises and is composed of hardware and supported software, and an AI application can be developed and run on it.

IBM PowerAI is a package of software distributions for many of the major deep learning software frameworks for model training, such as TensorFlow, Caffe, Chainer, Torch, Theano, and their associated libraries, such as CUDA Deep Neural Network (cuDNN) and nvCaffe. These libraries are extensions that take advantage of accelerators, for example, nvCaffe is the NVIDIA extension to Caffe that works on graphics processing units (GPUs). As with nvCaffe, IBM has an extension to Caffe, which is called IBM Caffe. The stack also comes with supporting libraries such as Deep Learning GPU Training System (DIGITS), OpenBLAS, Bazel, and NVIDIA Collective Communications Library (NCCL), as shown in Figure 2-33.

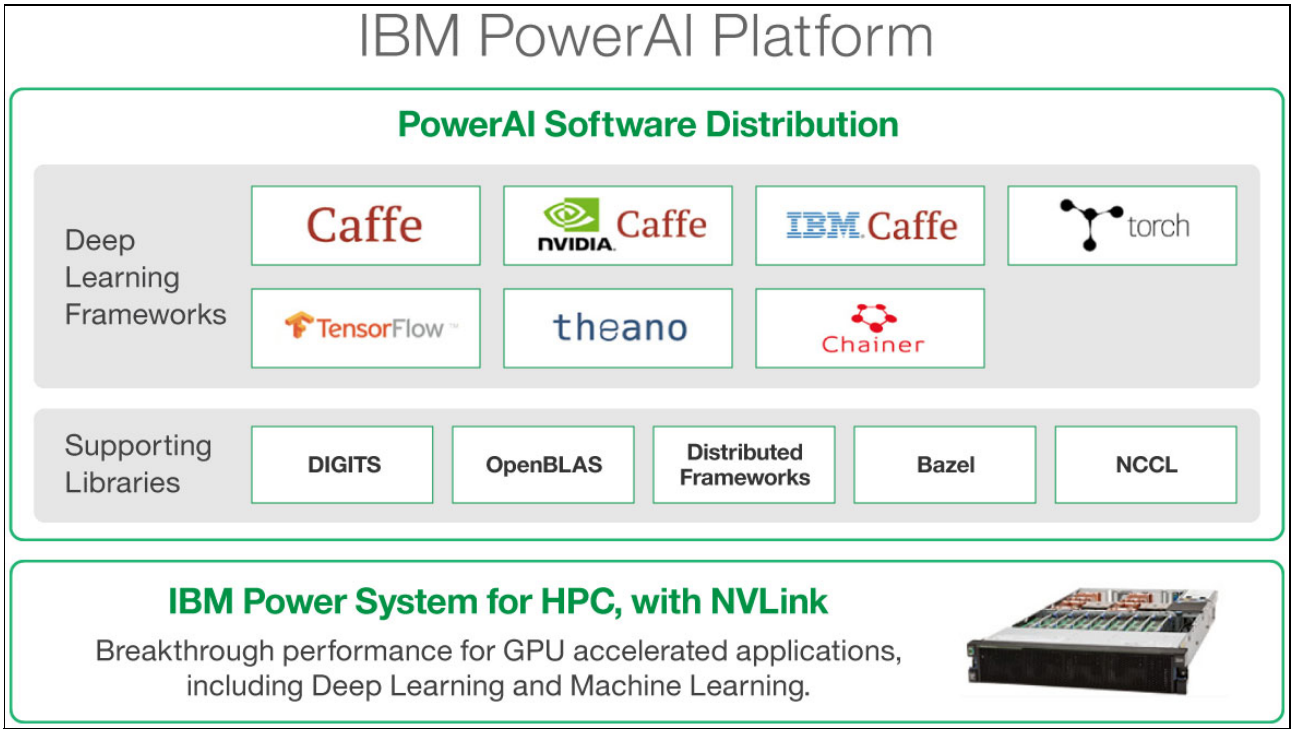

Figure 2-33 PowerAI components

2.4.1 Use cases

Table 2-1 shows the positioning of Watson services and PowerAI from a use case perspective. The key differentiator is that Watson provides pre-built modules that suit different use cases, and PowerAI provides the tools to build your own machine learning algorithms.

Table 2-1 Use cases positioning for Watson services and PowerAI

Watson services	PowerAI
Use Watson services if you need to add one of the following AI capabilities to your application: ► Add a natural language interface to your application to automate interactions with your users. ► Add a cognitive search and content analytics engine to applications to identify patterns, trends, and actionable insights that drive better decision-making. ► Securely unify structured and unstructured data with pre-enriched content, and use a simplified query language to eliminate the need for manual filtering of results. ► Dynamically translate news, patents, conversational documents, or text from one language to another. ► Analyze text to extract metadata from content such as concepts, entities, keywords, categories, sentiment, emotion, relations, and semantic roles. ► Derive insights from transactional and social media data to identify psychological traits that determine purchase decisions, intent, and behavioral traits. ► Convert the human voice into text. It can be used anywhere there is a need to bridge the gap between the spoken word and the written form. ► Process text and natural language to synthesize speech into several voices. ► Use cognitive linguistic analysis to identify various tones, such as joy, sadness, anger, or agreeableness at both the sentence and document level. This insight can then be used to refine and improve communications. ► Find meaning in visual content and analyze images for scenes, objects, faces, and other content. ► Analyze the visual content of images or video frames to understand what is happening in a scene.	PowerAI and the open source machine learning frameworks can fulfill an unlimited number of use cases. Here are the high-level use cases for using PowerAI: ► Machine Learning (including but not limited to): – K-Means Clustering – Supervised Learning – Unsupervised Learning – Reinforcement Learning – Neural Networks ► Deep Learning: – Deep Neural Networks – Transfer Learning – Capsule Learning ► Use cases examples: – Fraud Detection – Credit Analysis – Image Analysis for security

2.4.2 Infrastructure

Table 2-2 shows the positioning of Watson services and PowerAI from the infrastructure perspective. The key differentiator is that Watson services are hosted in the cloud, and PowerAI is hosted on-premises.

Table 2-2 Infrastructure positioning for Watson services and PowerAI

Watson services	PowerAI
Watson services run only in the cloud. IBM Cloud is compliant with ISO 27001, and other universal standards. For more information, see Compliance on the IBM Cloud.	Use it if there is a strong requirement that the data stays local; there might be organizational restrictions or regulatory restrictions, or you need to deploy the full solution on-premises.
Training data can be hosted in the cloud.	Multiple training runs must be carried out locally (higher focus on training) or a large amount of data for training exists on-premises.
Scarce visibility into the infrastructure and its management (compute, storage, and so on).	Complete visibility into the infrastructure and its management.
No need to invest in or manage a data center.	Customer already has a data center, or is planning to building one.

2.4.3 Data

Table 2-3 shows the positioning of Watson services and PowerAI from the data perspective. The key differentiator is that Watson services deal mainly with unstructured data, and PowerAI deals with both unstructured data and structured data.

Table 2-3 Data positioning for Watson services and PowerAI

Watson services	PowerAI
Deals with unstructured data (Systems of Engagement).	Deals with: ► Unstructured data (Systems of Engagement) ► Structured data (Systems of Records)
Supports text, audio, images, and videos formats.	Supports text, audio, images, and video formats in addition to transactional data and warehouse data.

2.4.4 Targeted Users

Table 2-4 shows the positioning of Watson services and PowerAI from the user targets perspective. The key differentiator is that Watson services are used mainly by application developers, and PowerAI is used mainly by data scientists.

Table 2-4 Targeted users positioning for Watson services and PowerAI

Watson services	PowerAI
Used mostly by application developers.	Used mainly by data scientists.
Easier to use because it hides data science complexity.	Provides full data science capabilities.
Add AI capabilities to applications quickly.	Slice and dice data to get optimal results, which requires longer training efforts.

3

Typical IBM Power Systems applications

This chapter explores typical Power Systems based applications from a business and technical perspective.

The chapter includes the following topics:

© Copyright IBM Corp. 2018. All rights reserved.

3.1 Benefits of Power Systems servers

Welcome to the cognitive era. The rise of machine learning, natural language processing, speech and visual recognition, big data, analytics, and social business are creating explosive growth in compute-intensive workloads, including transactional processing.

These applications are changing business needs for powerful computing. To keep competitive, companies are looking for increased performance, scalability, and virtualization to address their compute requirements.

With Power Systems servers, businesses of all sizes can accomplish the following tasks:

- *Reduce costs and improve information technology (IT) efficiency* with systems that provide efficient and rapid response to workload demands, enabling consistently high service levels across hundreds of virtual workloads on a single system at an affordable price.
- *Enable new and growth workloads, such as data and analytics, to drive faster insights* with servers that are optimized for big data and compute-intensive analytic applications.
- *Deliver an exceptional client experience* with systems and software that provide unique capabilities, Power Systems experts and expertise, and a vast network of IBM Business Partners and solutions providers to ensure that you are receiving rapid and long-lasting benefits from your investment.
- Performance: POWER8 offers a full line of servers with up to 256, high-performance cores and eight threads per core. New POWER9 processor-based servers come in versions with either four or eight threads per core for workload flexibility, a shorter instruction pipeline, and the ability to use attached or buffered memory for either scale-out or scale-up servers. Intel sells smaller systems with current offerings topping out at 60 cores and two threads per core that require much more heating, cooling, and management.
- Virtualization: IBM PowerVM is the only hypervisor in the market that can boast of zero security vulnerabilities. No other x86-based hypervisor can make that claim. PowerVM is in the firmware, which reduces latency and enables much higher overall usage of the hardware.
- Resilience: Power Systems servers are engineered by the same team that engineered the IBM Mainframe. That same resilience is built into Power Systems servers, enabling higher uptimes and ensuring 99.997% uptime in a calendar year. Intel cannot come near that claim. Power Systems hardware is self-healing and the most critical updates can be performed without any downtime through features such as Live Partition Mobility (LPM). By contrast, x86 servers currently do not have the same mission-critical resilience characteristics.

Suited to compute-intensive workloads, Power Systems servers have a large range of designed and optimized offerings for small and enterprise business, including IBM PowerVM virtualization software, IBM PowerHA software for high availability, and IBM PowerSC™ software for security and compliance. Developing the systems and software together enables high systems usage, high integration from independent software vendors (ISVs), and a choice of AIX, IBM i, or Linux operating systems (OSs).

3.2 Use case scenarios

This section describes some hypothetical business scenarios that are typical of IBM Power Systems:

3.2.1 Hospitality industry

In this scenario, a hotel chain with 11 different brands holds information about their facilities and their guests on a Power platform.

The challenge

A target for all companies of all sizes is to enable direct interaction with customers and provide a better experience. Following the trend and exponential growth of chatbots, this hotel chain wants to use its knowledge base about its guests and create a tool that enables them to choose the next best experience at any of their hotels from all of their brands. By combining all information about the hotels, such as location, facilities, amenities, price, and guest history, such as hotels they have visited, duration of stay, and the chosen room category, the hotel chain is ready to offer suggestions for the next trip. The hotel chain chooses to create a chatbot that interacts with customers as a complimentary customer service agent.

The solution

Taking advantage of the system on Power Systems servers to query data about hotel and guests, this company built a tool that enables its guests to find detailed suggestions to highly specific questions. Guests interact with the tool by using the company’s website, social media pages, or the mobile application.

Using the Watson Assistant service, the hotel’s cognitive solution connects to the IBM Cloud and enables guests to ask questions in natural language, for example “I am planning my next trip to Austin, do you have any suggestions?”. The solution accounts for the guest’s prior experiences and responds with the appropriate response.

The solution was developed by training the Watson Assistant service to recognize many different intentions, several entities, and multiple dialog flows. The solution follows these simple steps:

1. Guests ask Watson for recommendations in natural language, such as “Where should I stay on my next trip to Austin?”.
2. The application logic inspects the intention and entities that are identified by Watson during the conversation with the guest. For the previous example, it would be “Intent: hotel recommendation, Entity: Austin”.
3. The application sends a request to the Transformation and Connectivity node that uses guest data and the entities that are identified to create a query and run it against the Enterprise Data.

4. Finally, the application responds to the guest with a list of recommendations.
5. Guests can ask more questions, provide more information, or book a reservation based on the information that is received.

Architectural design

Figure 3-1 shows a typical conversation flow and how the information is processed.

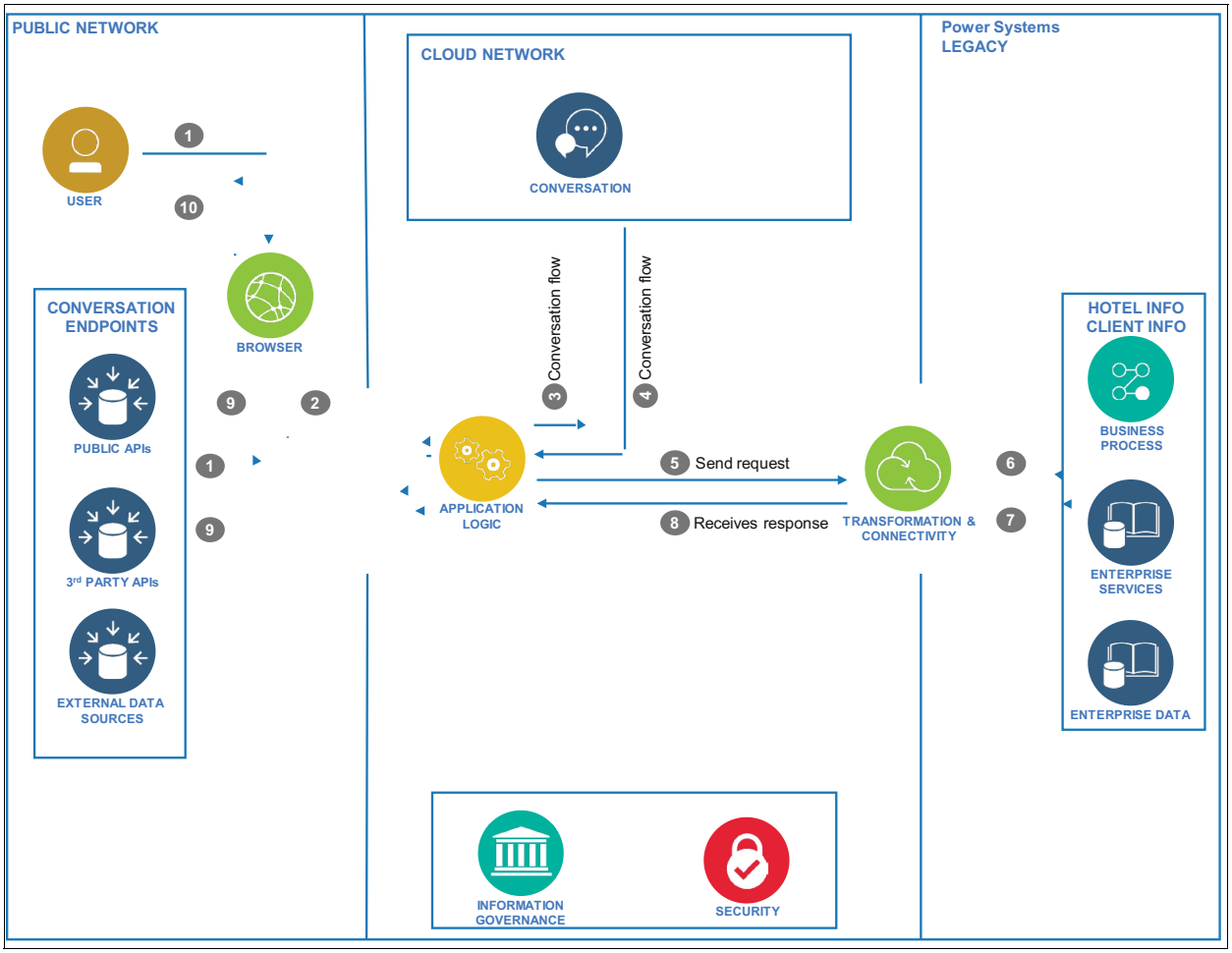

Figure 3-1 Architectural design document for the hospitality industry scenario

1. A guest interacts with the application from a browser on the public internet or by using another tool, such as a mobile application, a third-party service, or social media.
2. If the guest is using the web application, the page renders on the guest's interface.
3. The application sends the guest's input to the Watson Assistant service on the IBM Cloud network.
4. The application receives the intent and the entities that are identified by Watson based on the guest's input.
5. If Watson is confident of the guest's intent, the application sends a request to the Power Systems system through a Transformation & Connectivity module. If Watson returns a low level of confidence, the application interacts with the guest asking for more information.

6. The Transformation & Connectivity module creates a query that is based on the parameter that is received and runs it on a DB2 database that contains hotel information (for example, location, amenities, or prize) and the guest's information (such as membership category, last stays, or preferences).
7. The query response is sent to the Transformation & Connectivity module.
8. The Transformation & Connectivity module sends the response back to the application.
9. The application formats the result and returns a list of suggestions to the requester tool.
10. The result is rendered on the guest's interface. The guest can interact again with the application for more information.

Benefits

Before this solution, only customer service agents had access to the information by using internal tools. Now, there is a self-service system that uses natural language and the hotel's brand.

All hotel guests can receive recommendations based on their prior experiences for all locations that the hotel chain has facilities for worldwide.

Watson helped reduce the time spent searching for guest information and location data to make quicker, smarter, and more fact-based decisions.

3.2.2 Agriculture industry

In this scenario, a company in the agricultural industry manages a large plantation and needs to monitor plant health to avoid infestation. They want to create a new application by using Internet of Things (IoT) and Watson services to analyze photographs of the plantation by using the Power platform.

The challenge

Plantation health is a top concern of this industry, which is susceptible to infestations.

This company is looking for a 25% reduction in plant infestation over a period of 10 months. It is necessary to understand patterns of infestations, how they relate to weather and environmental conditions, and how to set realistic plant budgets.

For this company, achieving this goal requires more than just the collection of information weekly for investigation. It requires more intensive monitoring of the entire plantation. Because of the large scale and inaccessibility of the facilities, an implementation of monitoring cameras is not a viable solution.

The solution

The company uses the Watson Visual Recognition service to identify features from drone images and insights into the dynamic patterns of weather to create a complete solution running on a Power platform.

Because a cognitive solution can process more information than a manual process, and a drone can collect many more pictures in less time than human inspection, the solution generates a full view of plantation health. The system is trained to identify many different types and degrees of infestation.

Figure 3-2 shows one example of an infested plant that is analyzed by the tool.

Figure 3-2 Unhealthy plant picture to be analyzed by the tool

Architectural design

Figure 3-3 shows the data from each layer of the application.

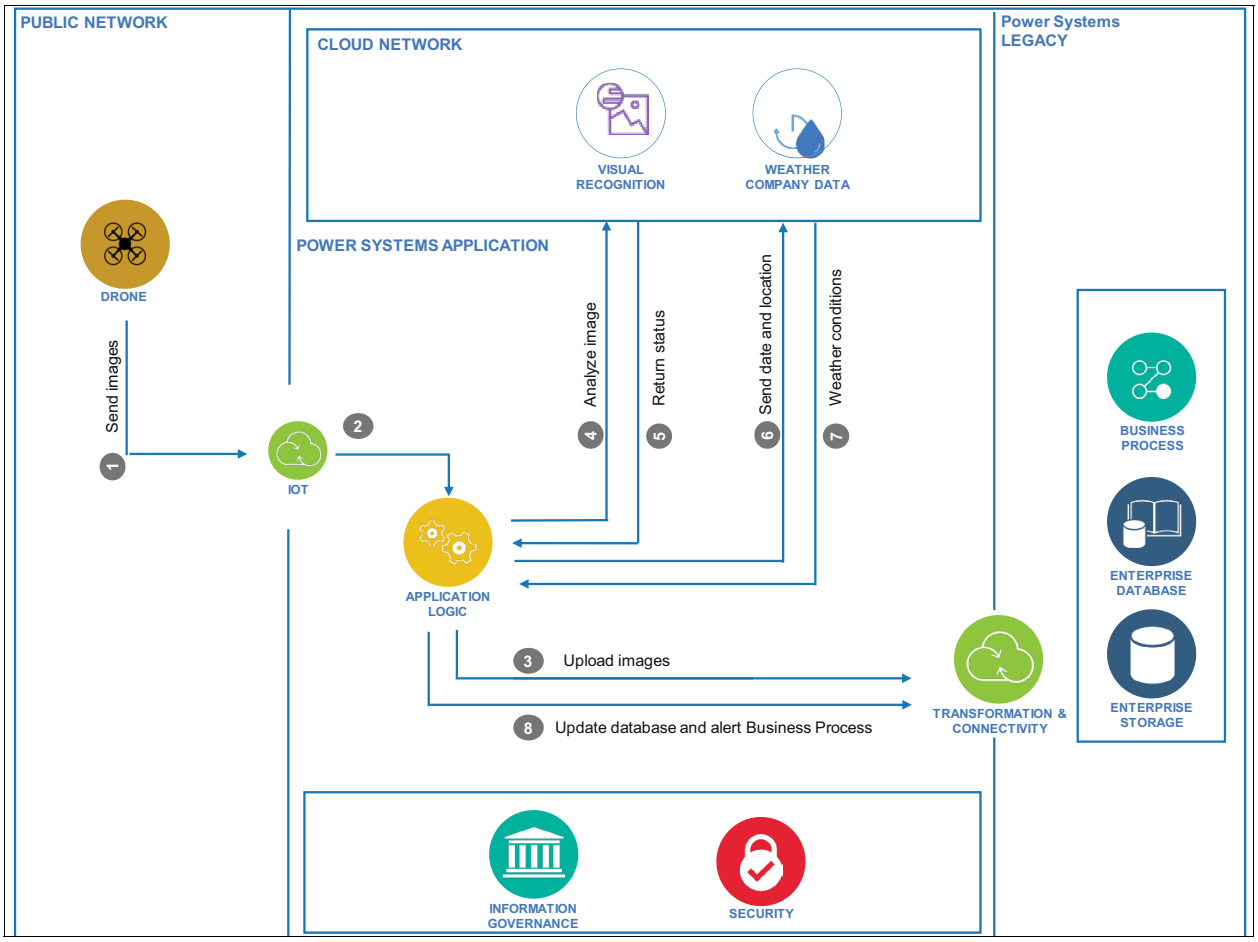

Figure 3-3 Architectural design for the agriculture industry scenario

1. A drone flies over the plantation, records images, and sends them to an IoT device.
2. The IoT device sends the images to the application.
3. The application uploads the images into the Power Systems server on the enterprise system by using the Secure File Transfer Protocol (SFTP) protocol.
4. The images are sent, one by one, to the Watson Visual Recognition service on IBM Cloud.
5. Watson returns with a status that is based on the recognition of each image.
6. The application sends the location, date, and time where the image was taken to the Weather Company Data service on IBM Cloud.
7. The Weather Company® Data service returns the weather conditions for the location at the time that the image was taken.
8. All the information is stored on the Power Systems enterprise DB2 database. If any evidence of infestation is found, the Business Process application is alerted.

Benefits

The solution enables the company to process instantly images several times faster than was possible with the manual process. The company can now process images multiple times per day every day.

The solution also provides a capability to combine the images with prevailing weather conditions and creates insights for administrators and the research team to understand infestation behavior.

3.2.3 Human resources talent management

In this scenario, a company has many employees working in different business units worldwide, and many roles and positions. Talent management in this scenario is a challenge for medium- and large-sized companies worldwide.

Challenge

How do you ensure that you have the correct skilled and talented employees working in the correct position and reduce unwanted attrition of talent leaving to start careers with your competitor? This is a question common among all HR departments who look for a solution similar to this use case.

For example, IT is constantly evolving. The skill set that is required to operate effectively in the IT field is also constantly evolving. Skills that were in demand two years ago might be redundant two years in the future. The challenge is to know that your employees have the correct knowledge and motivate them to train and keep their skills relevant.

The solution

An internal career advisor tool can make it easy for a company to support each employee on the path to the employee's next promotion and identify potential talent for other opportunities.

Taking advantage of the HR system and storage on Power Systems servers, the HR department receives updated resumes from employees and identifies opportunities that best fit with the employee's skills and aspirations. The tool can also provide recommendations for training that the employee can take to be better prepared for the opportunities and move to the next level of the employee's career inside the company.

Using the Watson Assistant service, the cognitive tool connects to the IBM Cloud and enables employees to ask questions in natural language, such as "What opportunities are available for me?". Watson asks for the most updated version of the candidate's resume and builds a response that is appropriate to the key data that is extracted from the resume, matching experience against all open opportunities.

Watson Assistant is trained to identify multiple different intentions and entities, and keep a conversation with different dialog flows. This is a typical conversational flow:

1. Employees ask Watson for career recommendations in natural language, such as "What opportunities are available for me?".
2. The application identifies that this is the candidate's first iteration and asks for an updated resume to be uploaded.
3. After the file is uploaded, the file is stored on the Power Systems server and a sequence of analysis begins:
    a. Watson Personality Insights extracts information from the candidate's resume and returns personality insights, such as the ones that are listed in the Personality models note box that follows step b.
    b. Watson Natural Language Understanding (NLU) analyzes the resume and extracts entities, keywords, and concepts.

Personality models:

The Watson Personality Insights service is based on the psychology of language in combination with data analytics algorithms. The service analyzes the content that you send and returns a personality profile for the author of the input. The service infers personality characteristics based on three models:

- *Big Five personality characteristics* represent the most widely used model for generally describing how a person engages with the world. The model includes five primary dimensions:
  - *Agreeableness* is a person's tendency to be compassionate and cooperative toward others.
  - *Conscientiousness* is a person's tendency to act in an organized or thoughtful way.
  - *Extraversion* is a person's tendency to seek stimulation in the company of others.
  - *Emotional range*, also referred to as *neuroticism* or *natural reactions*, is the extent to which a person's emotions are sensitive to the person's environment.
  - *Openness* is the extent to which a person is open to experiencing various activities.

  Each of these top-level dimensions has six facets that further characterize an individual according to the dimension.
- *Needs* describe which aspects of a product resonates with a person. The model includes 12 characteristic needs.
- *Values* describe motivating factors that influence a person's decision making. The model includes five values.

4. With all this information, the application runs a query on the opportunities database and returns a list of opportunities that matches the candidate’s profile.

Architectural design

Figure 3-4 describes a step-by-step application flow.

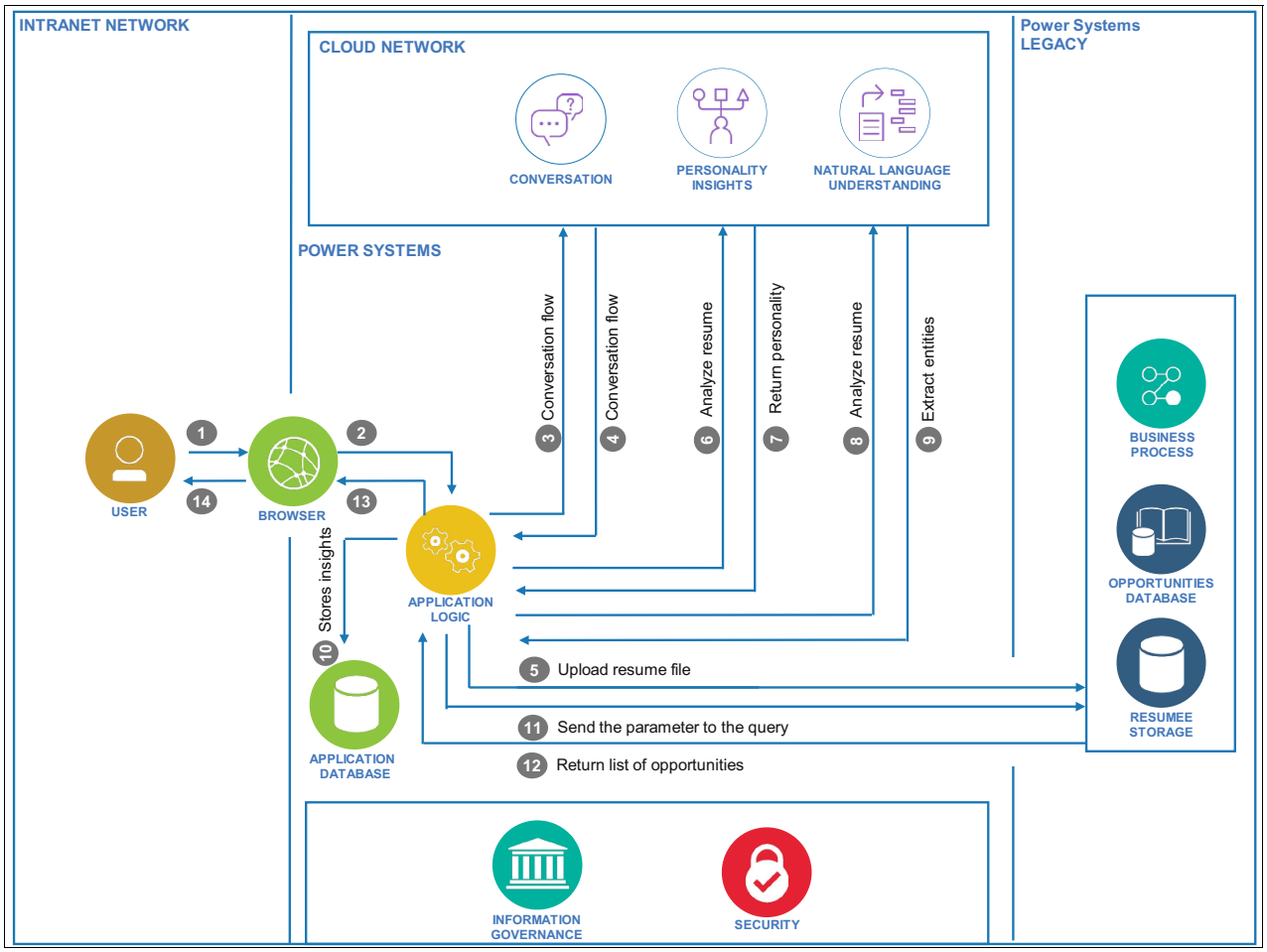

Figure 3-4 Architectural design for the human resources talent management scenario

1. The employee accesses the application by using a browser on the secure intranet network.
2. The application receives the employee's input.
3. The application sends the employee's input to the Watson Assistant service on IBM Cloud.
4. The Watson Assistant service identifies the employee's intention and extracts the entities, if there are any.
5. If the intention is a new conversation, the application asks to upload the updated version of employee's resume by an SFTP connection. The file (PDF or `.doc` format) is stored in the Power Systems server.
6. The Watson Personality Insights service running on IBM Cloud receives the resume content and analyzes the employee's personality.
7. The employee personality insights are sent to the application.
8. The NLU service running on IBM Cloud receives the same resume content and extracts entities, keywords, relationships, and the categories.
9. All information that is extracted is sent to the application.

10. The original resume content, the personality insights, and all the information that is extracted from the NLU service are stored in a local instance of a DB2 database.
11. With all information as parameters, the application runs a query on a Power Systems DB2 instance database.
12. The database returns a list of opportunities that match with the employee's profile.
13. The list of opportunities renders in the browser.

Now, the employee can ask more questions to start the conversation flow again.

Benefits

With the implementation of this new tool on their intranet, employees can explore career paths inside their company. All employees have direct access, can evaluate their competencies, and can manage their career progression through opportunities within the organization, across business units, and across job functions. They can evaluate themselves against the requirements of the roles. They are better prepared to discuss career paths with their managers.

The company now has valuable and updated information about their resources and can easily identify talent.

Better communication with managers along with more transparency about career progression within the organization improves employee loyalty and facilitates retention efforts.

3.2.4 Customer service feedback: Migration plan

In this scenario, a worldwide company in the telecommunication industry manages a large set of products and wants to migrate the customer service feedback database to a cognitive system that identifies their customer's opinions on their products to generate insights that can improve customer satisfaction.

Challenge

The company holds all customer feedback information on a DB2 instance running on a Power platform, and wants to extract the insights and store them in another instance of DB2 that is accessed by a dashboard application. This application generates a visual dashboard with the information that enables the customer service team to improve customer satisfaction by identifying complaints.

Because it is a worldwide company, customer feedback can be stored in different languages and must be translated into English before it can be analyzed by the Watson services.

Another concern is around data privacy. Therefore, the company mandates an extra layer that masks customer data before it can be sent to the cloud.

The solution

Identifying and understanding the views of customers is an important advantage for customer retention. Using cognitive computing to analyze customer satisfaction for their products and services, a company in the telecommunication industry can generate insights to provide a better experience for their customers.

Regulation is also a concern. This company must take care of customer-sensitive information when sending data to the cloud. To avoid any risk of data exposure, the solution contains a security layer that masks customer data before sending it outside of their private network, and unmasks data when it returns to the intranet network.

Using the language identifier and translate methods in the Watson Language Translator service on IBM Cloud to identify the original language, the feedback is recorded and translated into English. The solution can extract the tone of the content by using the Watson Tone Analyzer service and escalate customer feedback or find opportunities to improve customer services. Using the NLU service on IBM Cloud, the solution extracts entities such as company name, keywords such as the product name, and sentiment for each identified entity.

When migrating voice records, the Watson Speech to Text service transcribes the audio into text before performing the tone and NLU processing.

Another application running on the Power platform reads the extracted information and creates a dashboard, enabling customer service agents to identify difficulties and work with the customer to improve satisfaction and identify what is going well.

Architectural design

Figure 3-5 describes the flow of the database migration to a cognitive solution.

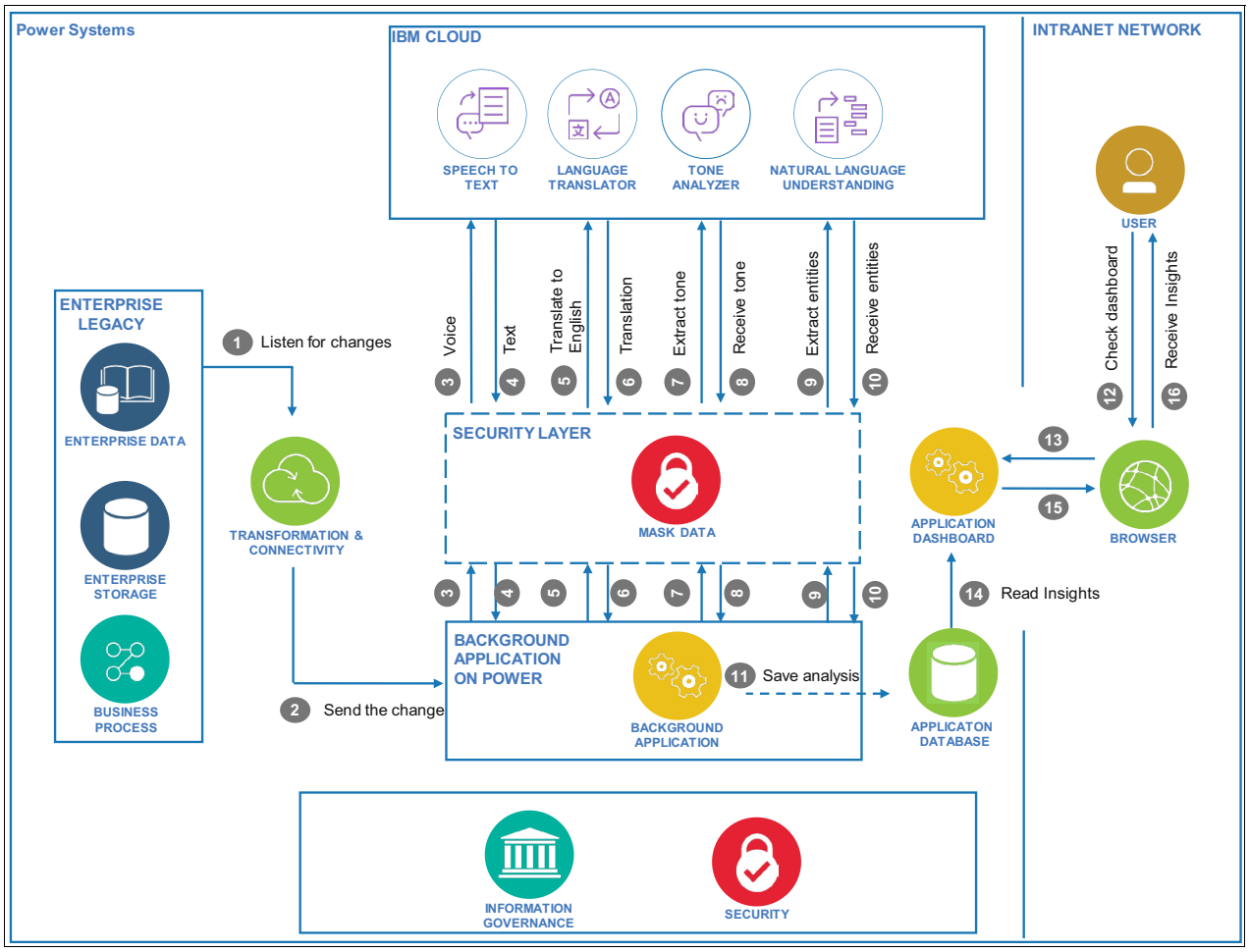

Figure 3-5 Migration scenario for the customer service feedback system

1. The solution loads all customer feedback information that is stored in the Power Systems application. It can be from a database in case of feedback that is received from their web application, social media network, or voice recording files (`.wav` audio files) when extracted from phone calls.
2. An intermediate layer that is connected to the enterprise database sends the data to an application running in the background. This application orchestrates all communication with IBM Cloud and the internal systems.

   All sensitive data is masked as it passes through the security layer that is developed to protect customer-sensitive information by following regulations that this company must adhere to in the countries that it operates.
3. If an audio file is received by the application, the file is submitted to the Watson Speech to Text service on IBM Cloud. If it is not an audio file, skip to step 5.
4. Watson returns transcribed text to the application, which passes through the security layer on the return.
5. Because the company is a worldwide one, the customer feedback might be in multiple different languages. The content is submitted to the Watson Language Translator service, which identifies the source language and translates it to English if required.
6. The translation is sent back to the application.
7. The text is submitted to the Watson Tone Analyzer service.

   If the text is in English or French, the original text is sent; otherwise, the English translated text is sent.
8. Watson identifies whether the customer is happy with their products or services, and sends this information to the application.
9. The text is submitted to IBM Cloud again to be analyzed by the NLU service, which extracts the entities, keywords, and sentiment score.

   Depending on the language that is identified in step 3, the application sends the original text or the translated version. For more information about the supported languages for each feature of the NLU service, see Detectable languages.
10. The application receives the features that are extracted from the NLU and removes the mask.
11. The application saves all the information (original text, translated version (if not English), tone, entities, keywords, and sentiment score) in an instance of the DB2 database.
12. The customer service agent accesses another web application through the browser.
13. An internal application, running on the local intranet, that the user accesses.
14. All the insights that are extracted from the customer feedback database are read from the database.
15. A visual dashboard renders on the customer service agent browser.
16. Customer service receives the information and can take appropriate action.

Benefits

Before the migration of the customer feedback to the new cognitive system, the company had only the limited options that their customers had available to signal their sentiment for products and services. Now, the company has a dashboard showing the true sentiment for each product or service, which is detected through natural language.

The customer service agent can take the appropriate action based on a deep analysis of customer feedback and can escalate customer feedback when it turns sour, or find opportunities to improve customer service scripts, dialog strategies, and customer journeys.

3.2.5 Customer service feedback: Real-time monitoring

In this scenario, a worldwide company in the telecommunication industry manages a large set of products and wants to identify customer satisfaction in real time when their customers are using their channels to provide feedback. A cognitive system identifies their customer's opinion on their products to generate insight.

Challenge

The ability to monitor customer satisfaction in a real-time environment is a competitive advantage that companies of all sizes are pursuing. When a company operates worldwide, creating a solution to understand customer needs in different languages becomes much more complicated. Because it is a worldwide company, customer feedback can come in different languages and must be translated into a standard language before being analyzed and interpreted.

The handling of customer-sensitive information adds concerns about regulations and compliance that must be part of the solution. To satisfy the requirement around data privacy, the company requires an extra layer to mask customer data before the data can be sent to the cloud.

The solution

The ability to properly identify and understand opinions of customers is an important advantage in customer retention. Using cognitive computing to analyze customer satisfaction about their products and services, a company in the telecommunication industry can generate insights to provide a better experience for their customers.

Regulation is also a concern, and this company must carefully handle customer-sensitive information and avoid any risk of data exposure. The solution contains a security layer that masks customer data before sending it outside of the private network and unmasks data when it returns to the intranet network.

The customer can use different channels to provide feedback (company's website, social networks, mobile application, or phone calls) by using the customer's native language.

Using Watson Language Translator service on IBM Cloud to identify the original language, the feedback is recorded and translated to English. The solution can extract the tone of the content by using the Watson Tone Analyzer service and escalate customer feedback or find opportunities to improve customer services by using the NLU service on IBM Cloud. The solution extracts entities such as the company name, keywords such as the product name, and the sentiment for each entity.

Architectural Design

Figure 3-6 describes the flow of the database migration to a cognitive solution.

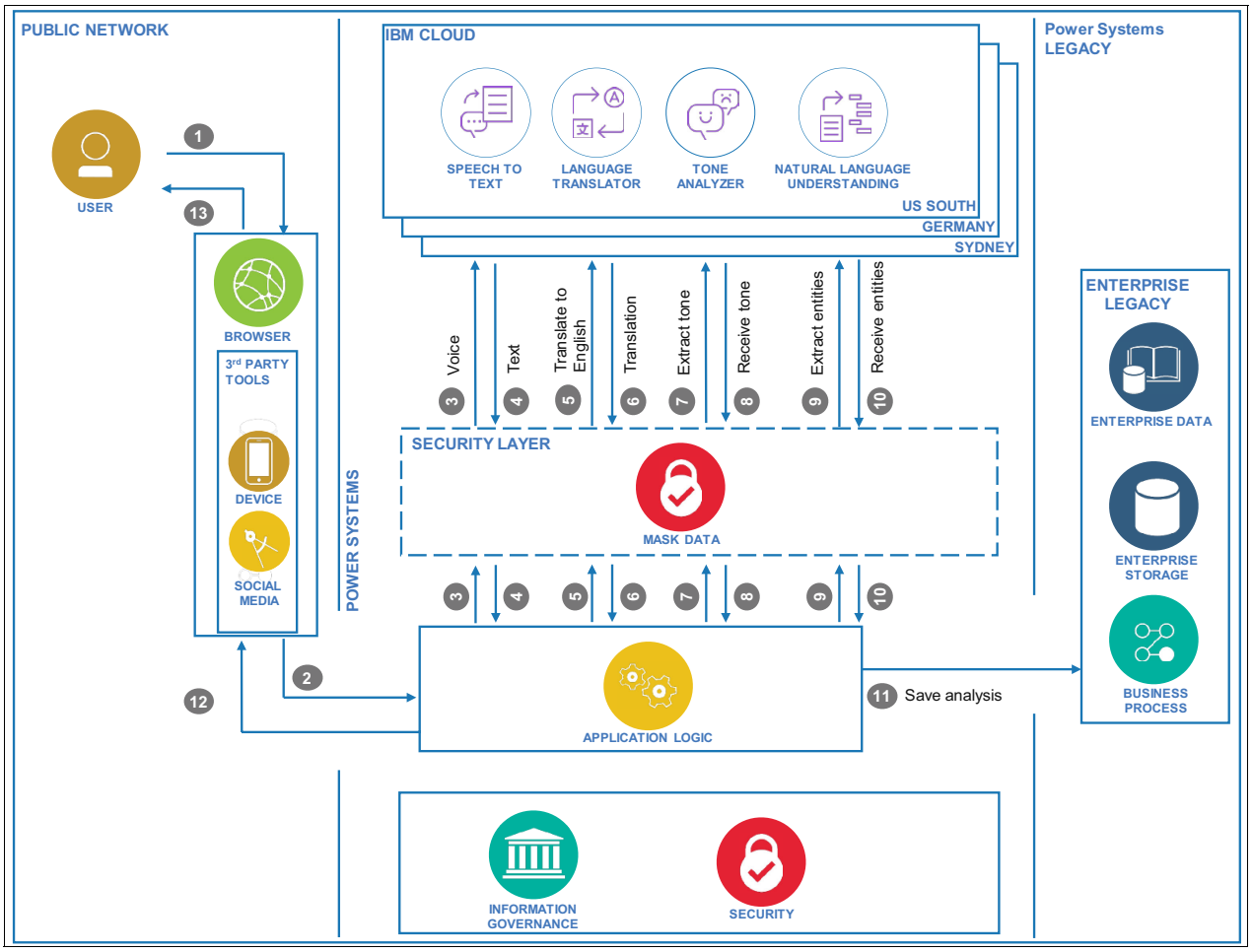

Figure 3-6 Real-time scenario for the Customer Service Feedback system

1. The customer interacts with the application by using one of the different channels (company's web page, social network, mobile application, or phone calls).
2. The content is submitted to the application.
3. If a phone call or a voice message is identified, the application sends the content to the Watson Speech to Text service on IBM Cloud.
4. Watson returns a transcript of the voice.
5. The text is submitted to the Watson Language Translator service, which identifies the original language and translates it into English if needed.

   Because it is a worldwide company, the customer feedback can be in multiple different languages.
6. The translated text is sent back to the application.
7. The text is submitted to the Watson Tone Analyzer service.

   If the text is in English or French, the original text is sent; otherwise, the English translated text is sent.
8. Watson identifies whether the customer is happy with their products or services and sends this information to the application.

9. The text is submitted to IBM Cloud again, but this time to be analyzed by the NLU service, which extracts the entities, keywords, and sentiment score.

   Depending on the language that is identified in step 5 on page 67, the application sends the original text or the translated version. For more information about the supported languages for each feature of the NLU service, see Detectable languages.

10. The application receives the features that are extracted from the NLU service and removes the mask.

11. The application saves all information (original text, translated version (if not English), tone, entities, keywords, and sentiment score) in an instance of the DB2 database.

Benefit

Before the implementation of the customer feedback real-time monitoring cognitive system, the company could rely only on the limited options that were available to customers to indicate their sentiment about products or services. Now, the company has access to real-time sentiment and tone for each product or service, which are determined from the natural language the customers use on all the contact channels, such as the internet, social media network, or phone calls.

The customer service agent can take appropriate action based on a deep analysis of customer feedback and escalate customer feedback when they turn sour, or find opportunities to improve customer service scripts, dialog strategies, and customer journeys.

3.2.6 IBM Watson Health use cases

The IBM Watson Health™ cognitive services apply natural language processing and cognitive reasoning to healthcare data, enabling developers to create a generation of cognitive applications and solutions in healthcare and life sciences.

This section briefly introduces specific scenarios for solutions running on Power platforms that are connected to Watson Health services. For each sample, a scenario description is provided and how the combination of the Power platform with the Watson Health services can provide a solution to customers.

The Watson Health services are not available in IBM Cloud. They are a unique offering directly from IBM. If you are interested in using these services, then contact Watson Health Consulting.

Performing triage on patients by using a chatbot

Performing triage on patients in the waiting room can save time and money for hospitals and clinics. What if the triage begins even before the patient arrives at the hospital?

Using patient historical data that is stored on a Power Systems environment, a company in the health industry can listen to patients talking about their conditions by using both Watson Speech to Text and Watson Assistant services. The application combines with the patient's historical data and schedules a consultation with the most appropriate doctor, or routes the patient to a pre-examination.

This solution can be combined with Watson Health services such as the Annotator Service for Clinical Data, and Medical Insights, for a more robust solution.

Cognitive scribe

Increasingly complex reimbursement requirements lead to physicians spending more time on documentation. Shortcuts to save time, such as copying and pasting, reduce documentation quality and value. This effort reduces the time that is spent with patients and contributes to physician burnout.

The solution uses a combination of the Insights Service for Patients Data and the Annotator Service for Clinical Data, which are available on Watson Health, and the Watson Speech to Text service, which is available on IBM Cloud. The solution generates a document summarization for easier access to patient data at the point of care by capturing patient encounters and updates as they happen, which alleviates documentation.

Speech recognition plus annotation are used to help the post-visit documentation process so that doctors spend more time treating patients.

Chronic disease management

The coordination of care for chronic diseases, such as diabetes and hypertension, is complex and costly. Providers do not have the time to read through hundreds of pages of notes to understand the history of care and trends of a patient.

The solution taps into the structured and unstructured data from a patient's Electronic Medical Record (EMR). It provides an at-a-glance summary of the patient's status (key vitals, labs, and medications that are related to the chronic disease) and a visualization of key trends and the history of care by using the Insights Service for Patient Data on Watson Health. The solution provides direct access to the organized unstructured EMR data, which empowers the providers with access to the data that they need to coordinate the care of chronic disease patients.

Transition of care assistant

The medical condition that is listed in an EMR, which is critical to billing and coding, is rarely kept up to date, and the discharge process requires a lengthy manual review of many days or weeks of care notes. Providers seeing a referred or new patient must get up to speed on the full history of the patient in minutes.

The solution uses of the Insights Service for Patient Data on Watson Health to generate a problem list that is backed by evidence in structured and unstructured EMR data. The solution provides an at-a-glance summary of the patient's history and key information that are tied to problems, enabling care team collaboration by summarizing patient data for easier discharge or transition of care.

Coding and reporting

Medical coding, directly tied to billing and revenue, in an error-prone process requires reading all of the clinical notes from a visit. Reporting in areas such as cancer is expensive and time-consuming. Time is spent adhering to regulations to avoid fines rather that deriving insights from the aggregated data.

The solution uses the Annotator Service for Clinical Data and the Terminology Service on Watson Health to annotate clinical tests, and pull out the key concepts, including Current Procedural Terminology (CPT) codes for the coding and reporting professionals,

The solution uses existing medical ontologies, including Unified Medical Language System (UMLS) and Systematized Nomenclature of Medicine (SNOMED) to pre-code concepts and identify their context in the medical text to help the coding and reporting processes.

3.2.7 Other scenarios

This section briefly introduces other typical scenarios for solutions running on a Power platform that is connected to Watson services on IBM Cloud. For each sample, a scenario description and how the combination of the Power platform and the Watson services can provide the solution to the customers is provided.

Retail industry

Companies in the retail industry have much information about their products and customers, and a combination of this information can generate insights that helps sellers to identify opportunities to offer new products or services for their customers.

A good example of a solution that can be developed on the Power platform is a cognitive chatbot that uses the Watson Assistant service, which can identify customer intention and entities during a chat conversation. The application can generate a query to search for products on the company database to respond to customer requests and include suggestions for a complimentary experience.

Insurance reports that use natural language

Handling of insurance claims requires people that are involved in car accidents to describe the incident and the conditions. Each person provides their own version of the facts.

This situation is why insurance companies use the NLU service to extract the entities, keywords, and sentiment score from the people that are involved, and use the Watson Visual Recognition service to analyze the picture and identify the severity of the accident and prevailing environmental conditions.

Educational industry

When it comes to education, data shows that deep knowledge and practical skills in science and math levels the playing field for young people regardless of their ethnicity and location.

Although success in math in elementary grades connects directly to achievement across all subjects, elementary school teachers, who teach all subjects, often do not get the preparation or the support that they need to improve their ability to teach math effectively.

Companies in the educational industry can use the Watson Discovery service with NLU and Watson Knowledge Studio services to create target recommendations for teachers.

The solution enables students to find relevant lessons, activities, standards information, and strategies faster than ever, all from a corpus of proven-effective materials that are recommended by educators.

Information technology industry

It is common for an enterprise environment to be formed of different applications, and sometimes the user becomes lost in the midst of available tools and options.

The Watson Natural Language Classifier can help the user to understand the language of short texts and predict how to handle them. A classifier learns from your example data and then can predict information for texts that it is not trained on.

The solution uses the service to take predictive actions and routes users to the correct application. If the user has a complaint, the solution open issues in another tool to determine the severity based on the text. By incorporating the Watson Speech to Text service in the same application, the solution can also route voice questions to the appropriate department.

4

Setting up the development environment

This chapter explains how to set up the development environment on IBM AIX, Linux on Power, and IBM i systems to build and deploy an application that connects to Watson Services.

The chapter includes the following topics:

© Copyright IBM Corp. 2018. All rights reserved.

4.1 Setting up the environment on IBM AIX

This section describes how to set up the development environment on IBM AIX, including all prerequisites.

4.1.1 Downloading and installing the curl and Git tools

To download both tools, go to the curl and Git download sites. Curl and Git are useful tools that many developers have in their toolbox. Curl can be used to test Representational State Transfer (REST) application programming interfaces (APIs), and Git is used to manage application code.

Installation prerequisites for the curl and Git tools

To locate the prerequisites for both tools, go to AIX Toolbox for Linux Applicatons.

Installing the packages on AIX

To install the packages, run the command that is shown in Example 4-1.

Example 4-1 Running the rpm command

```
# rpm -ihv RPM_PACKAGE_NAME
```

4.1.2 Installing Node.js

There are many programming languages that you can use to develop your application. The most popular for Watson services are Java, Node.js, and Python. The Watson developer cloud provides software developer kits (SDKs) for these and other programming languages. These SDKs provide an abstraction layer to the REST APIs, making them easier to use. In this example, we choose to develop the application by using Node.js because most of the Watson Sample applications are built with Node.js.

Node.js is a JavaScript run time for developing server-side or command-line applications. The Node Package Manager (NPM) is the package infrastructure for Node.js, and enables developers to share and reuse Node.js packages. Some of these packages provide web frameworks.

The application that is developed for this publication uses the Node.js Express web framework.

In this section, you install Node.js Version 6.12. Ensure that you install the following prerequisites for Node.js:

- `libgcc-4.8.3-1`
- `libstdc++-4.8.3-1`

You can find these packages at the OSS FTP site.

Table 4-1 shows a matrix of the AIX different versions.

Table 4-1 AIX different versions

AIX 6.1	AIX 7.1	AIX 7.2
libgcc-4.8.3-1.aix6.1.ppc.rpm	libgcc-4.8.3-1.aix7.1.ppc.rpm	libgcc-4.8.3-1.aix7.2.ppc.rpm
libstdc++-4.8.3-1.aix6.1.ppc.rpm	libstdc++-4.8.3-1.aix7.1.ppc.rpm	libstdc++-4.8.3-1.aix7.2.ppc.rpm

Note: You must `libgcc` before you install `libstdc`.

To install Node.js Version 6.12, complete the following steps:

1. Download the Node.js binary file from IBM SDK for Node.js Version 6.
2. Upload the file to the AIX server.
3. Add the execution permission to the package so that you can install it, as shown in Example 4-2.

Example 4-2 Running the chmod command to add the execution permission

```
# chmod a+x ibm-6.12.0.0-node-v6.12.0-aix-ppc64.bin
```

4. Run the package as a shell script to install it, as shown in Example 4-3.

Example 4-3 Running the installation by calling the package as a shell script

```
# ./ibm-6.12.0.0-node-v6.12.0-aix-ppc64.bin
```

5. Follow the wizard, and type `back` if you want to change the settings.

Note: The default installation folder is `/ibm/node` under the user home directory. The installation requires approximately 80 MB.

6. After the installation finishes, the completion message appears, as shown in Example 4-4.

Example 4-4 Successful installation message

```
Installation Complete
Congratulations. IBM SDK for Node.js (TM) has been successfully installed to:
    /ibm/node
PRESS <ENTER> TO EXIT THE INSTALLER
```

7. Include the `nodejs` binary path in the general `PATH` variable in one of two ways:
   - Edit `/etc/environment` by using a vi editor, as shown in Example 4-5.

     Example 4-5 Editing the environment variable file to update the PATH variable

     ```
     # vi /etc/environment
     ```

   - Update it temporarily in your current shell, as shown in Example 4-6.

     Example 4-6 Updating the PATH variable temporarily

     ```
     # export PATH=$PATH:/ibm/node/bin
     ```

8. Test the Node.js installation by checking the version, as shown in Example 4-7.

   Example 4-7 Using the node command and npm command to check the version

   ```
   # # node --version
   v6.12.0
   # npm --version
   3.10.10
   ```

4.2 Setting up the environment on Linux on Power

This section describes how to set up the development environment on Linux on Power, including all the prerequisites.

4.2.1 Installing the curl and Git tools on your Linux on Power server

You can install the tools in one of two ways:

- Install them by running yum to pull them from the Linux repository, as shown in Example 4-8.

  Example 4-8 Running the yum command to pull the tools

  ```
  # yum install curl
  # yum install git
  ```

- Separately locate the packages in the Linux DVD media and then install them by running the command that is shown in Example 4-9.

  Example 4-9 Running the rpm command to install the tools

  ```
  # rpm -ihv RPM_PACKAGE_NAME
  ```

4.2.2 Installing Node.js

To install Node.js, complete the following steps:

1. Download theNode.js binary file from IBM SDK for Node.js Version 6.
2. Upload it to the AIX server.

3. Add the execution permission to the package, as shown in Example 4-10.

Example 4-10 Running the chmod command to add the execution permission

```
# chmod a+x ibm-6.12.0.0-node-v6.12.0-linux-ppc64.bin
```

4. Run the installation, as shown in Example 4-11.

Example 4-11 Running the installation by calling the package as a shell script

```
# ./ibm-6.12.0.0-node-v6.12.0-linux-ppc64.bi
```

5. Follow the wizard, and type `back` if you want to change the settings.

Note: The default installation folder is `/root/ibm/node` under the user home directory. The installation requires approximately 60 MB.

6. If you have a successful installation, you see the message that is shown in Example 4-12.

Example 4-12 Successful installation message

```
Installation Complete
Congratulations. IBM SDK for Node.js (TM) has been successfully installed to:
/root/ibm/node
PRESS <ENTER> TO EXIT THE INSTALLER
```

7. Include the `nodejs` binary path in the general `PATH` variable in one of two ways:
   a. Edit `/etc/environment`, as shown in Example 4-13.

Example 4-13 Editing the environment variable file to update the PATH variable

```
# vi /etc/environment
```

   b. Update it temporarily in your current shell, as shown in Example 4-14.

Example 4-14 Updating the PATH variable

```
# export PATH=$PATH:/root/ibm/node/bin
```

8. Test the Node.js installation by checking the version, as shown in Example 4-15.

Example 4-15 Running the node command and npm command to check the version

```
# # node --version
v6.12.0
# npm --version
3.10.10
```

4.3 Setting up the environment on IBM i on Power

The example that we use to deploy Node.js on IBM i is a Node.js web application that connects to IBM Watson services. We also want to get the source code from GitHub and make it run on the system. To do these tasks, you must install Node.js and the Git client on IBM i.

4.3.1 Software prerequisites

To obtain IBM i distribution media installation images, go to IBM Entitled Software Support.

Node.js and Git for IBM i are packaged as part of 5733OPS, which supports only IBM i Version 7.1 and later. Node.js Version 6.x and the Git client are delivered in option 10 and 6 of this new licensed program offerings (LPOs). In addition to installing the new LPOs, some extra software programs are required to use Node.js and Git on IBM i.

Here are the required licensed programs:

- 5770SS1, option 33 for Portable Application Solutions Environment (PASE)
- 5733OPS, option 10 for Node.js V6.x or option 5 for Node.js V4.x
- 5733SC1, option 1 for OpenSSH, OpenSSL, or zlib

Optional software is required if you must add more capabilities to the Node.js environment. For example:

- Python (5733OPS, option 4) is required by NPM to install third-party addons.
- GCC is required if you want to install C/C++-based third-party addons through NPM.

Table 4-2 has a list of options that you can obtain from 5733OPS.

Table 4-2 5733OPS options details

Option names	Software or tool
5733OPS option 1	Node.js V1
5733OPS option 2	Python 3
5733OPS option 4	Python 2
5733OPS option 5	Node.js V4
5733OPS option 6	Git client
5733OPS option 10	Node.js V6
5733OPS option 11	Nginx

4.3.2 Installing the Node.js and Git environment on IBM i

To install the Node.js environment, complete the following steps:

1. Install the 5733OPS option 10 product from the physical DVD media or image files.
2. Apply the current 5733OPS PTF Group to get the current Node.js support based on your IBM i operating system (OS) version for IBM i 7.1 PTF Group SF99123 – level 3 (or higher).

Installation steps for Node.js and Git

Assuming that you have the required media installation images available and mounted to your system, complete the following steps:

1. Log in and run **GO LICPGM**, and then select option 11 on your IBM i system to install the required options, as shown in Figure 4-1.

Figure 4-1 Accessing the Work with Licensed Programs menu

2. Ensure that the IBM i Open Source Solutions *base is included with the required options, which are 6 for Git, and 10 or 5 for Node.js, as shown in Figure 4-2.

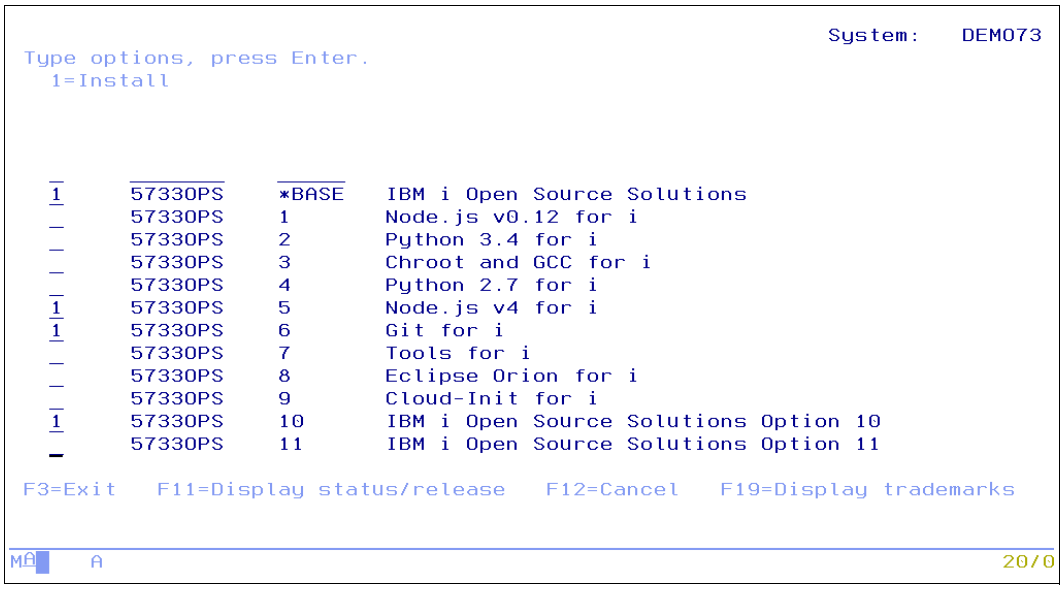

Figure 4-2 Selecting the options to install from 5733OPS

Testing the Node.js and Git installation

If the installation is successful, the directory for Node.js is created under `/QOpenSys/QIBM/ProdData/OPS/Node6`.

To check to see whether Node.js and Git are installed and ready to use, create a Qshell session and run the commands that are shown in Figure 4-3 and Figure 4-4 on page 79.

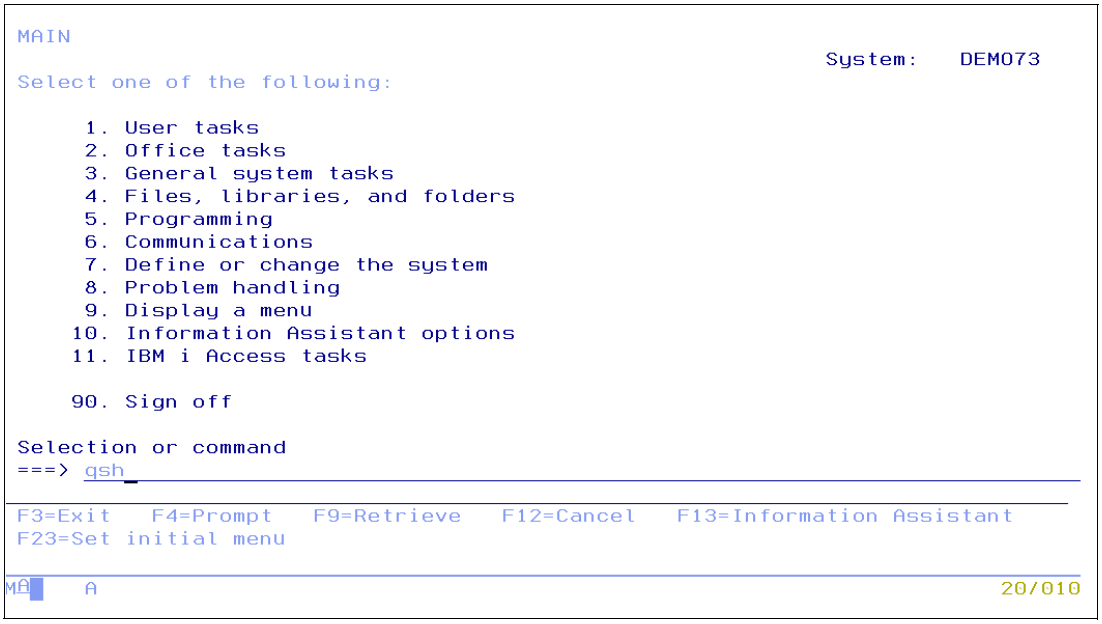

Figure 4-3 Accessing Qshell from the main menu

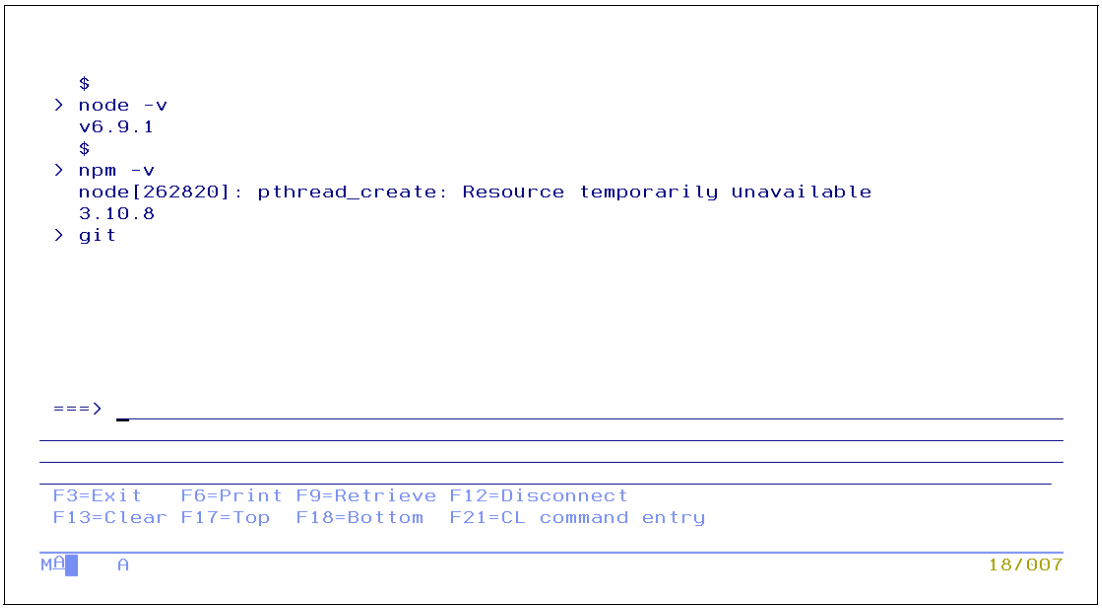

Figure 4-4 Check whether Node.js, NPM, and Git are installed and ready to use

4.4 Continuous integration and continuous delivery

A continuous integration and continuous delivery approach enables your developers and testers to code, release, and test code incrementally. This approach enables you to detect errors early, and release functions in small parts. You need tools that enable you automate the process of integration of code updates, packaging, and delivery. There are several options that are available. For this book, we chose Jenkins as the continuous integration tool for UNIX and Linux because it offers a fully on-premises option.

This section describes the functions of the Jenkins automation tool, which is an open source continuous integration platform that runs under a Java web servlet container (such as the Liberty profile server).

Continuous integration is the practice of frequently building and testing software projects during development. The aim of this process is to discover defects and regressions early by automating the process of running unit and integration tests. These automated build and test cycles typically happen on a regular schedule (such as every night) or even after each change is delivered.

Jenkins can integrate with many frameworks and toolkits by using its extensive library of available plug-ins, which includes the following items:

- Source code management and version control platforms, including CVS, Subversion, Git, Mercurial, Perforce, IBM ClearCase®, and IBM Rational Team Concert™
- Build automation tools, such as Apache Ant and Maven, and also standard OS batch and script files
- Testing frameworks, such as JUnit and TestNG
- RSS, email, and instant messenger clients for reporting results in real time
- Artifact uploaders and deployers for several integration platforms

4.4.1 Jenkins automation tool installation

This section describes the installation of Jenkins into various OSs on a Power platform.

Jenkins installation on IBM AIX

Installing Jenkins on IBM AIX requires Java8 to be installed on the AIX server.

Complete the following steps:

1. Make sure that you installed Java8 on to your AIX server.
2. Select Java SE Version 8 and download the 64-bit version.
3. Upload it to the AIX server (use bin mode if you are using FTP).
4. Run **`smitty install`** to install it.
5. After Java8 is installed, add it to your `PATH` environment.
6. Make sure that AIX resolves the new Java version by running the command that is shown in Example 4-16.

Example 4-16 Checking the Java version

```
# java -version
Java(TM) SE Runtime Environment (build 8.0.5.5 - pap6480sr5fp5-20171114_01(SR5
FP5))
IBM J9 VM (build 2.9, JRE 1.8.0 AIX ppc64-64 Compressed References
20171102_369060 (JIT enabled, AOT enabled)
OpenJ9   - 7ade437
OMR      - 1b656cb
IBM      - 59c3d96)
JCL - 20171113_01 based on Oracle jdk8u151-b12
```

7. Download the generic Jenkins Java package.
8. Upload it to AIX server (use bin mode if you are using FTP).
9. Run the Jenkins setup by running the command that is shown in Example 4-17.

Example 4-17 Jenkins setup

```
# java -jar jenkins.war
Running from: /jenkins.war
webroot: $user.home/.jenkins
Dec 14, 2017 8:51:46 AM Main deleteWinstoneTempContents
WARNING: Failed to delete the temporary Winstone file /tmp/winstone/jenkins.war
Dec 14, 2017 8:51:47 AM org.eclipse.jetty.util.log.Log initialized
INFO: Logging initialized @2295ms to org.eclipse.jetty.util.log.JavaUtilLog
Dec 14, 2017 8:51:47 AM winstone.Logger logInternal
INFO: Beginning extraction from war file
Dec 14, 2017 8:51:47 AM org.eclipse.jetty.server.handler.ContextHandler
setConte
```

10. Try to access the AIX server on port 8080, as shown in Example 4-18.

Example 4-18 Accessing the AIX server

```
http://ip_address:8080
```

Jenkins installation on Linux on Power

This section describes the Jenkins installation on Linux on Power.

Complete the following steps:

1. To download Jenkins from the Red Hat repository, run the command that is shown in Example 4-19.

Example 4-19 Downloading Jenkins

```
# wget -O /etc/yum.repos.d/jenkins.repo
http://pkj.jenkins-ci.org/redhat/jenkins.repo
```

2. Import the verification key by using the package manager RPM, as shown in Example 4-20.

Example 4-20 Importing the verification key

```
# rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
```

3. Run the **yum** command to install the package, as shown in Example 4-21.

Example 4-21 Installing the Jenkins package

```
# yum install jenkins
Loaded plugins: langpacks, search-disabled-repos
jenkins
| 2.9 kB  00:00:00
rhel-7-for-power-rpms
| 2.0 kB  00:00:00
jenkins/primary_db
104 kB  00:00:00
Resolving Dependencies
--> Running transaction check
---> Package jenkins.noarch 0:2.94-1.1 will be installed
--> Finished Dependency Resolution

Is this ok [y/d/N]: y
Downloading packages:
jenkins-2.94-1.1.noarch.rpm
|  71 MB  00:00:06
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
 Installing : jenkins-2.94-1.1.noarch
1/1
 Verifying : jenkins-2.94-1.1.noarch
1/1
Installed:
 jenkins.noarch 0:2.94-1.1

Complete!
```

Alternatively, you can download the package directly.

4. Start Jenkins as a service, as shown in Example 4-22.

Example 4-22 Starting Jenkins as a service

```
# systemctl start jenkins.service
```

5. Check the service status by running the systemctl command, as shown in Example 4-23.

Example 4-23 Checking the service status

```
# systemctl status jenkins.service
jenkins.service - LSB: Jenkins Automation Server
   Loaded: loaded (/etc/rc.d/init.d/jenkins; bad; vendor preset: disabled)
   Active: active (running) since Wed 2017-12-13 16:17:33 CST; 2min 29s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 22269 ExecStart=/etc/rc.d/init.d/jenkins start (code=exited,
status=0/SUCCESS)
Dec 13 16:17:32 powervc systemd[1]: Starting LSB: Jenkins Automation Server...
Dec 13 16:17:32 powervc runuser[22274]: pam_unix(runuser:session): session
opened for user jenkins by (uid=0)
Dec 13 16:17:33 powervc runuser[22274]: pam_unix(runuser:session): session
closed for user jenkins
Dec 13 16:17:33 powervc jenkins[22269]: Starting Jenkins [  OK  ]
Dec 13 16:17:33 powervc systemd[1]: Started LSB: Jenkins Automation Server.
```

6. Access the Jenkins web interface on port 8080, as shown in Example 4-24.

Example 4-24 Accessing the Jenkins web interface

```
http://ip_address:8080
```

5

Deploying a sample Node.js application that integrates with IBM Watson services

This chapter describes how to deploy a sample Node.js application that integrates with Watson services on Power Systems servers.

The chapter includes the following topics:

© Copyright IBM Corp. 2018. All rights reserved.

5.1 Signing up for an account at IBM Cloud

Register with IBM Cloud by providing a valid, unique email address. Your email address acts as your user name for IBM Cloud, and you must also provide a valid password. When you sign up to IBM Cloud, you are prompted for your demographic information (such as your name and company). You receive an email containing a link to confirm that your email account is valid. Click that link to activate your account.

You need an account to create a Watson Personality Insights service on IBM Cloud. To create an account at IBM Cloud, complete the following steps:

1. Open the IBM Cloud console in a web browser.
2. Click **Create a free account** and follow the steps to create an account.

5.1.1 Logging in to IBM Cloud

Log in to IBM Cloud by completing the following steps:

1. Open the IBM Cloud console in a web browser.
2. On the IBM Cloud home page, click **Log in**.
3. Enter the email address that you used to register for IBM Cloud and click **Continue**, as shown in Figure 5-1. Enter your password when you are prompted.

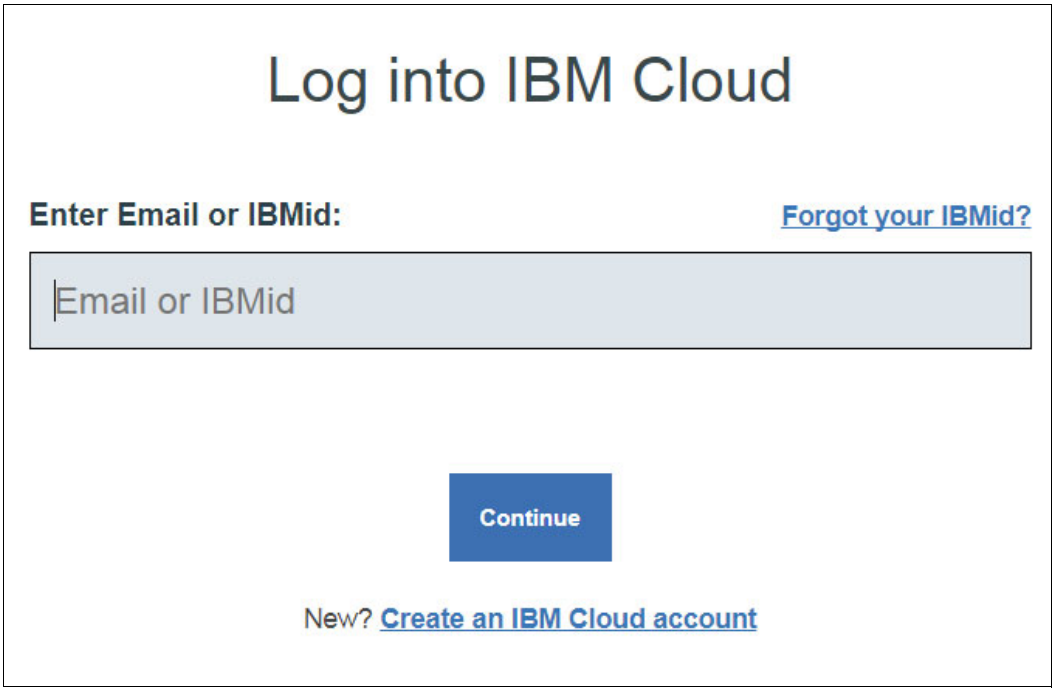

Figure 5-1 Logging in to IBM Cloud

5.2 Creating a Watson Personality Insights service on IBM Cloud

The Watson Personality Insights service enables deeper understanding of the personality of individuals.

You interact with the service through a Representational State Transfer (REST) application programming interface (API). Input the service text (such as email, blogs, tweets, or other communication) that is written by one individual, and the service outputs the personality analysis of that individual.

To create a Watson Personality Insights service on IBM Cloud, complete the following steps:

1. From the Dashboard, click **Create resource**, as shown in Figure 5-2.

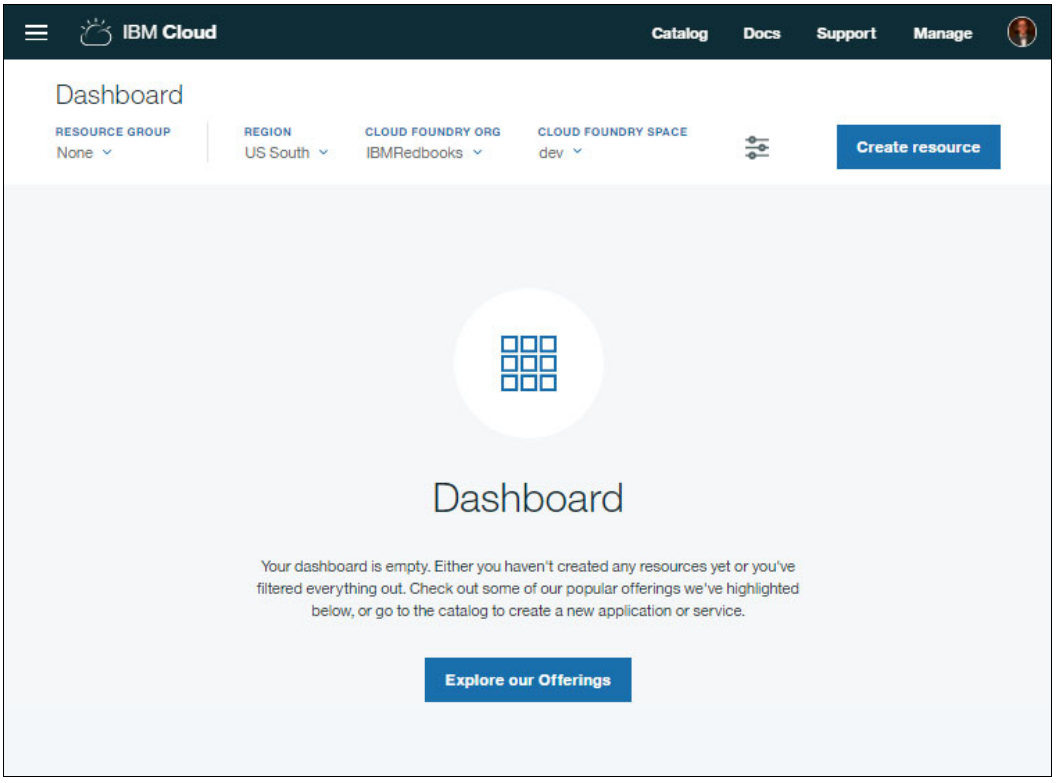

Figure 5-2 Dashboard

2. From Catalog window (Figure 5-3), find and click **Personality Insights** to create the service.

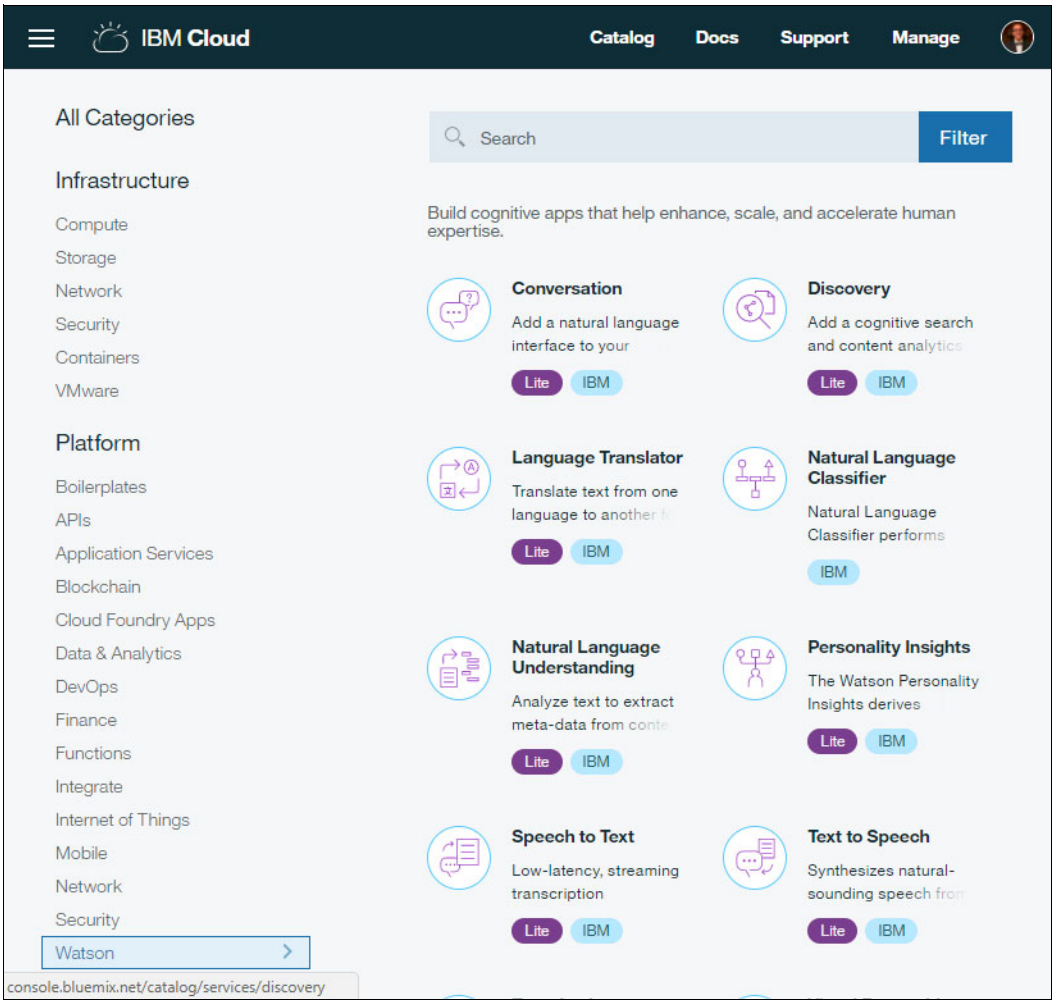

Figure 5-3 Catalog

Note: Services are extensions to the cloud environment that IBM Cloud hosts and manages. The predefined services that are provided by IBM Cloud include NoSQL and SQL databases, the ability to send push notifications to your mobile app, and language translation by using Watson. The infrastructure for services is managed by IBM Cloud, and your app must focus on the provided endpoint only.

You can add services from the IBM Cloud catalog. Services provide a predefined endpoint that you can access from your application to use the predefined functions of that service.

3. Keep the default values in the Watson Personality Insights creation dialog (Figure 5-4), and click **Create**. This action creates the Watson Personality Insights service on IBM Cloud.

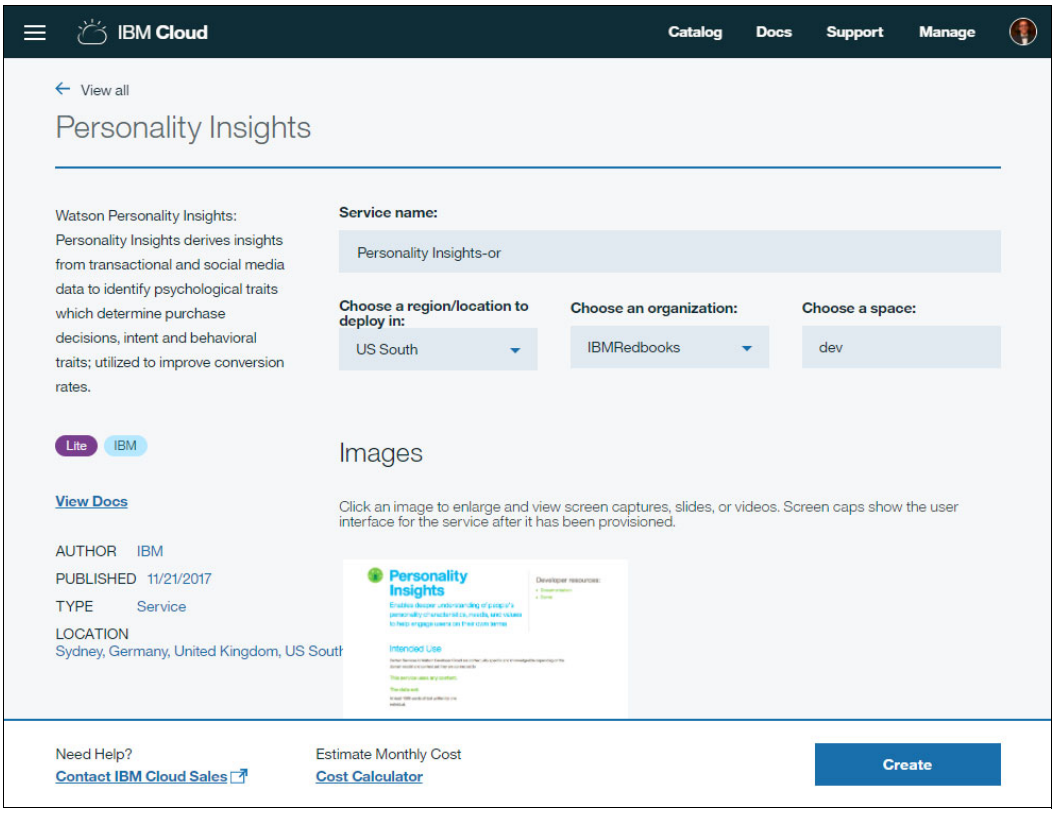

Figure 5-4 Watson Personality Insights creation dialog

4. To get your credentials for the Watson Personality Insights service, complete the following steps. You must configure the sample application with these credentials.

   a. Click **Service credentials** from the left navigation bar in Service Details (Figure 5-5).

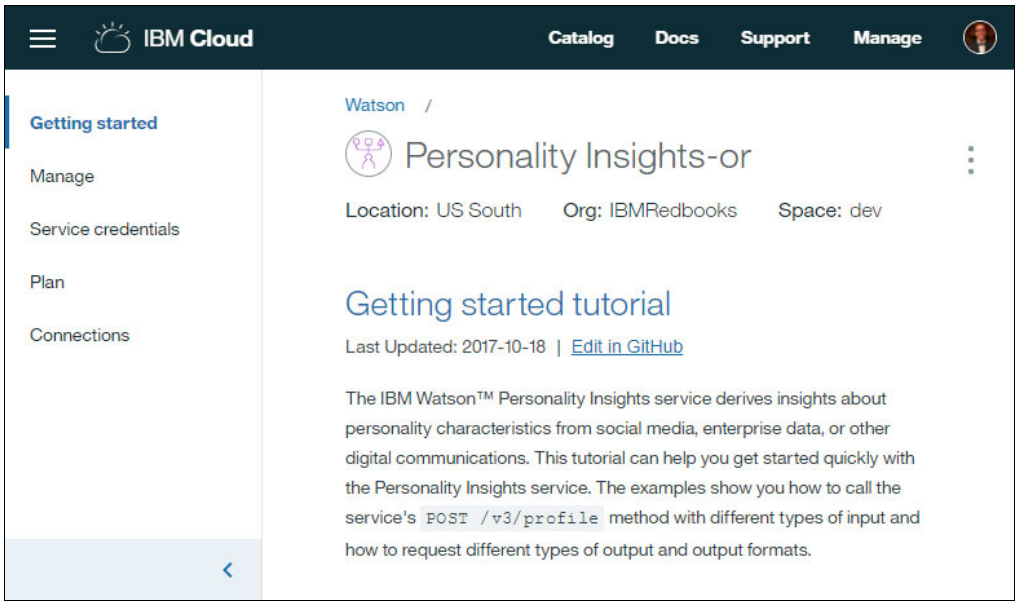

Figure 5-5 Watson Personality Insights service details

   b. Click **New credential** (Figure 5-6).

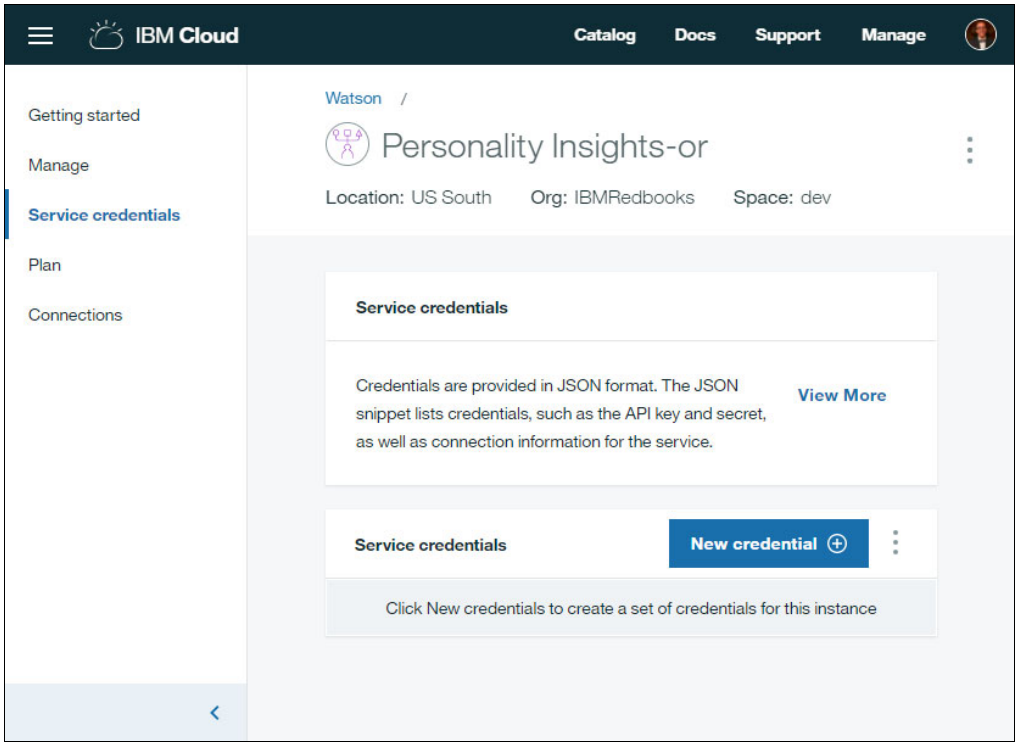

Figure 5-6 Service credentials

c. A dialog window opens where you add a credential (Figure 5-7). Click **Add**.

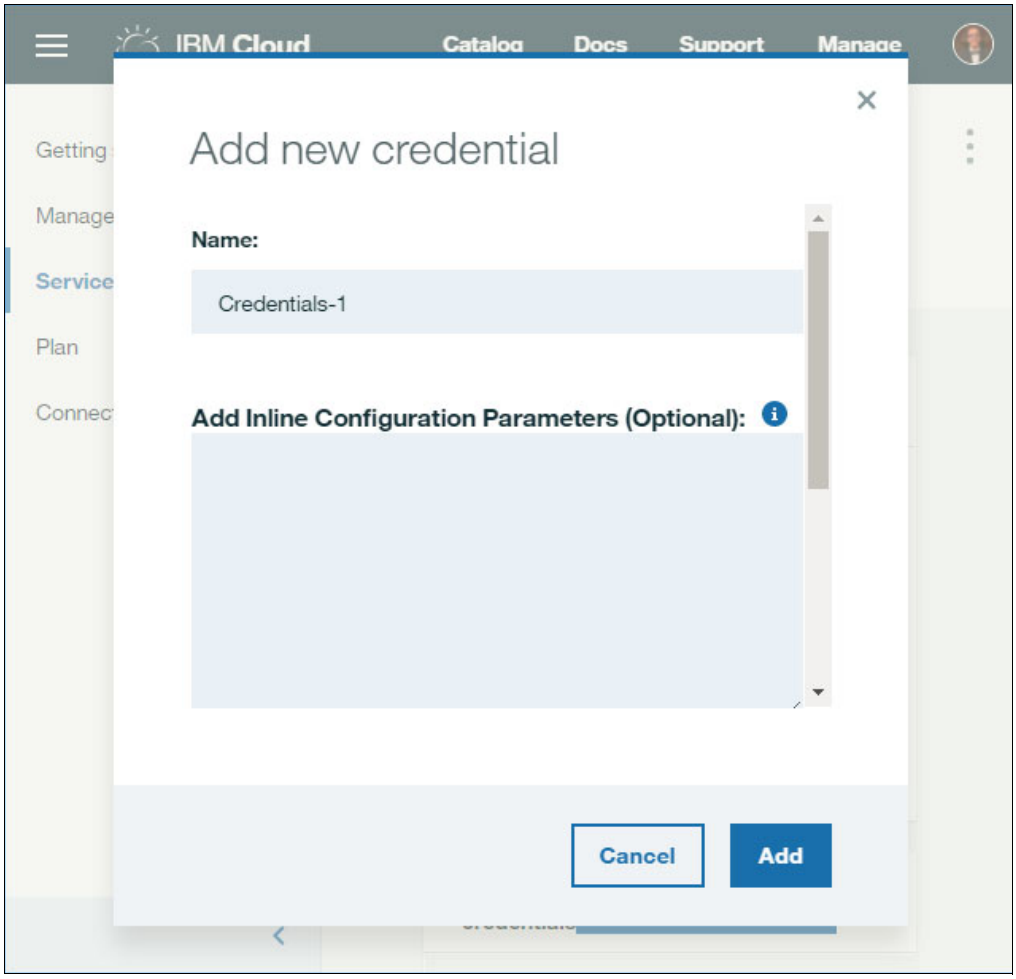

Figure 5-7 Add new credential window

IBM Cloud adds the credentials for the service, as shown in Figure 5-8.

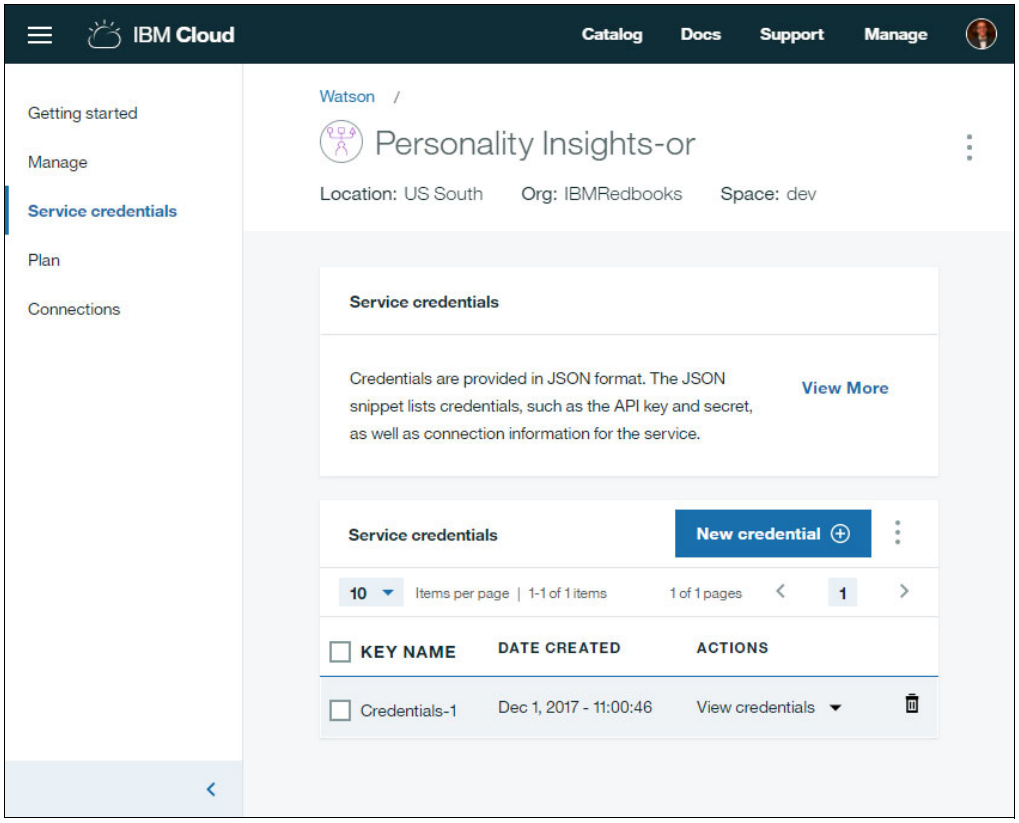

Figure 5-8 Service details with Watson Personality Insights credentials added

d. Click **View credentials**.

e. Your Watson Personality Insights service credentials are displayed (Figure 5-9). Copy the credentials (user name and password) because you need them to configure the sample Node.js application on the Power Systems server to use the credentials.

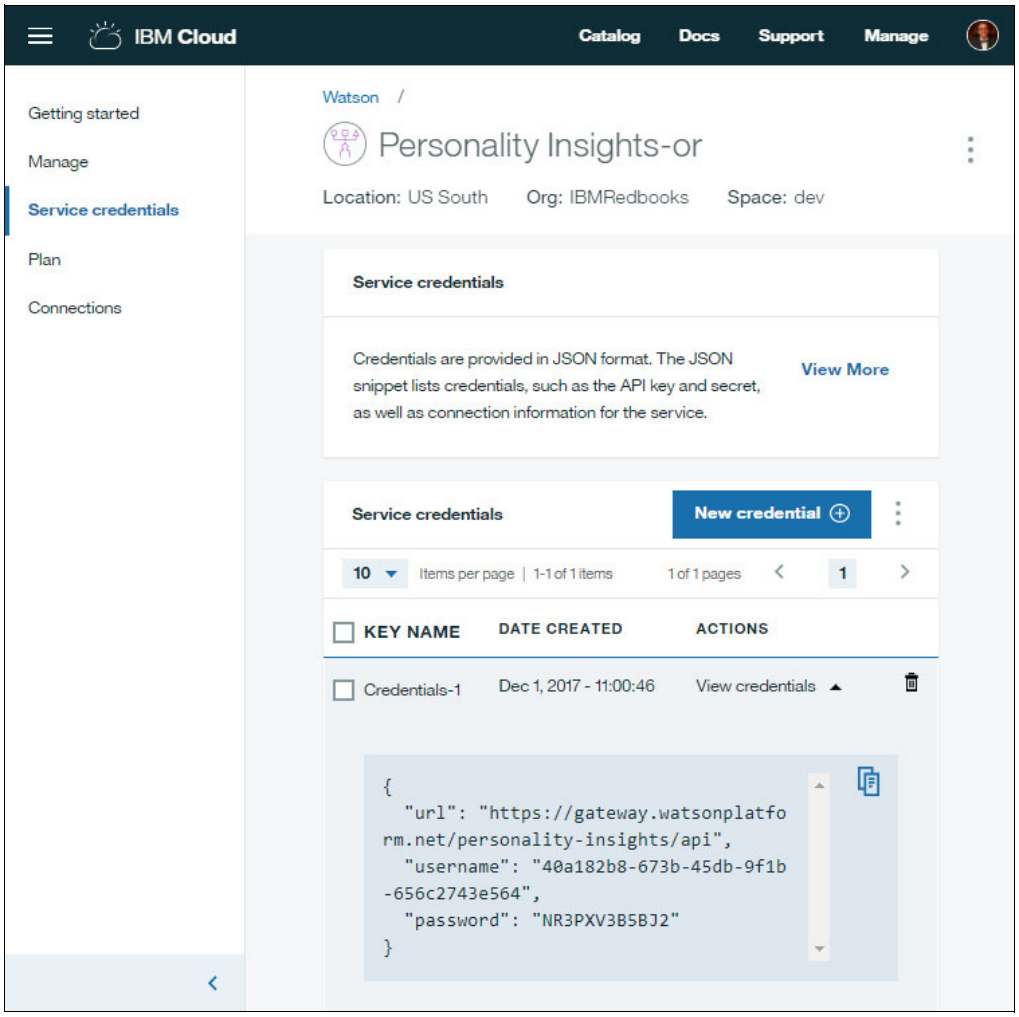

Figure 5-9 Watson Personality Insights credentials

5.3 Cloning and deploying the Watson Personality Insights Node.js demonstration app on AIX and Linux on Power

In this section, you complete the following steps on your Power Systems server to clone and deploy a sample Node.js application that integrates with Watson APIs:

1. Clone a sample Node.js application for the Watson Personality Insights Service.
2. Edit the credentials to point to the Watson Personality Insights service that you created on IBM Cloud.
3. Deploy the sample application.

5.3.1 Cloning a sample Node.js application for the IBM Watson Personality Insights Service

The demonstration application that you are going to clone is the IBM demonstration app for Watson Personality Insights. It is available at IBM Bluemix.

To clone the application to your Power Systems server, complete the following steps:

1. The actions that you take depend on the operating system (OS) that you are using:
   - If you are using Linux on Power, create a directory for `ibm-redbooks` by running the following command and then go to step 2 on page 94:

     ```
     mkdir ibm-redbooks
     ```

   - If you are using AIX, create a file system for `ibm-redbooks` by completing the following steps:

     i. Run the command in Example 5-1 to run **`smitty`**.

     Example 5-1 Running the smitty tool

     ```
     smitty jfs2
     ```

     Note: The **`smitty`** command starts the System Management Interface Tool (SMIT). SMIT is an interactive interface application that simplifies system management tasks. The **`smitty`** command displays a hierarchy of menus that can lead to interactive dialogues. SMIT builds and runs commands as directed by the user. Because SMIT runs commands, you need the authority to run the commands that SMIT runs.

     The Enhanced Journaled File System (JFS2) is the file system that is used in AIX.

     ii. Choose `Add an Enhanced Journaled File System`, as shown in Figure 5-10.

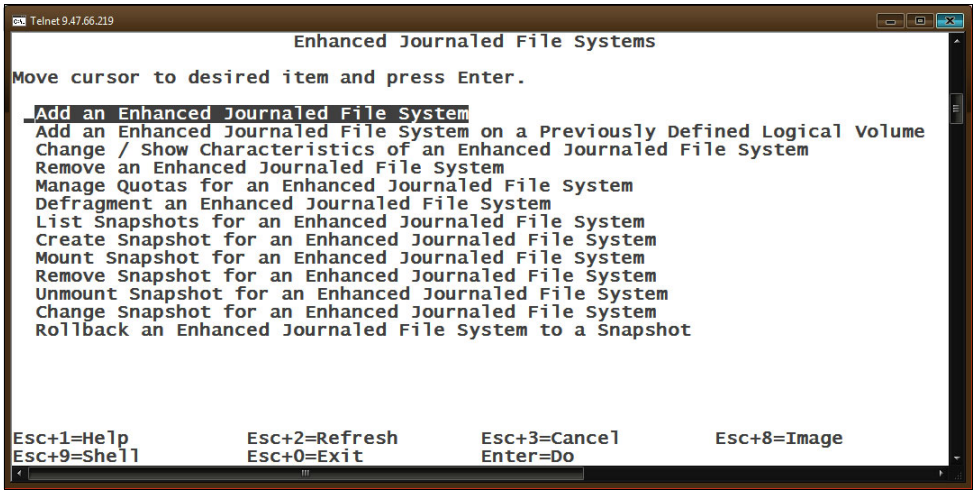

Figure 5-10 Smitty menu

iii. Select `datavg` or any other user volume group (VG), as shown in Figure 5-11. This VG will be the container of the file system.

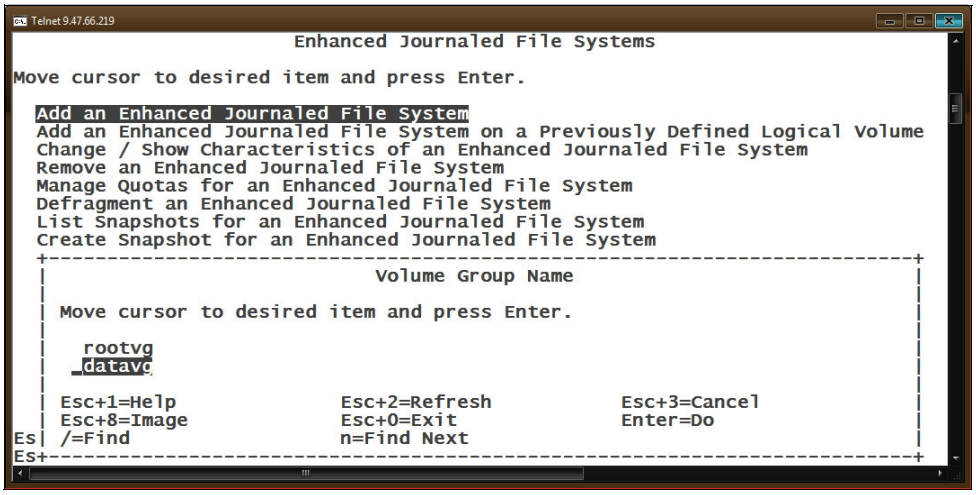

Figure 5-11 Selecting the volume group

iv. Make the size of the file system at least 200 MB, and input `ibm-redbooks` as a mount point, as shown in Figure 5-12.

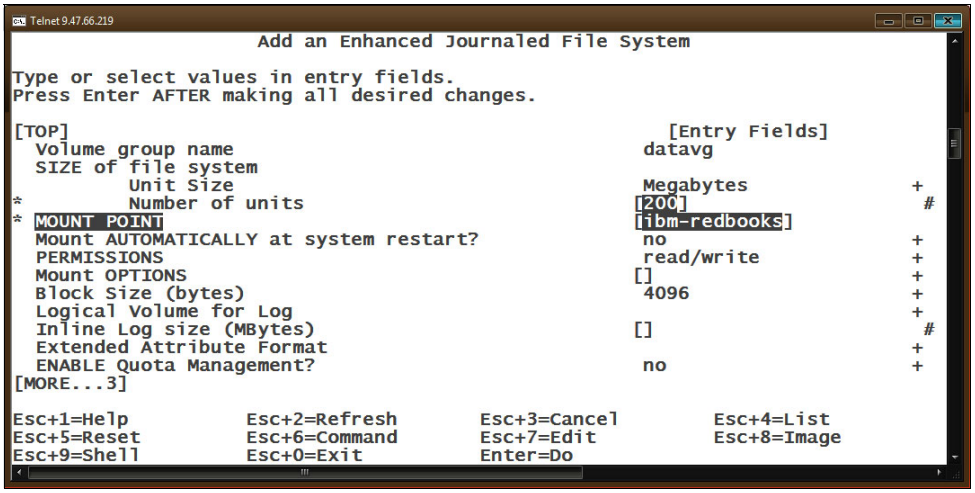

Figure 5-12 File system configuration

You should receive a success message similar to Figure 5-13.

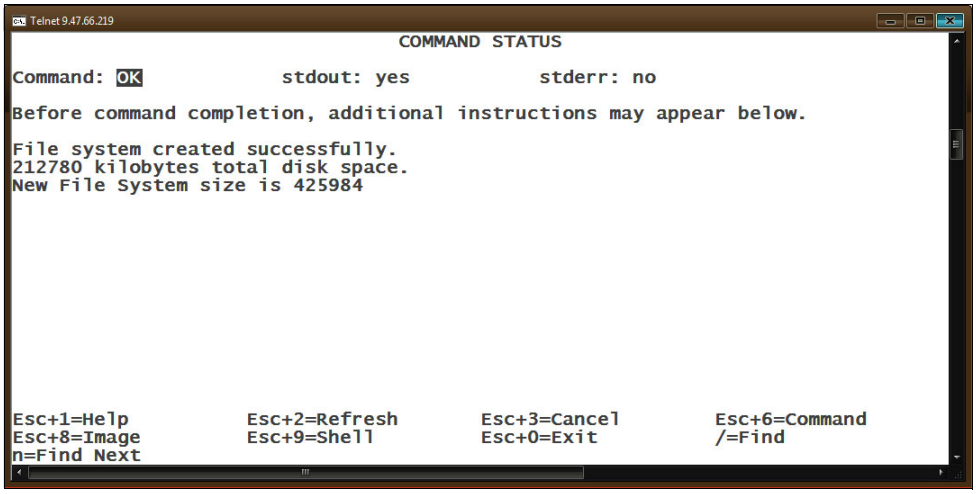

Figure 5-13 File system creation confirmation

v. Close SMIT.

vi. Mount the file system by running the following command:

```
mount /ibm-redbooks
```

2. Go to `ibm-redbooks` by running the following command:

```
cd ibm-redbooks
```

3. Clone the Watson Personality Insights Node.js demonstration application by running the following command:

```
git clone https://github.com/watson-developer-cloud/personality-insights-nodejs
```

You should see a successful message similar to the following one:

```
# git clone
https://github.com/watson-developer-cloud/personality-insights-nodejs
Cloning into 'personality-insights-nodejs'...
remote: Counting objects: 1938, done.
remote: Total 1938 (delta 0), reused 0 (delta 0), pack-reused 1938
Receiving objects: 100% (1938/1938), 7.63 MiB | 5.56 MiB/s, done.
Resolving deltas: 100% (1117/1117), done.
Checking connectivity... done.
#
```

4. Go to the directory of the application by running the following command:

```
cd personality-insights-nodejs
```

5.3.2 Editing the credentials

In this section, you edit the credentials to point to the Watson Personality Insights service that you created earlier on IBM Cloud.

To edit the credentials, complete the following steps:

1. Create an `.env` file by copying the sample `.env.example` file by using the following command:

   ```
   cp .env.example .env
   ```

2. Edit the `PERSONALITY_INSIGHTS_USERNAME` and `PERSONALITY_INSIGHTS_PASSWORD` fields in the `.env` file and add the credentials of the Watson Personality Insights service that you copied in 5.2, “Creating a Watson Personality Insights service on IBM Cloud” on page 85 by running the following command:

   ```
   vi .env
   ```

   Note: The `.env` file contains the credentials of the services that are used by the application. This application uses Watson Personality Insights and Twitter integration.

   The `.env` file should be similar to Example 5-2.

   Example 5-2 The modified .env file

   ```
   # service credentials
   PERSONALITY_INSIGHTS_USERNAME=1167fdb3-2f97-4635-84c2-0b6fb0b6dffe
   PERSONALITY_INSIGHTS_PASSWORD=PVt1iwBjeOVJ

   # twitter credentials
   # see https://apps.twitter.com/
   TWITTER_CONSUMER_KEY=<twitter-consumer-key>
   TWITTER_CONSUMER_SECRET=<twitter-consumer-secret>
   ```

   Note: `TWITTER_CONSUMER_KEY` and `TWITTER_CONSUMER_SECRET` are not used because this exercise focuses on integrating the sample application with Watson services.

5.3.3 Deploying the sample application

In this section, you deploy the sample application by completing the following steps:

1. Before you install the app, you must increase the security limits to unlimited values. Security limits define different process resource limits, including CPU, threads, and memory.

   To increase the security limits, complete the following steps:

   a. Run the command in Example 5-3 to edit the security limits for AIX, and the command in Example 5-4 to edit it for Linux on Power.

   Example 5-3 Editing the security limits for AIX

   ```
   vi /etc/security/limits
   ```

   Example 5-4 Editing the security limits for Linux on Power

   ```
   vi /etc/security/limits.conf
   ```

b. Edit the file to have values of `-1` for unlimited resources in the `default:` section, as shown in Figure 5-14.

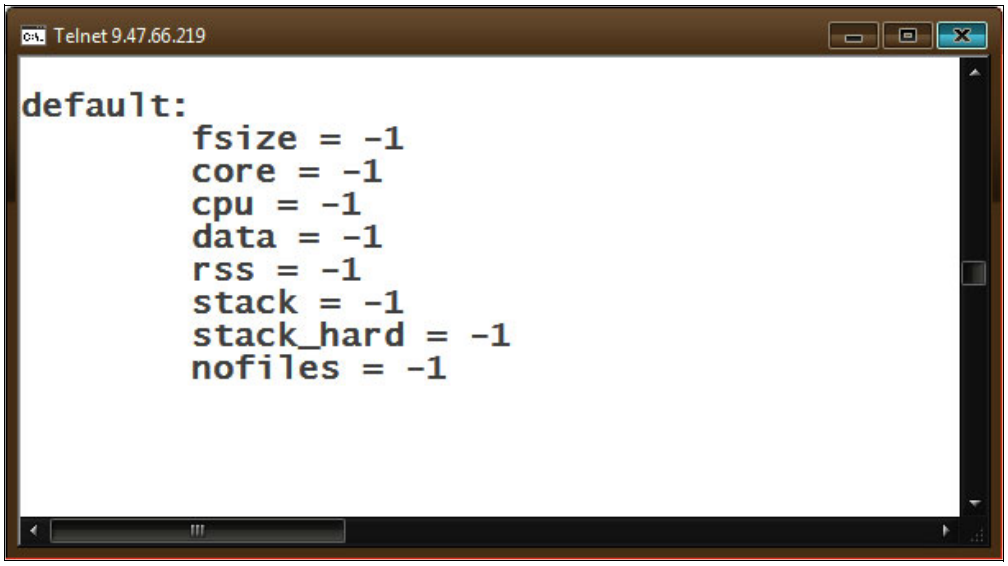

Figure 5-14 Modified security limits file

c. Log out. and then log in to your session by running the following command:

```
login
```

2. Go to the directory of the application by running the following command:

```
cd ibm-redbooks/personality-insights-nodejs
```

3. Install all the required dependencies of the application by running the following command:

```
npm install
```

Note: Node Package Manager (NPM) is the default package manager for the JavaScript runtime environment Node.js.

NPM provides two main functions:

- Online repositories for Node.js packages/modules.
- A command-line utility to install Node.js packages, and perform version management and dependency management of Node.js packages.

NPM accesses the `package.json` file to perform tasks such as registering the application by using the name field in the `package.json` file, and downloading the dependencies in the `package.json` file from the Node.js online repository with the specified versions.

Figure 5-15 shows a snippet of `package.json` for the sample application. Note the required dependencies for the application. By running **`npm install`**, NPM installs all the required dependencies.

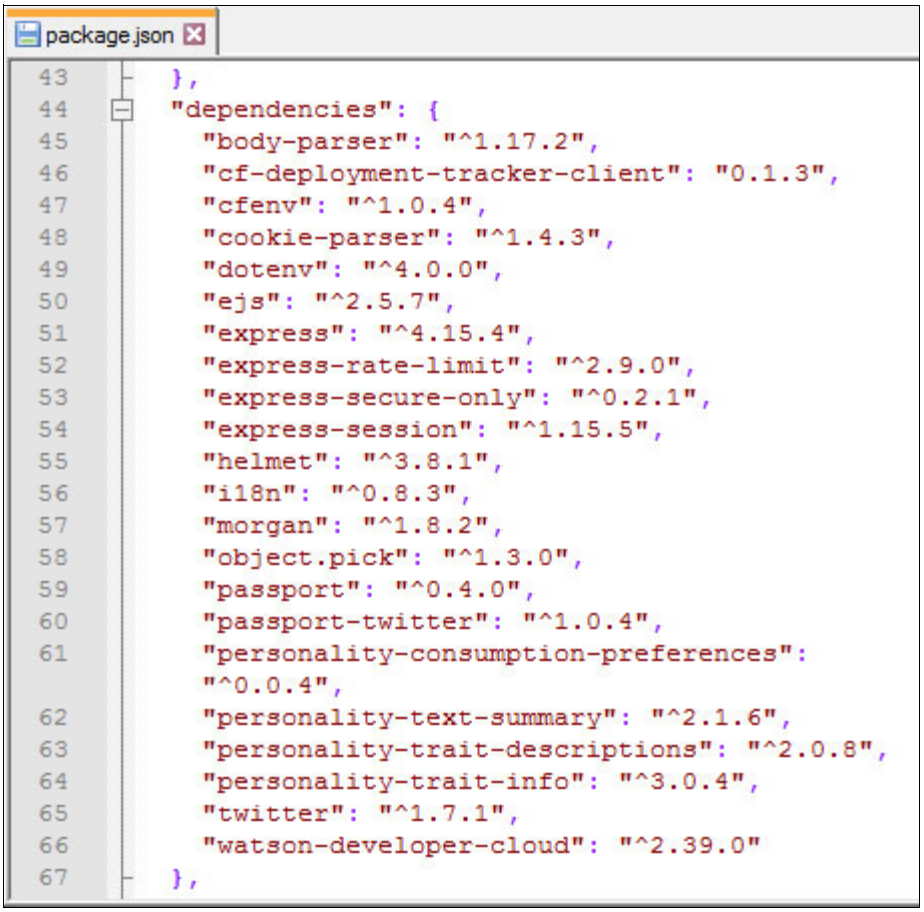

Figure 5-15 The package.json file of the sample application

4. Change the application port to 8080 or another port of your choosing by running the following command:

   ```
   export PORT=8080
   ```

5. Run the application in the background as a service by running the following command and pressing Enter:

   ```
   nohup npm start &
   ```

> **Note:** The **`nohup`** command runs the command as a service. The **`&`** runs the command in the background even after logging off.
>
> The command that you specify **`nohup`** to run is **`node server`**.
>
> When you run **`npm start`**, NPM runs the command in the `scripts.start` field in `package.json`, which is **`node server.js`**. The **`node server.js`** command runs the `server.js` file, which is the starting point (entry `.js` file) for this application.

5.4 Cloning and deploying the Watson Personality Insights Node.js demonstration application on IBM i

Deploying and running the Node.js application on IBM i is slightly different from AIX and Linux. This section describes how to get the source code of the example Node.js application from the GitHub repository and run the application on IBM i environment.

Before completing the steps in this section, make sure that you have your IBM i environment set up correctly to support Node.js and Git, as described in 4.3, “Setting up the environment on IBM i on Power” on page 76.

5.4.1 Cloning the example application by using Git

GitHub has the source code for the application. You can clone the code and download it in to your IBM i machine by using Git client through Qshell terminal by completing the following steps:

1. Open the Qshell by running one of the following commands (Figure 5-16):
   - **CALL QP2TERM**
   - **QSH**

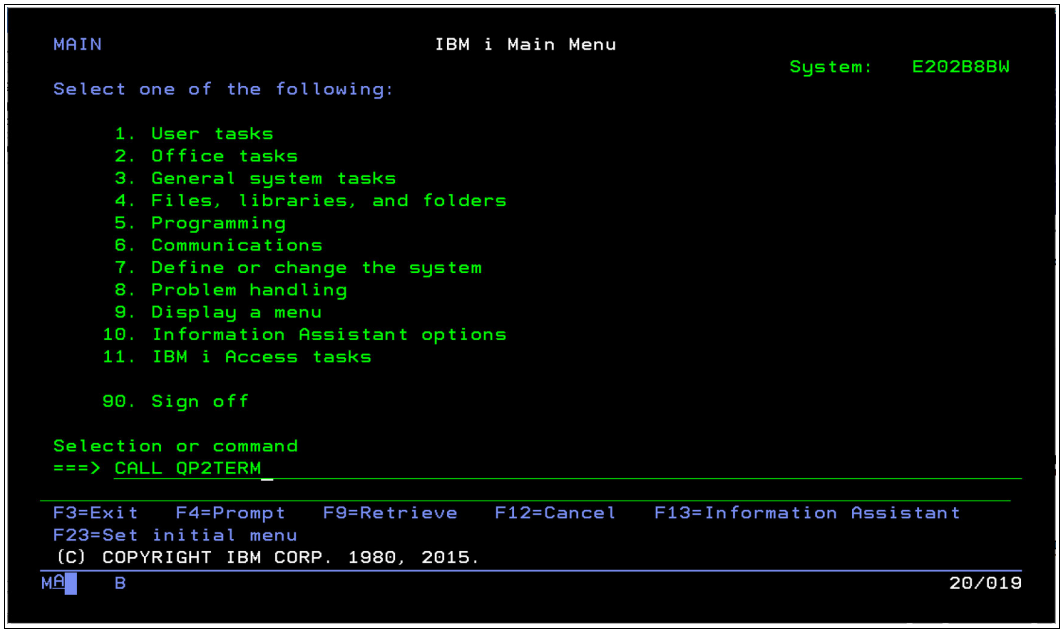

Figure 5-16 CALL QP2TERM to open the Qshell terminal

If you are interested in using a more modern shell interface, it is possible with the current version of Access Client Solutions. Install your favorite shell on your PC (for Windows, an example is PuTTY, and for a Macintosh, you have bash installed). Click the **SSH Terminal** link to start a more modern shell interface (Figure 5-17).

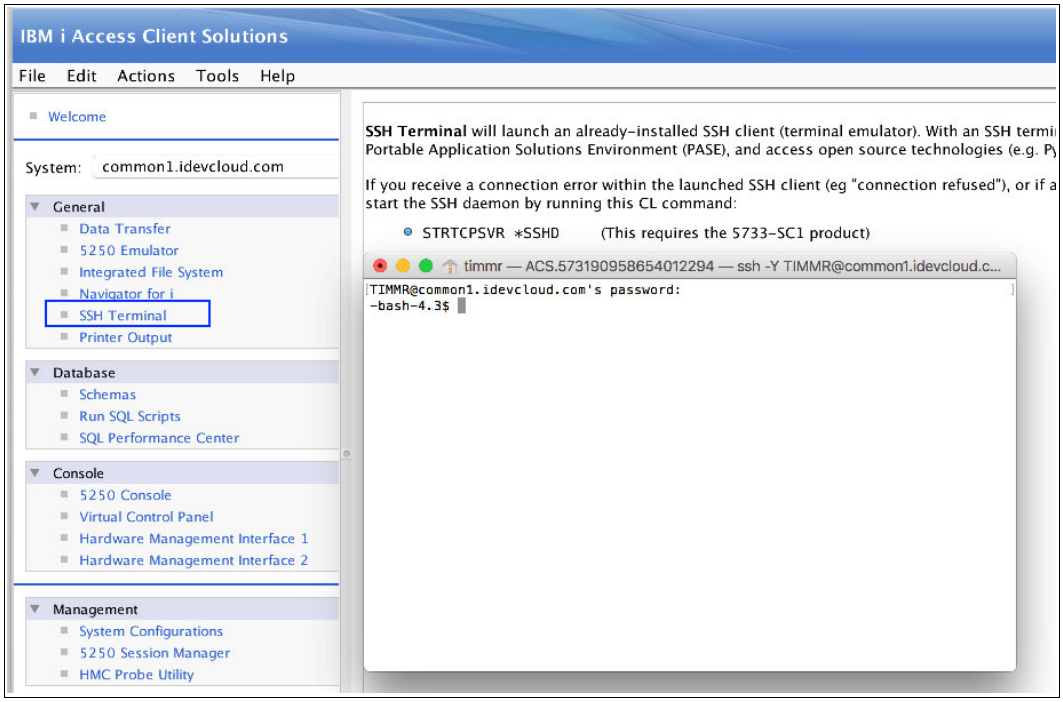

Figure 5-17 Access Client Solutions dialog

For more information, see Access Client Solutions.

2. Change to the directory in which you want to store the code, and run the following command (Figure 5-18):

```
git clone https://github.com/watson-developer-cloud/personality-insights-nodejs
```

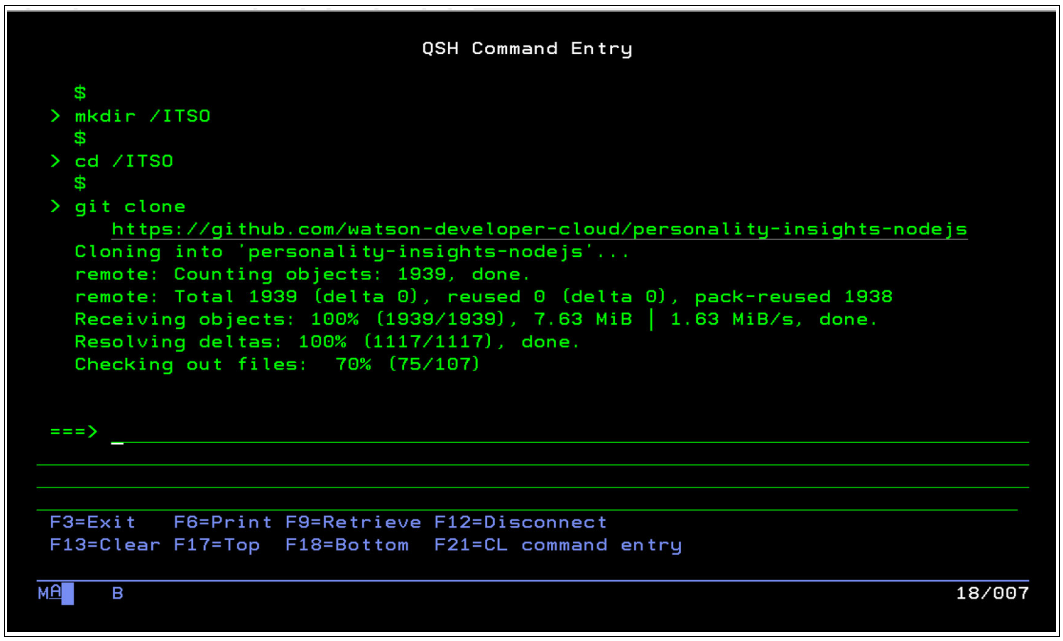

Figure 5-18 Using Git to clone the source code

Note: Check the contents of the server.js file to make sure that it is readable. If the file is not readable, this indicates that there is an encoding problem that happened while cloning the file. To solve this problem, complete the following steps:

1. Go to the following Uniform Resource Locator (URL) in your browser:

   `https://github.com/watson-developer-cloud/personality-insights-nodejs`

2. Click **Clone or download**, and then click **Download ZIP**.
3. Transfer the downloaded compressed file to your IBM i server.
4. Decompress the folder.

5.4.2 Providing the correct credentials to access Watson APIs

In order for the application to run, it must be configured with the correct credentials to access the Watson services. For more information about creating the necessary Watson services and obtaining the API credentials, see 5.2, “Creating a Watson Personality Insights service on IBM Cloud” on page 85.

To configure the credentials for the application, complete the following steps:

1. From the Qshell, go to the application source code directory by running the following command:

   ```
   cd /ITSO/personality-insights-nodejs
   ```

2. Create an `.env` file by copying the sample `.env.example` file by running the following command:

   ```
   cp .env.example .env
   ```

3. Press F3 to exit Qshell.
4. Change the `PERSONALITY_INSIGHTS_USERNAME` and `PERSONALITY_INSIGHTS_PASSWORD` values in the `.env` file by using the credentials of the Watson Personality Insights service that you created in 5.2, “Creating a Watson Personality Insights service on IBM Cloud” on page 85 by running the following command:

   ```
   EDTF '/ITSO/personality-insights-nodejs/.env'
   ```

The content should look similar to Figure 5-19.

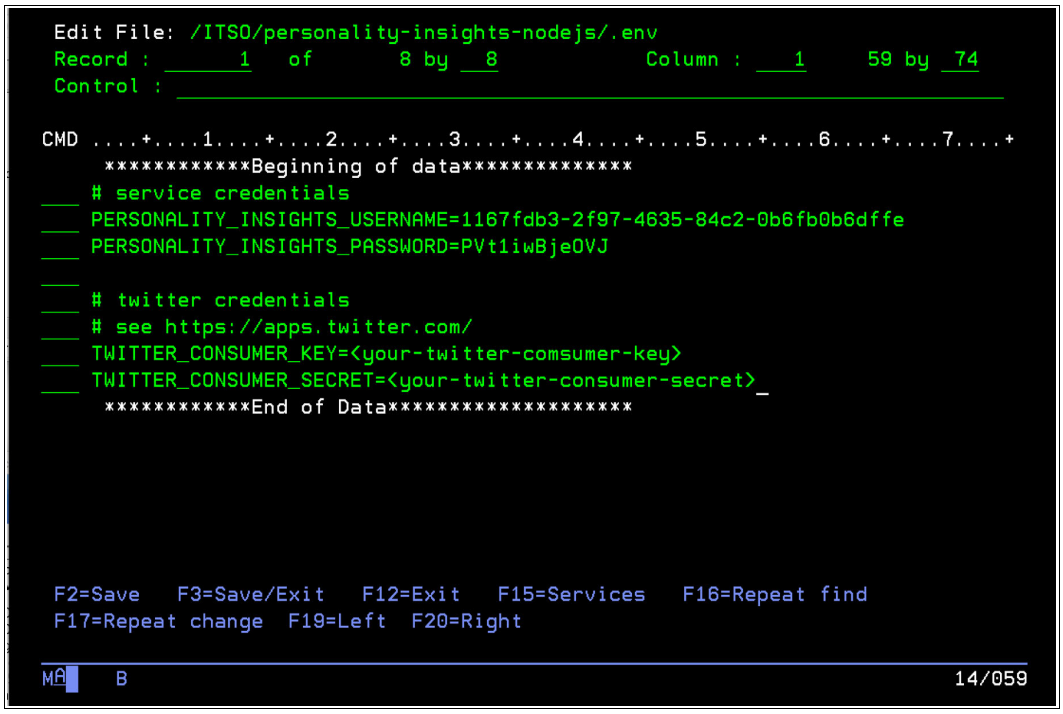

Figure 5-19 Environment file example

Note: You can provide your Twitter consumer key to explore more features of the application.

5. Press F3 to save and exit.

5.4.3 Running the application

To start the application by using Qshell, complete the following steps:

1. Start a Qshell session, and then change directory to the application directory by running the following command:

   ```
   cd /ITSO/personality-insights-nodejs
   ```

   Note: There is a thread limit in Qshell that sometimes generates errors such as `qsh: 001-0078 Process ended by signal 5`. To avoid those errors, before starting a Qshell, add the environment variable `QIBM_MULTI_THREADED` with a value of `Y`, by running the following command:

   ```
   ADDENVVAR ENVVAR(QIBM_MULTI_THREADED) VALUE(Y)
   ```

2. Install the required libraries and modules for the application by using the following command (Figure 5-20):

```
npm install
```

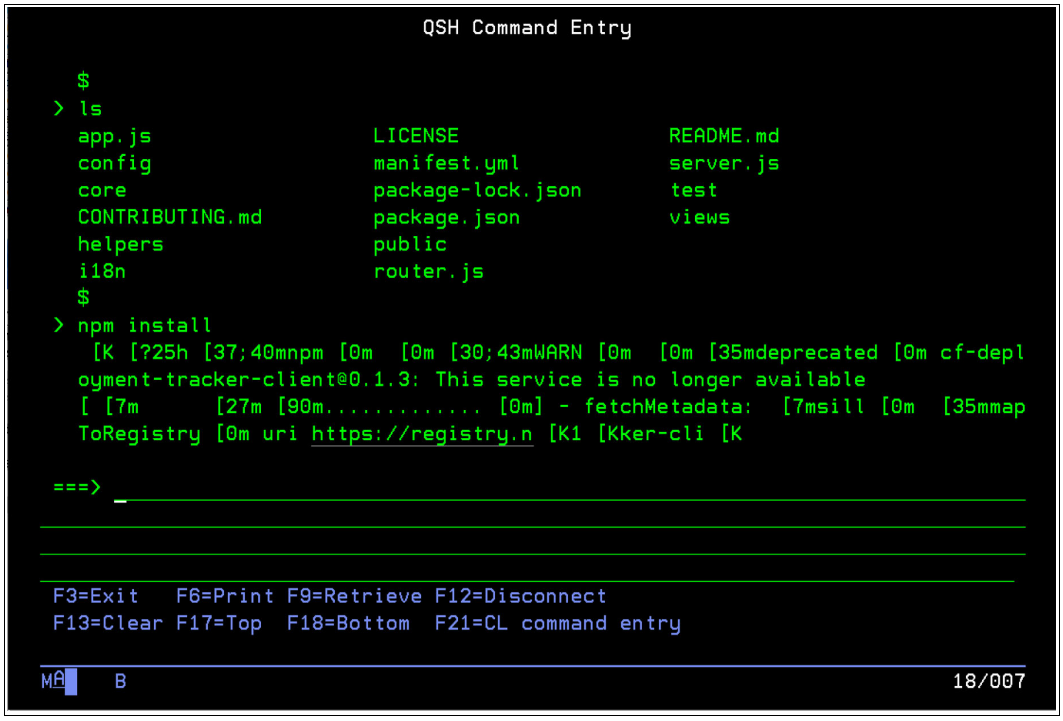

Figure 5-20 Installing the dependencies for the application

3. Change the application port to 8080 or another port that you prefer by running the following command:

```
export PORT=8080
```

4. Start the application by running this command (Figure 5-21):

```
npm start
```

> **Note:** If you want to run the application in the background, run the following command:
>
> `node server.js &`

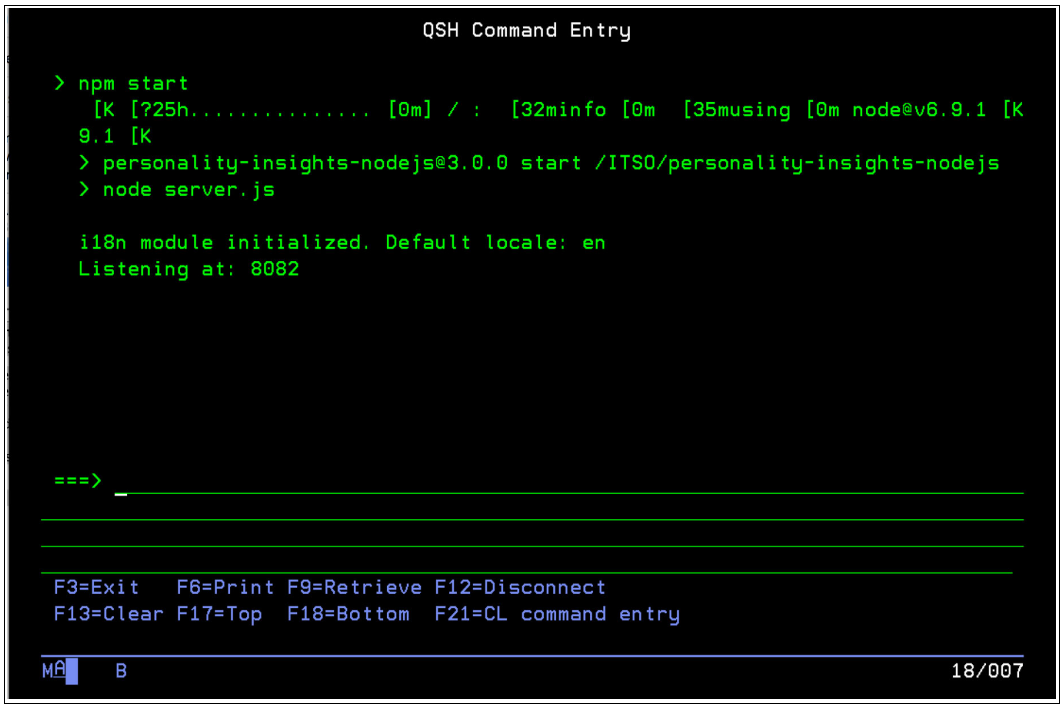

Figure 5-21 Starting the application

5.5 Testing the deployed application

Test the deployed application on Power Systems by completing the following steps:

1. Go to *{POWER_IP}:{APPLICATION_PORT}*. The window that is shown in Figure 5-22 opens.

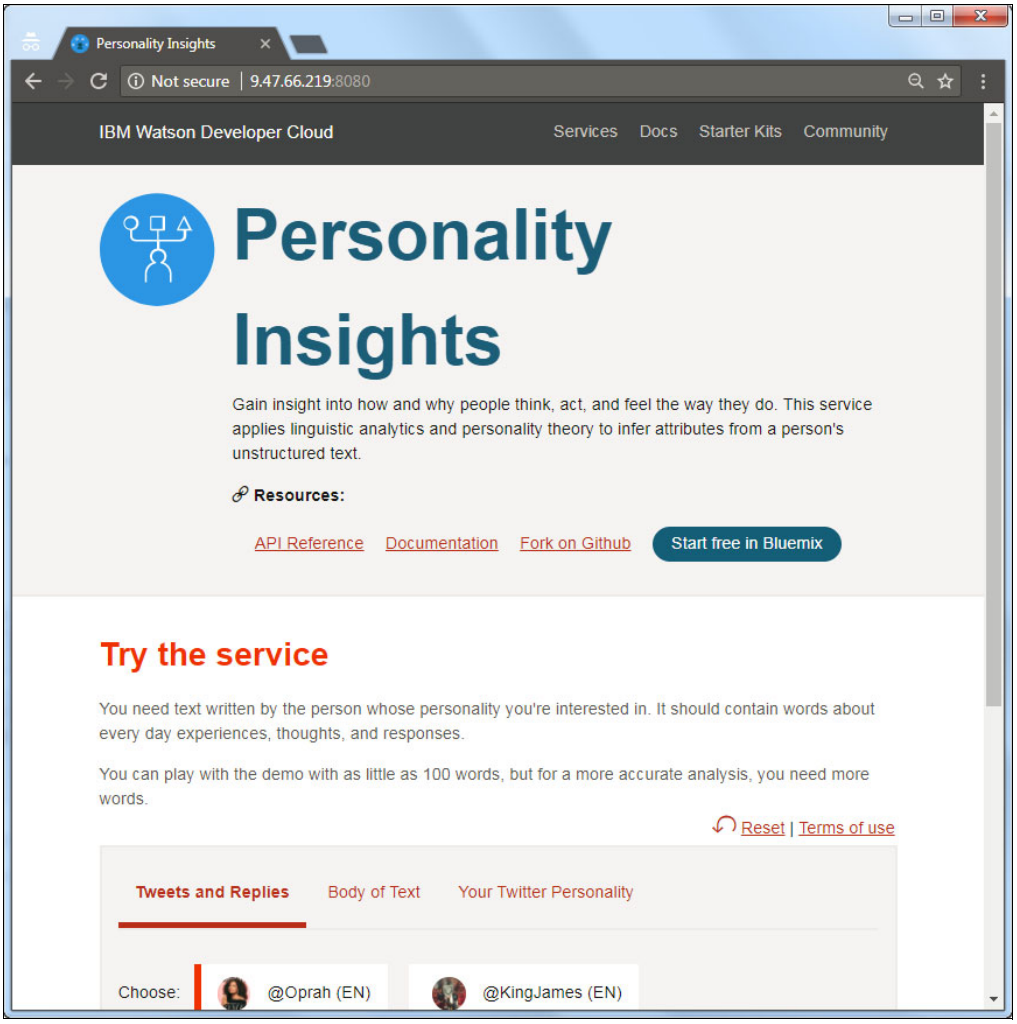

Figure 5-22 Watson Personality Insights demonstration app

2. Click **Body of Text** and then **Analyze**, as shown in Figure 5-23. This action sends the sample text to the Watson Personality Insights service and analyzes the personality of the individual based on the text.

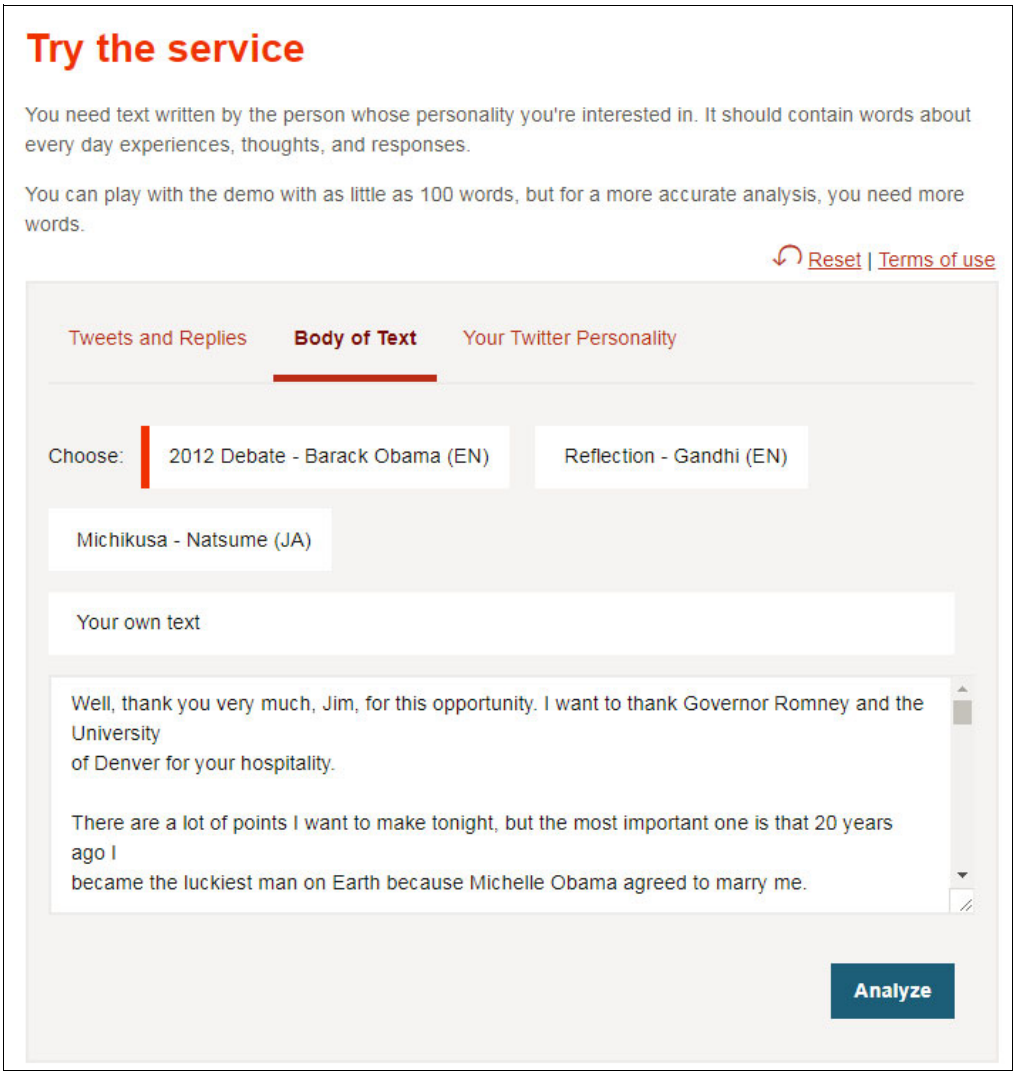

Figure 5-23 Text for Analysis

3. You receive the personality analysis based on the text, as shown in Figure 5-24.

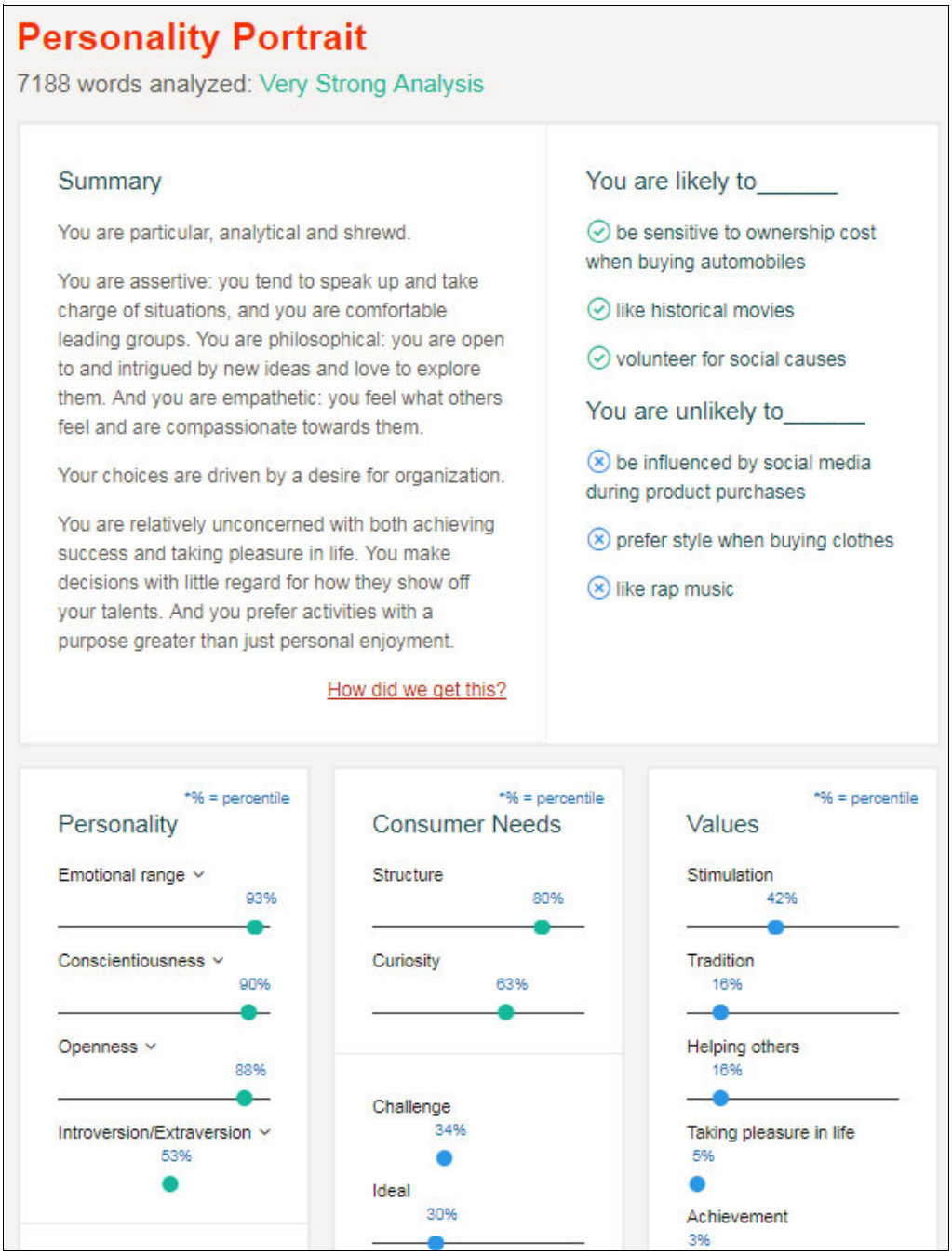

Figure 5-24 Personality Analysis Results

6

Use case implementation for IBM AIX and Linux on Power

In this chapter, you implement the Customer Feedback use case on a Power Systems server, which was described 3.2.5, “Customer service feedback: Real-time monitoring” on page 66.

The Customer Feedback application uses three Watson services. You create these services on IBM Cloud so that you can use them in the application.

The application uses DB2 on Power Systems for storing the customer feedback data. You connect to DB2.

This chapter describes how the integration between your application and Watson works by using the Node.js run time. You can use any other programming language that provides HTTP integration functions to integrate with Watson services.

The chapter includes the following topics:

© Copyright IBM Corp. 2018. All rights reserved.

6.1 Creating Watson services on IBM Cloud

The Customer Feedback use case depends on the following Watson services:

- Watson Language Translator
- Watson Natural Language Understanding (NLU)
- Watson Tone Analyzer

In this section, you create the Watson services on IBM Cloud and store the credentials to configure the application to connect to these services.

6.1.1 Creating the Watson Language Translator service

Customer feedback comes in different languages. You use Watson Language Translator to detect the language of the feedback and then translate it into English.

To create the Watson Language Translator service on IBM Cloud, complete the following steps:

1. Log in to the IBM Cloud console.
2. Click **Create resource**, as shown in Figure 6-1.

Figure 6-1 IBM Cloud Dashboard

3. Find the Watson Language Translator service in the IBM Catalog and click it (Figure 6-2).

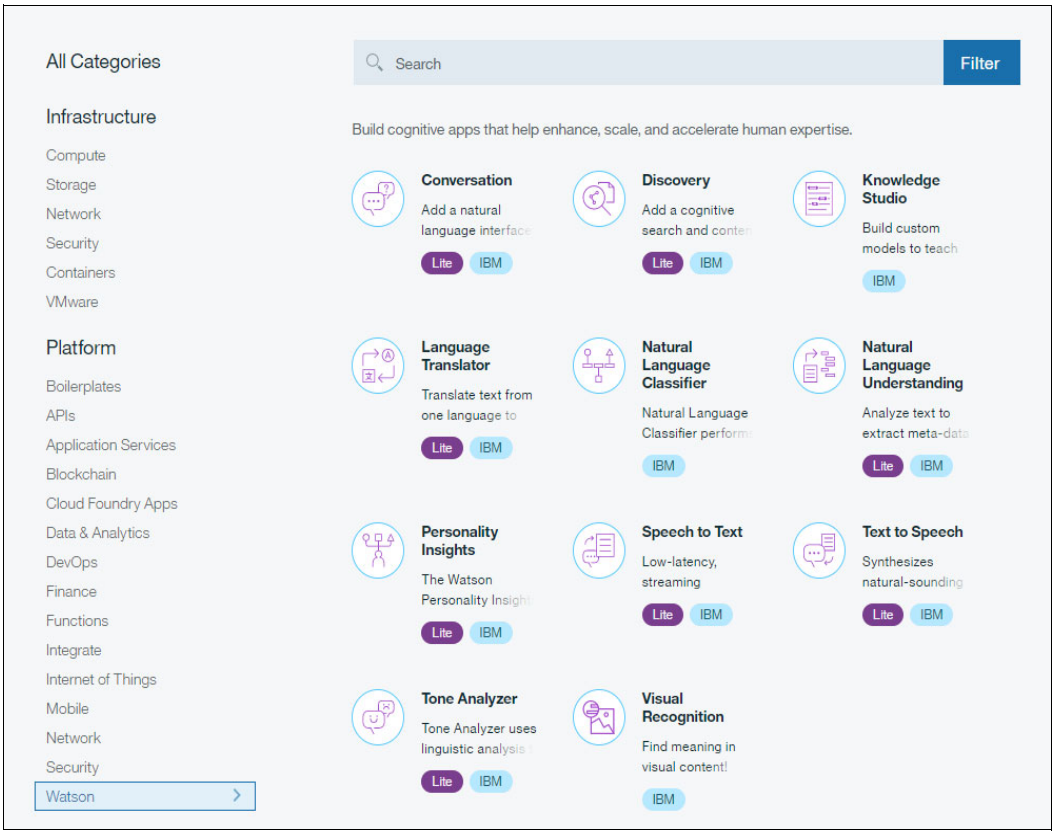

Figure 6-2 Watson services in the IBM Catalog

4. Keep the default values in the Watson Language Translator creation dialog (Figure 6-3) and click **Create**. This action creates a Watson Language Translator service on IBM Cloud.

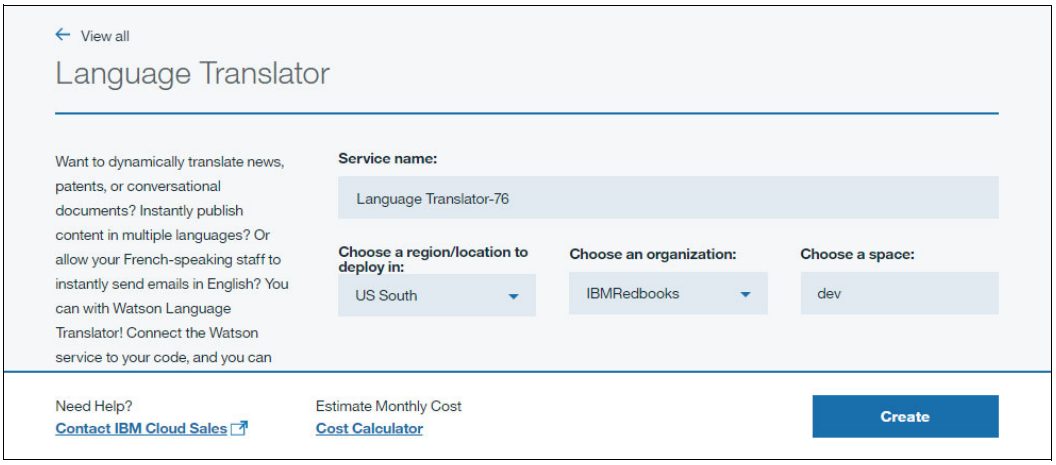

Figure 6-3 Watson Language Translator service creation dialog

5. To obtain your credentials of the Watson Language Translator service, complete the following steps. You must configure your application with these credentials.

   a. Click **Service credentials** from the left navigation bar in Service Details (Figure 6-4).

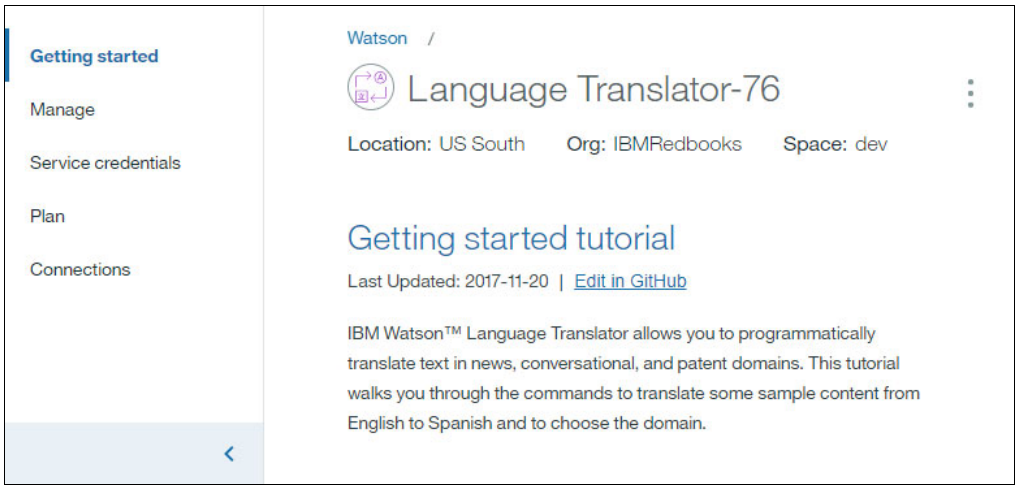

Figure 6-4 Watson Language Translator service details

   b. Click **New credential** (Figure 6-5).

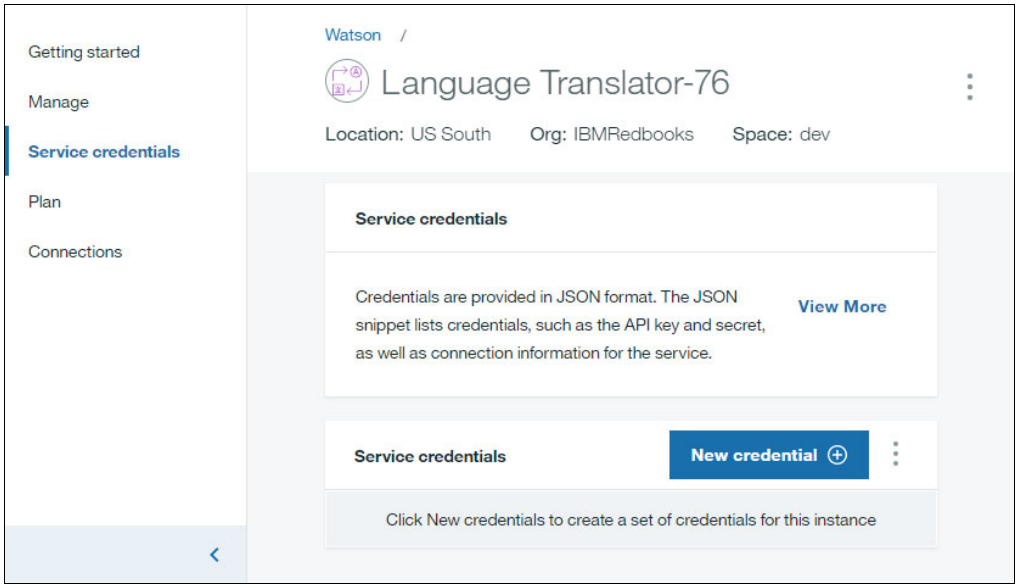

Figure 6-5 Watson Language Translator service credentials

c. A window opens where you add a credential (Figure 6-6). Click **Add**.

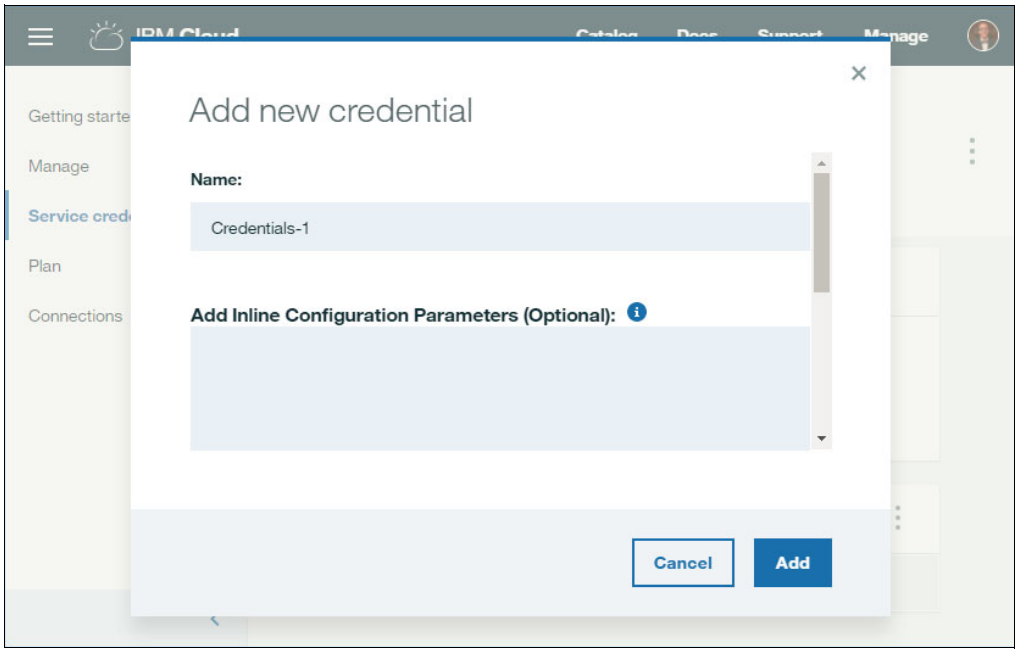

Figure 6-6 Add new credential window

d. IBM Cloud adds the credentials for the service, as shown in Figure 6-7.

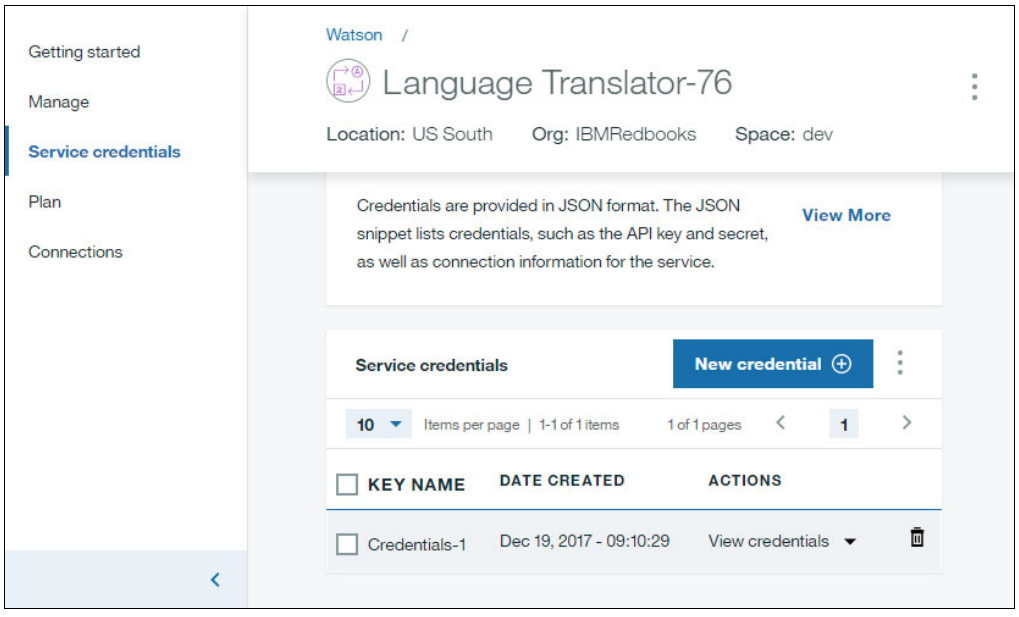

Figure 6-7 Service details with Watson Language Translator credentials added

e. Click **View credentials**.

f. Your Watson Personality Insights service credentials are displayed (Figure 6-8). Copy the credentials (user name and password) because you must configure your Customer Feedback application on the Power Systems server to use them.

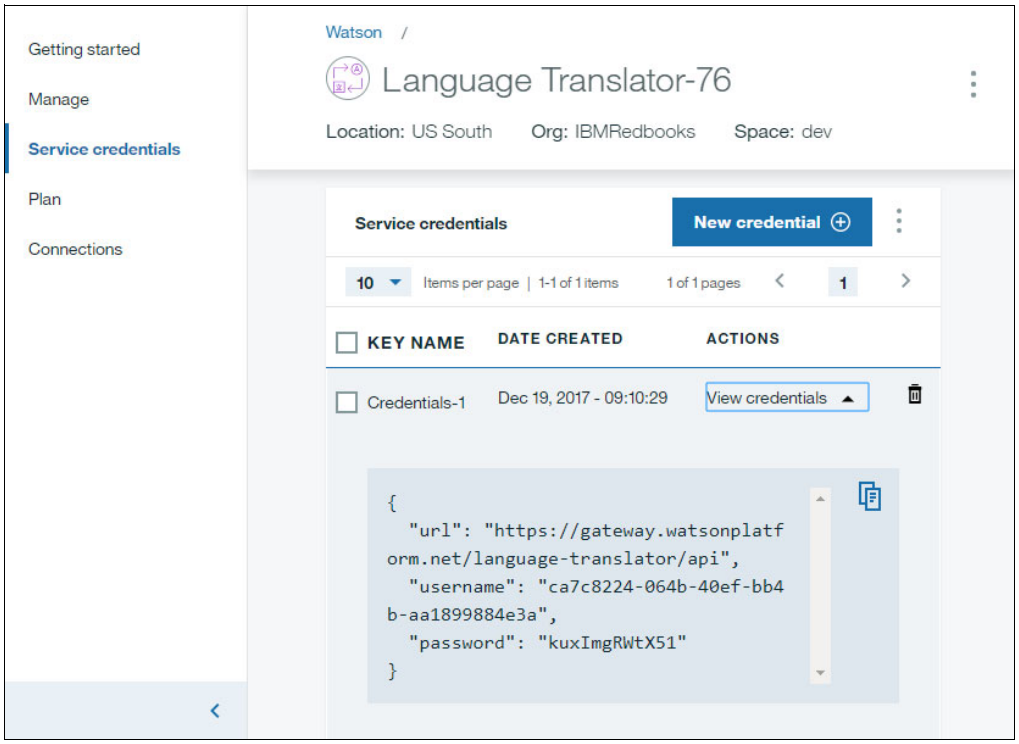

Figure 6-8 Watson Language Translator credentials

6.1.2 Creating the Watson Natural Language Understanding service

You use the NLU service to extract the top keywords in the customer feedback and analyze the sentiment of the feedback to determine whether it is positive, neutral, or negative feedback.

To create an NLU service on IBM Cloud, complete the following steps:

1. Open the IBM Cloud Catalog by clicking **Create resource** from the Dashboard or clicking **Catalog** from the top navigation bar (Figure 6-9).

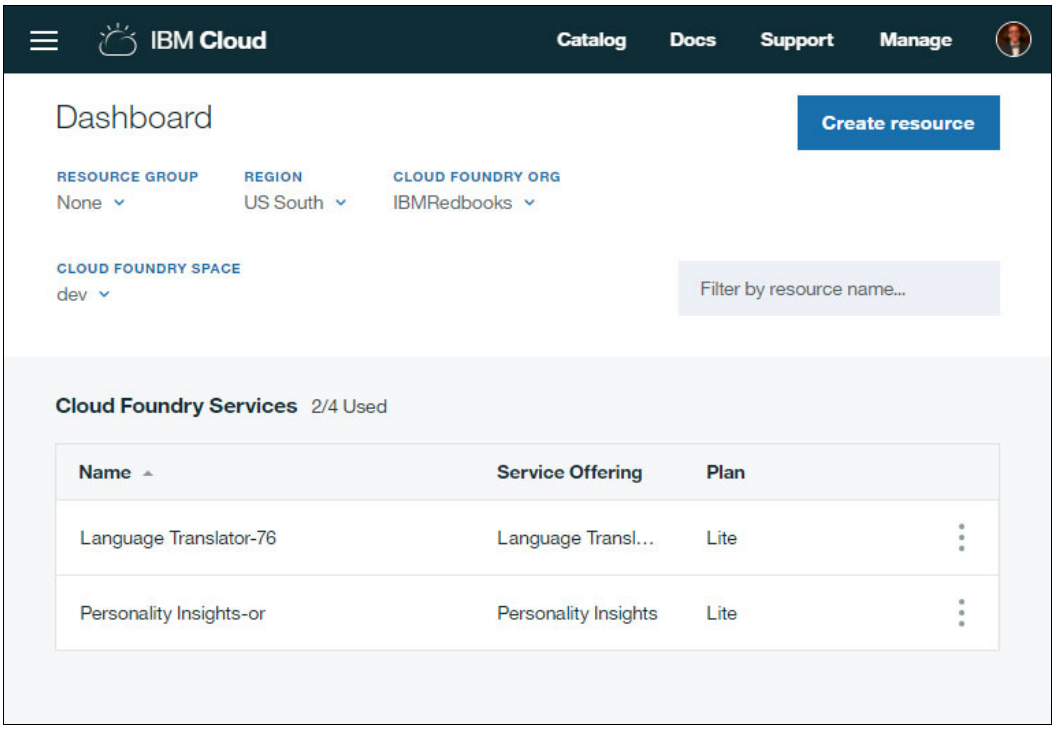

Figure 6-9 IBM Cloud Dashboard

2. Find the NLU service in the IBM Catalog and click it.
3. Keep the default values in the NLU service creation dialog and click **Create**. This action creates an NLU service on IBM Cloud.
4. To get the credentials of your NLU service, complete the following steps:
   a. Click **Service credentials** from the left navigation bar in Service Details.
   b. Click **New credential**.
   c. A window opens where you can add a credential. Click **Add**. IBM Cloud adds the credentials for the service.
   d. Click **View credentials**.
   e. Your NLU service credentials are displayed. Copy the credentials (user name and password) because you must configure your Customer Feedback application on the Power Systems server to use them.

6.1.3 Creating the Watson Tone Analyzer service

You use Watson Tone Analyzer service to extract the anger, fear, joy, and sadness tones from the customer feedback.

To create a Watson Tone Analyzer service in IBM Cloud, complete the following steps:

1. Open the IBM Cloud Catalog by clicking **Create resource** from the Dashboard or clicking **Catalog** from the top navigation bar in IBM Cloud Dashboard.
2. Find the Watson Tone Analyzer service in the IBM Catalog and click it.
3. Keep the default values in the Watson Tone Analyzer service creation window and click **Create**. This action creates a Watson Tone Analyzer service on IBM Cloud.
4. To get the credentials of your Watson Tone Analyzer service, complete the following steps:
   a. Click **Service credentials** from the left navigation bar in Service Details.
   b. Click **New credential**.
   c. A window opens where you can add a credential. Click **Add**. IBM Cloud adds the credentials for the service.
   d. Click **View credentials**.
5. Your Watson Tone Analyzer service credentials are displayed. Copy the credentials (user name and password) because you must configure your Customer Feedback application on Power Systems servers to use them.

6.2 Database creation

This section describes the detailed creation of a database. If you do not have DB2 installed on your Power Systems server, follow the instructions for DB2 installation that are described in Appendix A, “Setup and configuration of the IBM DB2 server” on page 185.

To create a database, complete the following steps:

1. Create the database and connect to it by running the three commands that are shown in Example 6-1.

Example 6-1 Creating a database and making a connection

```
$ db2 create database HPD2
DB20000I  The CREATE DATABASE command completed successfully.

$ db2 list db directory
 System Database Directory
 Number of entries in the directory = 1

Database 1 entry:
 Database alias                       = HPD2
 Database name                        = HPD2
 Local database directory             = /home/nodedb2
 Database release level               = 10.00
 Comment                              =
 Directory entry type                 = Indirect
 Catalog database partition number    = 0
 Alternate server hostname            =
 Alternate server port number         =
```

```
$ db2 connect to HPD2
   Database Connection Information
 Database server        = DB2/AIX64 10.5.8
 SQL authorization ID   = NODEDB2
 Local database alias   = HPD2
```

2. Create the database schema, as described in Example 6-2.

Example 6-2 Creating the database schema

```
$ db2 create schema testschema authorization nodedb2
DB20000I  The SQL command completed successful

$ db2 set schema=testschema
DB20000I  The SQL command completed successful
```

3. Enable the operating system (OS) authentication to enable you to log in remotely to the database, as shown in Example 6-3, by editing `ENABLE_OS_AUTHENTICATION` in the `db2rfe.cfg` configuration file.

Example 6-3 Enabling the operating system authentication

```
$ vi /home/nodedb2/sqllib/instance/db2rfe.cfg
Change ENABLE_OS_AUTHENTICATION to have the value "YES"

$ /home/nodedb2/sqllib/instance/db2rfe -f
/home/nodedb2/sqllib/instance/db2rfe.cfg
```

4. Create a table for storing customer feedback data by using the SQL command, as shown in Example 6-4. If necessary, download the database SQL script.

Example 6-4 Creating a table for customer feedback

```
db2=> CREATE TABLE TESTSCHEMA.FEEDBACK
    (
      ID INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT
BY 1),
        ORIGINAL_FEEDBACK VARCHAR(2000),
        LANGUAGE CHAR(2),
        ENGLISH_FEEDBACK VARCHAR(2000),
        SENTIMENT CHAR(8),
        ANGER DECIMAL(5,2),
        FEAR DECIMAL(5,2),
        JOY DECIMAL(5,2),
        SADNESS DECIMAL(5,2),
        KEYWORDS VARCHAR(2000)
    )
```

6.3 Deploying the code on Power Systems servers

To deploy the code on Power Systems servers, complete the following steps. For more information about the steps of deploying the code on Power Systems servers, see 5.3, “Cloning and deploying the Watson Personality Insights Node.js demonstration app on AIX and Linux on Power” on page 91.

1. Go to `ibm-redbooks` by running the following command:

   ```
   cd ibm-redbooks
   ```

2. Clone the Customer Feedback application by running the following command:

   ```
   git clone https://github.com/ibm-redbooks-dev/customer-feedback-power
   ```

3. Go to the directory of the application by running the following command:

   ```
   cd customer-feedback-power
   ```

   Edit the `.env` file with the credentials of the Watson services that you copied in 6.1, “Creating Watson services on IBM Cloud” on page 108 and set the following fields by running **`vi .env`**:

   - `LANGUAGE_TRANSLATOR_USERNAME`
   - `LANGUAGE_TRANSLATOR_PASSWORD`
   - `NATURAL_LANGUAGE_UNDERSTANDING_USERNAME`
   - `NATURAL_LANGUAGE_UNDERSTANDING_PASSWORD`
   - `TONE_ANALYZER_USERNAME`
   - `TONE_ANALYZER_PASSWORD`

4. Edit the DB2 database connection details in the `.env` file.

5. Install all the required dependencies of the application by running the following command:

   ```
   npm install
   ```

   Note: The **`npm install`** command might not work on your Power platform because there is an open issue with the ibm_db Node.js module.

   If **`npm install`** does not work for you, complete the following steps:

   a. Check the status of the issue by going to GitHub.

   b. Continue with 6.4, “Customer Feedback application code” on page 117.

6. Change the application port to 8080 or another port of your choosing by running the following command:

   ```
   export PORT=8080
   ```

7. Run the application in the background as a service by running the following command and pressing Enter:

   ```
   nohup npm start &
   ```

6.4 Customer Feedback application code

In this section, you explore the `package.json` file and the code structure.

6.4.1 The package.json file

The `package.json` file holds metadata that is relevant to the project. This file is used by Node Package Manager (NPM) to identify the project and handle project dependencies. It can also contain other metadata, such as a project description, the version of the project in a particular distribution, license information, scripts, and configuration data, all of which can be vital to both NPM and to the users of the package. The `package.json` file is normally in the root directory of a Node.js project.

Example 6-5 shows the package.json file that is used in the Customer Feedback application.

Example 6-5 The package.json file

```
{
  "name": "customer-feedback-power",
  "version": "0.0.1",
  "description": "Customer feedback application on Power Systems integrating with
Watson services",
  "dependencies": {
    "body-parser": "latest",
    "dotenv": "^4.0.0",
    "express": "4.*",
    "watson-developer-cloud": "^3.0.3",
    "ibm_db": "^2.2.1"
  },
  "scripts": {"start": "node app.js"}
}
```

The `name` of the project is `customer-feedback-power`, and because this is the initial pre-release version, it is labeled `0.0.1`.

The `description` briefly explains the purpose of the application, which is to create the Customer Feedback Use case scenario by using a Power Systems server with a few Watson services.

The application uses the following dependencies:

- `body-parser` as a dependency to manage HTTP POST requests.
- `dotenv` is used to stop hardcoding of the credentials, which in our use case is the database and Watson services credentials. Instead, the credentials are placed in the `.env` file.
- `express` handles routes and HTTP calls, and creates and initiates the HTTP server.
- `watson-developer-cloud` connects to Watson services on the IBM Cloud.
- `ibm_db` connects to the DB2 database.

The `scripts` section has an object that NPM uses to start the application.

6.4.2 Code structure

The code is organized to be clear to understand and to follow preferred practices. Figure 6-10 on page 119 shows the directory structure of the code:

- `dashboard`: A folder that contains pages that are used by intranet users. The `index.html` file is the dashboard that is used by intranet users to monitor the feedback sentiment in real time.
- `db_scripts`: This directory contains the script for the database. `CREATE_TABLE.sql` is the script that you must run against your database to create the tables that are used to reproduce the scenario and run the application.
- `node_modules`: This directory is populated by NPM and contains all the fetched dependencies (modules) for this project.
- `routes`: The folder that opens and imports all the endpoints into the express instance:
  - `api.js`: This file routes the database calls (**`GET`** and **`POST`**) that are used in this project.
  - `index.js`: This file is responsible for loading all route files. It is used to clarify application routing and for ease of future maintenance.
  - `watson.js`: This file routes the Watson services that are used in this project.
- `services`: Although the `routes` directory is responsible for the organization and distribution of the calls, `services` is the directory that contains the business logic:
  - `api`: This directory controls the services that aRE related to the Representational State Transfer (REST) application programming interface (API) calls that are used in this project. The `feedbackServices.js` file contains the logic to save the user feedback, after it is analyzed by Watson, into the database, and to fetch the data from the database to generate the dashboard. All calculations are performed in this file.
  - `watson`: This directory controls the services that are related to the Watson services:
    - `languageTranslatorServices.js`: This module is where the application detects the language of the original feedback and then translates it to English.
    - `naturalLanguageUnderstandingServices.js`: This module is where the application detects keywords and sentiment.
    - `toneAnalyzerServices.js`: This module is where the application detects tones, such as anger, sadness, joy, and fear.
- `views`: The folder that contains the user's web pages:
  - `index.html`: The entry point to view the application. It contains the form where users submit their feedback.
  - `thanks.html`: A web page that thanks the user for their feedback after it is analyzed and stored in the database.
- `.env`: This file contains all the private credentials that are used in the application. Its purpose is to *not* share sensitive information, such as application secrets and service credentials when promoting the code to GitHub or any other repository.

  To use the sample use case application, insert your own credentials into this file.
- `.gitignore`: This file is used by the Git client to exclude promotion of some files or folders that are used only in the development phase, such `node_modules` and configurations for the code editor.
- `app.js`: The application starter, which is responsible for loading of all dependencies, creating the routes, and creating and starting the server.
- `package.json`: This file is described in 6.4.1, "The package.json file" on page 117.

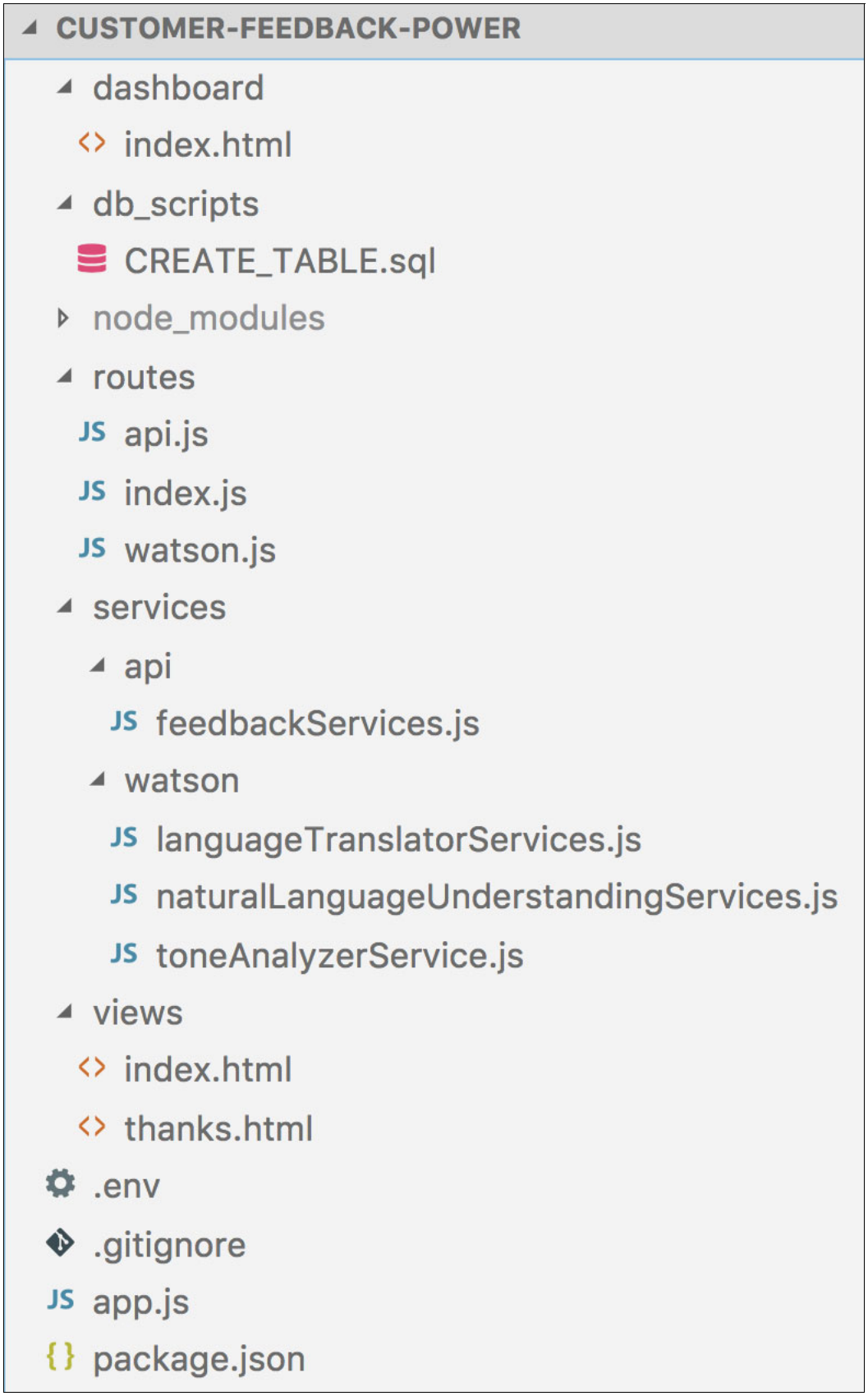

Figure 6-10 Code directory structure for the use case application

6.5 Watson services integration

This section describes how the integration with Watson services work in your Customer Feedback Node.js application.

6.5.1 Watson Language Translator

To learn about the integration between the Customer Feedback application running on a Power Systems server and Watson Language Translator service, complete the following steps:

1. Go to the following directory;

   `customer-feedback-power\services\watson\languageTranslatorServices.js`

   The `languageTranslatorServices.js` module contains the Node.js code that is integrated with Watson Language Translator service. In Example 6-6, the credentials of the service are taken from the `.env` file to connect to the service.

   Example 6-6 Initializing the Watson Language Translator service

```
var language_translator = new LanguageTranslatorV2({
    username: process.env.LANGUAGE_TRANSLATOR_USERNAME,
    password: process.env.LANGUAGE_TRANSLATOR_PASSWORD,
    url: 'https://gateway.watsonplatform.net/language-translator/api/'
});
```

   The `languageTranslatorServices.js` module contains two functions:

   – The **`identifyLanguage`** function takes the customer feedback as an input and returns the language of the feedback.

   – The **`translateIntoEnglish`** function takes the customer feedback and the language of the feedback as inputs and returns the English translation.

   In Example 6-7, you call the **`identify`** function from Watson Language Translator, passing the customer feedback to it.

   Example 6-7 Calling the identify function from the Watson Language Translator service

```
language_translator.identify({
            text: feedback
}
```

   For example, consider the customer feedback that is shown in Example 6-8 as an input to the service.

   Example 6-8 Sample customer feedback

```
Eu não gostei da recepcionista no primeiro dia. Eu tinha medo deles entrarem no
meu quarto e abrirem minhas malas. O café da manhã me deixou nervoso.
Eu estava a ponto de desistir porque eu não estava satisfeito com o hotel, mas
posso dizer que eu adorei de ficar no hotel desde então.
O bom é que por eu ter ficado bastante tempo, agora eu vou um cliente VIP, e
vou economizar bastante dinheiro na minha próxima estadia.
A vista daqui é maravilhosa, mas o dia estava chuvoso.
```

The Watson Translation service responds in a JavaScript Object Notation (JSON) format, as shown in Example 6-9.

Example 6-9 Sample Translation Service identify function response in JSON

```
{
  "languages": [
    {
      "language": "pt",
      "confidence": 1
    },
    {
      "language": "es",
      "confidence": 1.17276e-7
    },
    {
      "language": "it",
      "confidence": 1.49735e-17
    },
    ...
  ]
}
```

The Watson Translation service responds with an array of languages, which is sorted by the degree of confidence. In the example, the response is `pt`, which is the language code for Portuguese, with a confidence of 100%.

2. Example 6-10 shows the conversion of the response text into a JSON object. Get the first array element of `languages`, and then get the `language` field to return the identified language to the caller. In the example, the caller receives `pt` as an output.

Example 6-10 Parsing the JSON response

```
var languages = JSON.parse(JSON.stringify(language, null, 2));
var detectedLanguage = languages.languages[0].language;
```

3. To translate the feedback into English, see Example 6-11.You call the **`translate`** function from Watson Language Translator and pass it the customer feedback and a string representing the model in the form of (Source Language - Target Language). In the example, this information is `pt-en`, which signifies a translation from Portuguese to English.

Example 6-11 Translating the feedback into English

```
language_translator.translate({
            text: feedback,
            model_id: sourceLanguage + '-' + ENGLISH_LANGUAGE
}
```

The Watson Language Translator service responds in a JSON format, as shown in Example 6-12.

Example 6-12 Watson Language Translator JSON response for the sample customer feedback

```
{
  "translations": [
    {
      "translation": "I don't like the receptionist in the first day. I was
afraid of them come in my room and open my bags. The breakfast made me nervous.
```

```
\r\nI was about to give up because I wasn't satisfied with the hotel, but I can
say that I love to stay in the hotel since then. \r\nThe good is that I have
been for a long time, now I'm a VIP client, and I'll save enough money in my
next stay.\r\nThe view here is wonderful, but the day was rainy."
    }
  ],
  "word_count": 88,
  "character_count": 481
}
```

The JSON has three fields:

- `translations`: It consists of an array containing the translation output, corresponding to the input text. In the sample, it returns the English translation.
- `word_count`: It contains the word count of the translated text.
- `character_count`: It contains the character count of the translated text.

4. The following code example shows that you are returning the `translation` field of the first element of the `translations` array to the caller. You are passing the first element of the array because you passed a single text for translation.

```
translation.translations[0].translation
```

6.5.2 Watson Natural Language Understanding

To learn about the integration between the Customer Feedback application running on the Power Systems server and the NLU service, complete the following steps:

1. Go to the following directory:

```
customer-feedback-power\services\watson\naturalLanguageUnderstandingServices.js
```

The `naturalLanguageUnderstandingServices.js` module contains the Node.js code that is integrated with the NLU service.

The `languageLanguageUnderstandingServices.js` module contains the **`extractSentimentAndKeywords`** function. This function calls the Watson Language Understanding service to extract sentiments and keywords for the English feedback.

2. In Example 6-13, you call the `analyze` function from NLU, passing it the English translation of the customer feedback, and two features, `sentiment` and `keyword`. These two features form the request to the service to analyze the text for sentiment and keywords only.

Example 6-13 Calling the analyze function from the Watson NLU service

```
natural_language_understanding.analyze({
            text: feedback,
            features: {
                sentiment: {},
                keywords: {}
            }
        },..
```

Use the English translation of the feedback that is retrieved in Example 6-9 on page 121 as input for this service.

The NLU service responds in a JSON format, as shown in Example 6-14, for the example input.

Example 6-14 Sample Watson NLU service analyze function JSON response


```
{
  "usage": {
    "text_units": 1,
    "text_characters": 423,
    "features": 2
  },
  "sentiment": {
    "document": {
      "score": -0.212124,
      "label": "negative"
    }
  },
  "language": "en",
  "keywords": [
    {
      "text": "VIP client",
      "relevance": 0.992646
    },
    {
      "text": "long time",
      "relevance": 0.917157
    },
    {
      "text": "receptionist",
      "relevance": 0.720299
    },
    {
      "text": "hotel",
      "relevance": 0.671292
    },
    {
      "text": "stay",
      "relevance": 0.631676
    },
    {
      "text": "breakfast",
      "relevance": 0.630295
    },
    {
      "text": "room",
      "relevance": 0.562273
    },
    {
      "text": "bags",
      "relevance": 0.561604
    },
    {
      "text": "view",
      "relevance": 0.550267
    }
```

```
  ]
}
```

The following code example shows that you are returning the sentiment to the caller. In the example, the `sentiment` is `negative`.

```
output.sentiment = nluResponse.sentiment.document.label;
```

Example 6-15 shows that you are iterating over the `keywords` array to output the keywords in a space-separated string.

Example 6-15 Parsing the keywords array JSON to return the relevant keywords

```
// Return the relevant keywords as space separated
nluResponse.keywords.forEach(function (keyword) {
   if (keyword.relevance > 0.5) {
      output.keywords += keyword.text + ' ';
   }
});
```

6.5.3 Watson Tone Analyzer

To learn about the integration between the Customer Feedback application running on the Power Systems server and Watson Tone Analyzer service, complete the following steps:

1. Go to the following directory:

   `customer-feedback-power\services\watson\toneAnalyzerServices.js`

   The `toneAnalyzerServices.js` module contains the Node.js code that is integrated with the Watson Tone Analyzer service.

   The `toneAnalyzerServices.js` module contains the **`extractTones`** function. This function calls the Watson Tone Analyzer service to extract the tones for the English or French feedback.

2. Watson Tone Analyzer supports English and French. When the feedback language is French, you pass the original feedback text to the service. Otherwise, pass the English feedback translation. The service can analyze tones at the document level, and each sentence. The **`sentences`** parameter is passed as `false`, as shown in Example 6-16, which means that you are interested in retrieving only the document level tone analysis.

   Example 6-16 Watson Tone Analyzer input

```
if (feedbackLanguage === FRENCH_LANGUAGE) {
    params = {
        tone_input: originalFeedback,
        content_type: 'text/plain',
        sentences: false,
        content_language: FRENCH_LANGUAGE
    };
} else {
    params = {
        tone_input: translatedFeedback,
        content_type: 'text/plain',
        sentences: false
    };
}
```

3. In the following code, call the **tone** function from Watson Tone Analyzer and pass the parameters to it:

   ```
   tone_analyzer.tone(params,...
   ```

   Consider the English translation of the feedback that is retrieved in Example 6-9 on page 121 as input for this service.

   Watson Tone Analyzer service responds in a JSON format, as shown in Example 6-17, for the example input.

   Example 6-17 Sample Watson Tone Analyzer service tone function response in JSON

   ```
   {
     "document_tone": {
       "tones": [
         {
           "score": 0.671598,
           "tone_id": "joy",
           "tone_name": "Joy"
         },
         {
           "score": 0.623749,
           "tone_id": "fear",
           "tone_name": "Fear"
         }
       ]
     }
   }
   ```

4. Example 6-18 shows that you are iterating over the `tones` array to output the returned tones.

   Example 6-18 Parsing the tones array JSON to return the tones

   ```
   toneAnalyzerResponse.document_tone.tones.forEach(function(tone) {
       var outputTone = {};
       outputTone.name = tone.tone_name;
       outputTone.score = tone.score;
       outputTones.push(outputTone);
   });
   ```

   In the example, the returned tones are Joy with a score of 67% and Fear with a score of 62%.

6.6 Running the application

Test the deployed application on Power Systems by completing the following steps:

1. Go to *{POWER_IP}:{APPLICATION_PORT}*. The application opens, as shown in Figure 6-11.

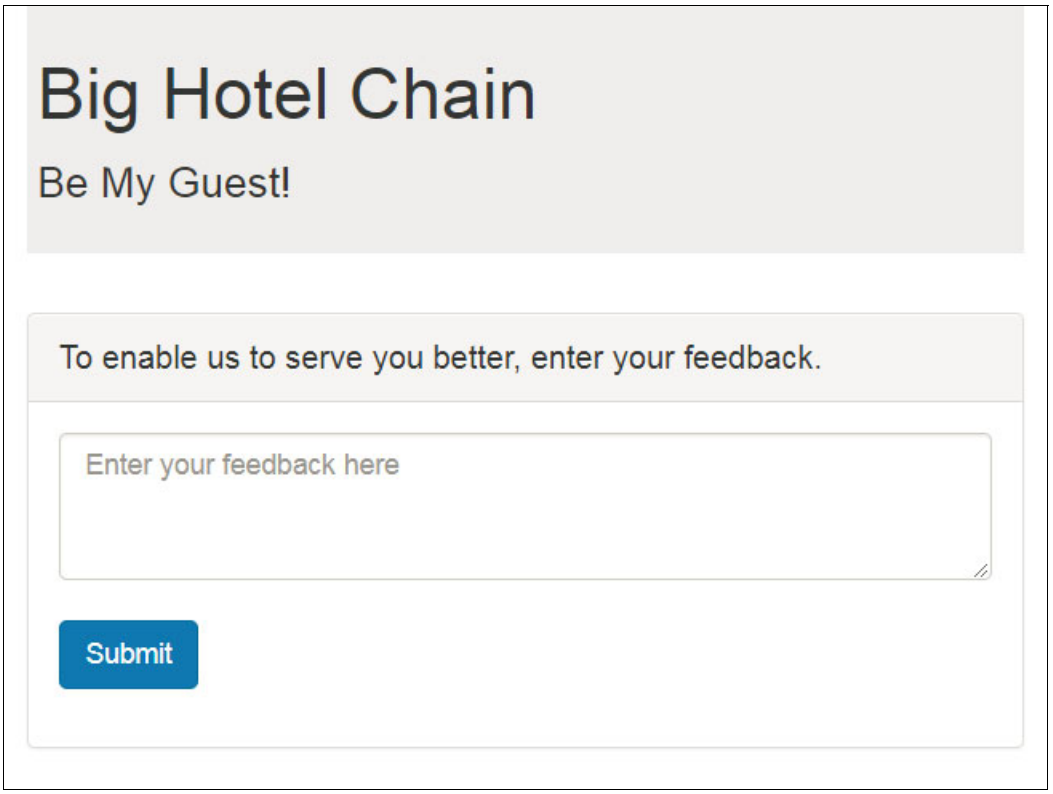

Figure 6-11 Customer Feedback home page

2. Enter any sample customer feedback in any language. For example, enter the sample feedback in Example 6-8 on page 120, as shown in Figure 6-12.

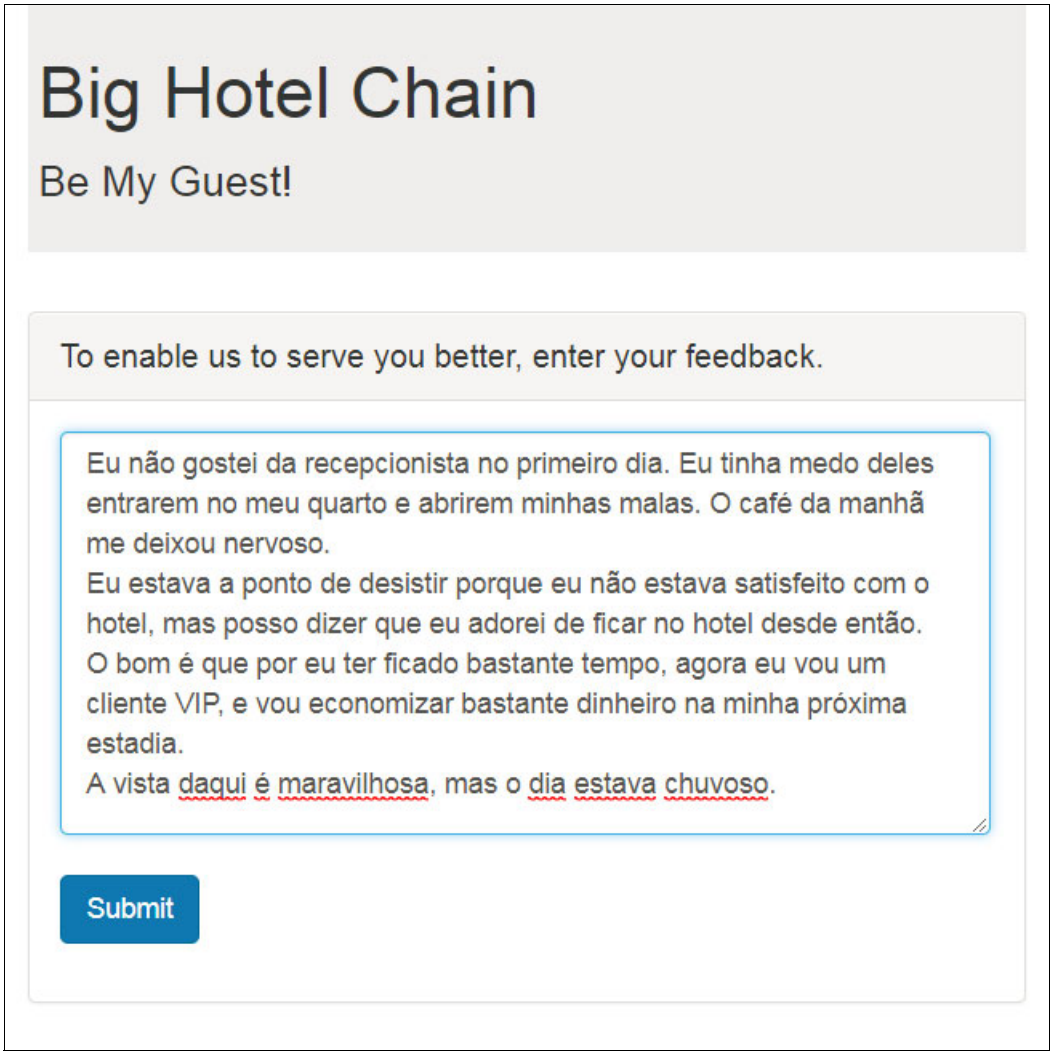

Figure 6-12 Sample feedback

3. Click **Submit**. You are redirected to a thank you page, as shown in Figure 6-13.

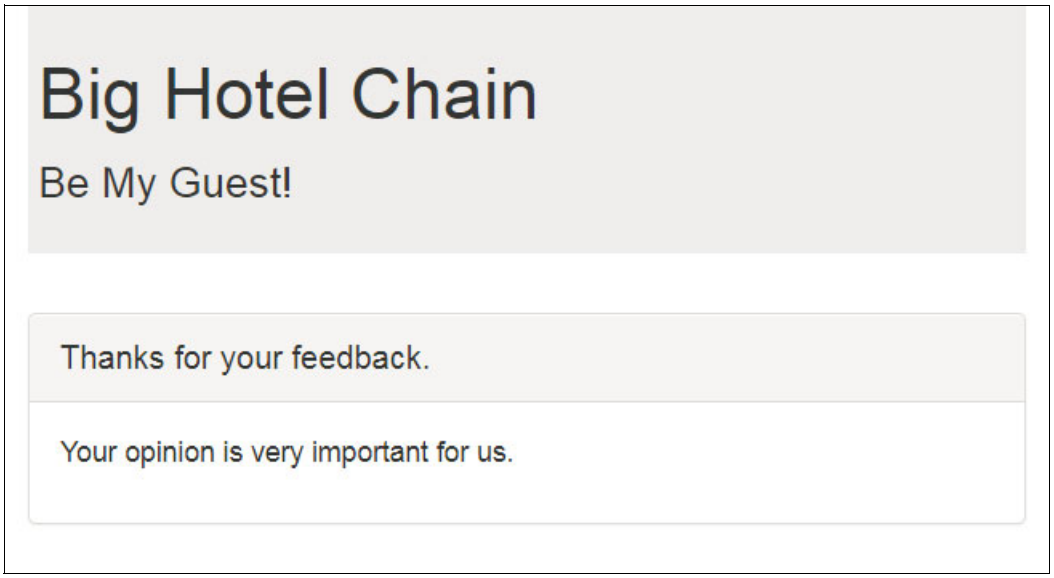

Figure 6-13 Thank you page

4. Go to *{POWER_IP}:{APPLICATION_PORT}/dashboard.* The Dashboard opens. In the Overall Customer Satisfaction section in the Dashboard, notice the negative sentiment, and the feelings Joy and Fear, as shown in Figure 6-14.

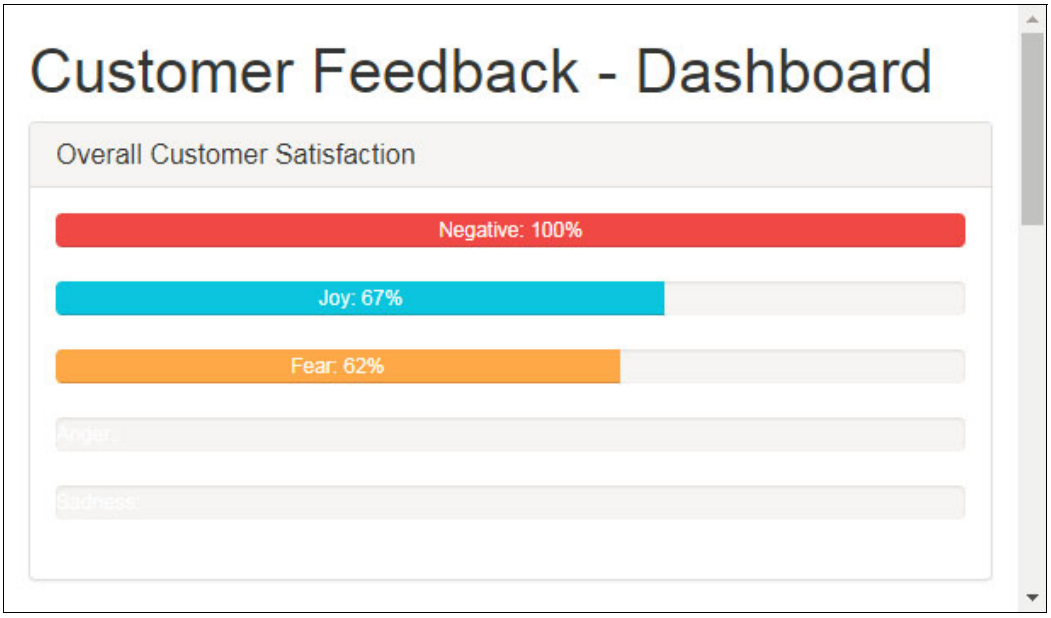

Figure 6-14 Customer Feedback Dashboard: Overall Customer Satisfaction

In the Detailed Customer Satisfaction section, notice the original feedback, the identified language of the feedback, the English translation of the feedback, and the extracted keywords, as shown in Figure 6-15.

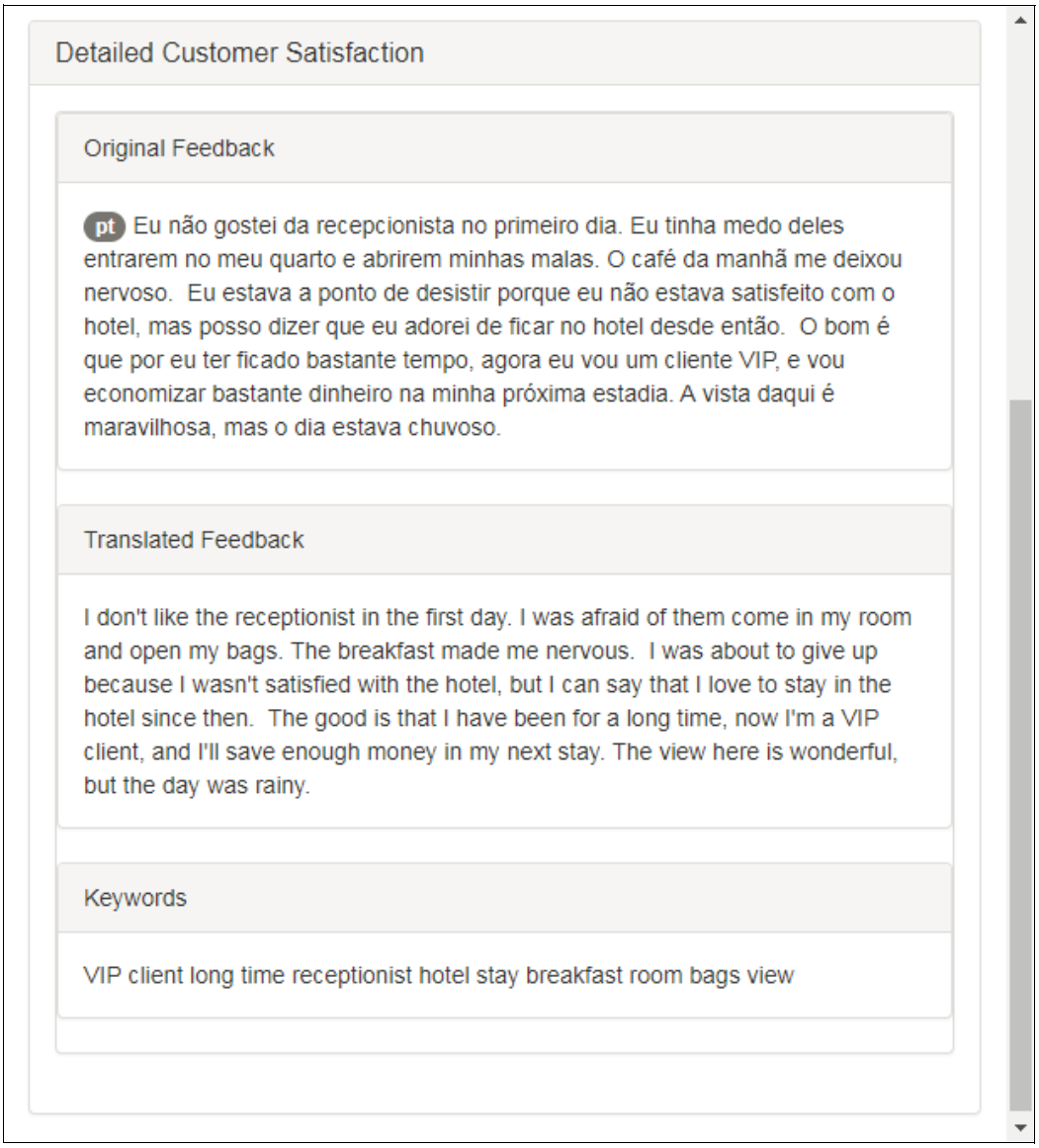

Figure 6-15 Customer Feedback Dashboard: Detailed Customer Satisfaction

5. Go back to step 1 on page 126, enter more feedback, and see the effect of that feedback on the Dashboard.

7

Use case implementation for IBM i

This chapter describes how an existing Report Program Generator (RPG) application running on IBM i can be used to join IBM Watson services to create a cloud-based application.

This chapter describes how to create an interactive user interface that runs in IBM Cloud, and uses IBM Watson services to improve the user experience while using a Representational State Transfer (REST) based application programming interface (API) to access your existing back-end system of record.

The chapter includes the following topics:

© Copyright IBM Corp. 2018. All rights reserved.

7.1 Use case and architecture overview

In this use case, you build an application that enables users to search for flights based on three input parameters: source location, destination location, and travel dates.

To build the application, you use IBM Watson services together with an existing RPG application running on IBM i to highlight the integration capability by using IBM i built-in tools and technologies. You use the IBM i Integrated Web Services (IWS) server to create a REST API over the existing RPG application.

Figure 7-1 shows the architecture overview of the solution for the use case.

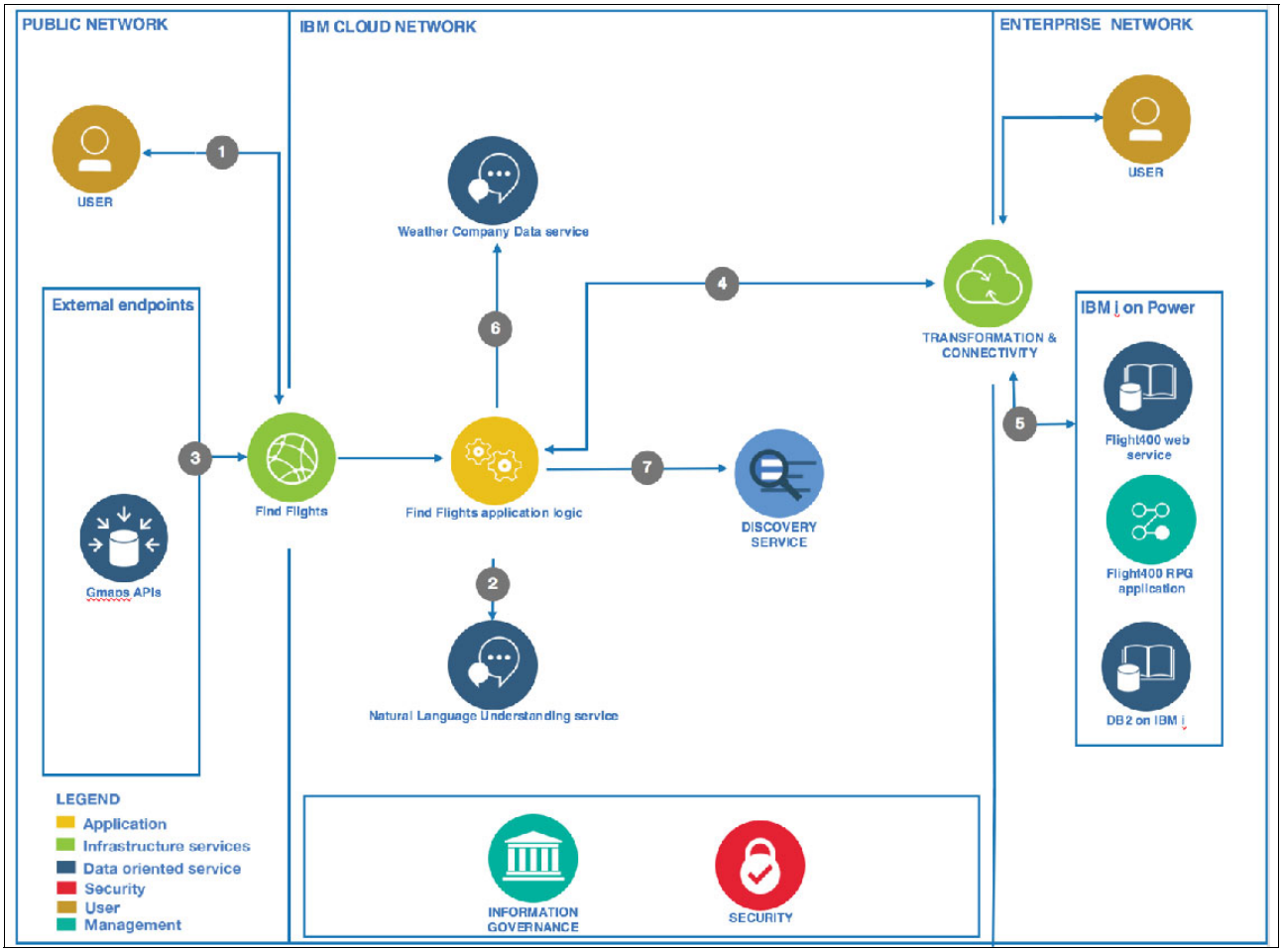

Figure 7-1 Architecture overview of IBM i and Watson integration use case

On the IBM i system, you create a RESTful web service that is named *Findflight* by using the existing Flight400 RPG program. The service is created by using the IWS server that is included as part of the IBM i operating system (OS).

The Flight400 application is an RPG program that simulates an airline ticketing system. It was developed as a sample application to show how to use RPG code to interact with the database and how to build 5250 screens to interact with the RPG. It has been updated many times over the years to use the most recent RPG technologies, including Modern RPG. For more information about Modern RPG, see *Who Knew You Could Do That with RPG IV? Modern RPG for the Modern Programmer*, SG24-5402.

In Flight400, there is a procedure that is included in an IBM Integrated Language Environment® (ILE) service program that is named *Findflight*. This procedure contains five parameters:

- **FROMCITY**
- **TOCITY**
- **FLIGHTDATE**
- **FLIGHTCOUNT**
- **FLIGHTS**

The procedure searches for available flights between **FROMCITY** and **TOCITY** for the date that is specified by **FLIGHTDATE**. The solution for this use case is to create a REST API over the top of this procedure, then use IBM Watson services to add abilities to understand user input, and provide other useful information that is related to the search, such as weather and highlighted news that is related to the destination city. The use of IBM Watson services enhances the user experience and provides a better searching experience.

These three services are used for this solution:

- The Watson Natural Language Understanding (NLU) service, which processes user input, and extracts city names that are used in searching for flights.
- The Weather Company Data service, which looks up the destination city's weather data for the next three days.
- The Watson Discovery service, which obtains the top five news articles for the destination city.

7.2 The Flight400 application

This section provides more details about the Flight400 application, including how to take a service program and use it as REST API. You create and host the REST API on the IBM i system.

In this scenario, you use the sample application that is called Flight400, which has been used in other IBM Redbooks publications and IBM i presentations. The application is a simulation of a flight reservation system application that is developed in RPG, and may be used by airline agents to create, query, modify, and print flight reservations. For example, in a case where a reservation is received, an agent is required to manually enter necessary data into the system.

If you want to set up the application locally on your IBM i system, complete the following steps:

1. Create save files on your IBM i, as shown in Example 7-1.

Example 7-1 Creating save files

```
CRTSAVF FILE(QGPL/FLGHT400) TEXT('flight 400 (1 of 2)')
CRTSAVF FILE(QGPL/FLGHT400M) TEXT('flight 400 (2 of 2)')
```

2. Download the Flight400 program.

3. Decompress the file, and send the resulting files to the IBM i system by using FTP, as shown in Example 7-2.

Example 7-2 Decompressing the file and sending the resulting files to IBM i by using FTP

```
unzip flght400.zip
ftp <your-server>
ftp> bin
ftp> cd qgpl
ftp> put flght400.savf flght400
ftp> put flght400m.savf flght400m
```

4. Restore the libraries on your IBM i, as shown in Example 7-3.

Example 7-3 Restoring the libraries

```
RSTLIB SAVLIB(FLGHT400) DEV(*SAVF) SAVF(QGPL/FLGHT400)
RSTLIB SAVLIB(FLGHT400M) DEV(*SAVF) SAVF(QGPL/FLGHT400M)
```

5. Add libraries to the library list on your IBM i, as shown in Example 7-4.

Example 7-4 Adding libraries to the library list

```
ADDLIBLE FLGHT400
ADDLIBLE FLGHT400m
```

c. Run the application, as shown in Example 7-5.

Example 7-5 Running the Flight400 application

```
GO FLGHT400/FRSMAIN
```

6. The main menu opens (Figure 7-2), and you can try the application by using the menu.

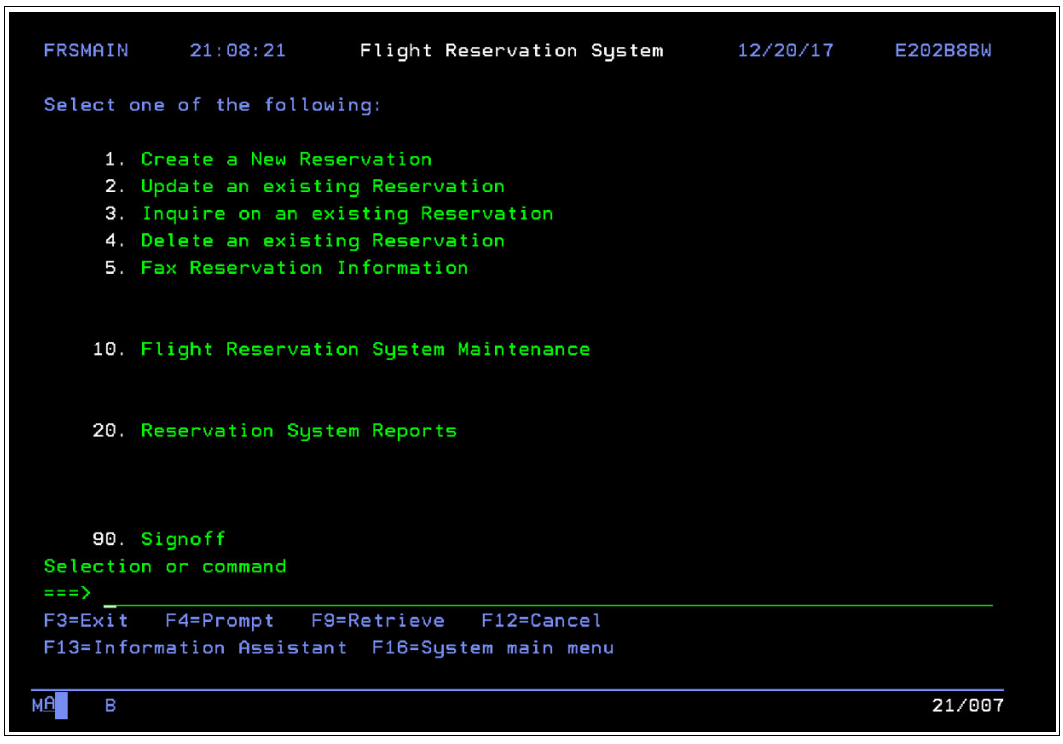

Figure 7-2 Main menu of the Flight400 application

7. Log on to the system.

   Before you do any activity, log on to the system. Use “mercury” as the password for all agents (Figure 7-3). This is a password that applies only to the FLGHT 400 application.

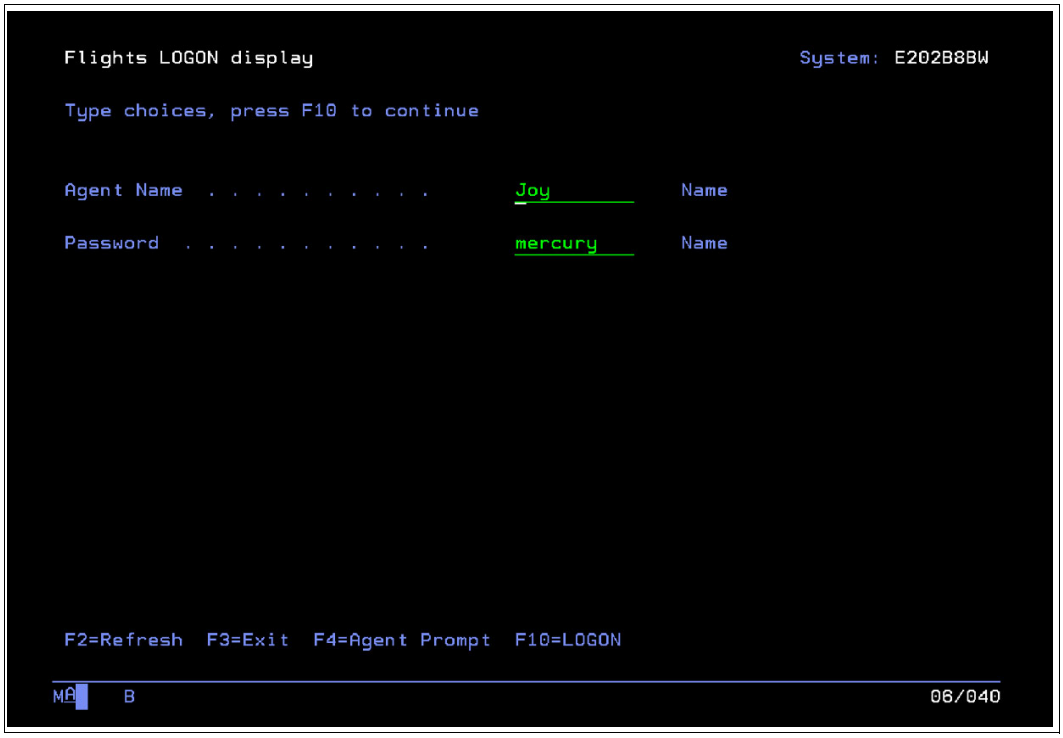

Figure 7-3 Log on to the system

8. Select F10 to continue navigating through records.

Figure 7-4 shows an example of a detailed order.

Figure 7-4 Detailed order example

7.3 The Watson services

This section describes t each Watson service that is used in this use case.

7.3.1 The Watson Natural Language Understanding service

This service helps you analyze text to extract metadata from content, such as concepts, entities, keywords, categories, sentiment, emotion, relations, and semantic roles. You can also use Watson Knowledge Studio to develop custom annotation models to identify industry, domain-specific entities, and relations in unstructured text.

This service is used in this use case to help extract information from the input text that the user provides in a natural language context to determine the flight locations for which the user is looking.

For more information about this service, see 2.1.6, “Watson Natural Language Understanding” on page 21.

7.3.2 Weather Company Data service

This service enables you integrate weather data from The Weather Company into your IBM Cloud application. You can retrieve weather data for an area that is specified by a geolocation. The data enables you to create applications that solve real business problems where weather has a significant impact on the outcome.

This service helps you get weather information about the destination location to enrich the search result.

For more about this service, see the Weather Company Data service at IBM Bluemix.

7.3.3 The Watson Discovery service

This service helps you add a cognitive search and content analytics engine to applications to identify patterns, trends, and actionable insights that drive better decision-making.

This service can securely unify structured and unstructured data with pre-enriched content, and use a simplified query language to eliminate the need for manual filtering of results.

This service is used in the use case to get the top news that is related to the destination location. This information enriches the user experience for the search results.

For more information about this service, see 2.1.3, “Watson Discovery” on page 14.

7.4 The orchestrating Findflights Node.js application

This section describes the Node.js application, which orchestrates the input and output between the Flight400 RPG application and the Watson services to provide a better searching experience.

Because the interaction between the front-end app and other services is through the HTTP web service, you have the flexibility to either deploy the front-end app on an IBM i system or in the IBM Cloud environment. To keep it simple for this use case, we made the application a Cloud Foundry Node.js application running on the IBM Cloud platform.

The front-end app is built by using an express framework, and provides plain text as a result to respond to requests from a simple web endpoint.

7.5 Putting it all together

Before you can pull everything together as an application in the IBM Cloud platform, you must create the REST API for the interactions with the system of record, which in this case is the back-end Flight400 RPG application. This section describes the process of creating a REST API over an ILE RPG application and making that API accessible on your IBM i system.

7.5.1 Creating a Findflight procedure as an API endpoint

To create an API endpoint out of the Flight400 application by using the IBM i IWS system, complete the following steps. For more information about the IWS server, see the IWS product page.

1. Access the IBM i Web Administration for i tool from a browser to create a Web Services Server. Sign on with your IBM i user and password. The Web Admin GUI requires you to have *ALLOBJ special authority.

   We realize this might be an issue for many companies because of their security policies. The good news is the Web Admin GUI also has a 'Permissions' support that enables an administrator to specify permission for a user that has no special authority. Once they have permission, a low-level user can create and manage web services server. Under the Advanced tab, click the permissions link in the left navigation to set permissions for a user. Specify a name for this web services server. Click **Next** to continue.

   With the correct policies in place, you can access the window that is shown in Figure 7-5.

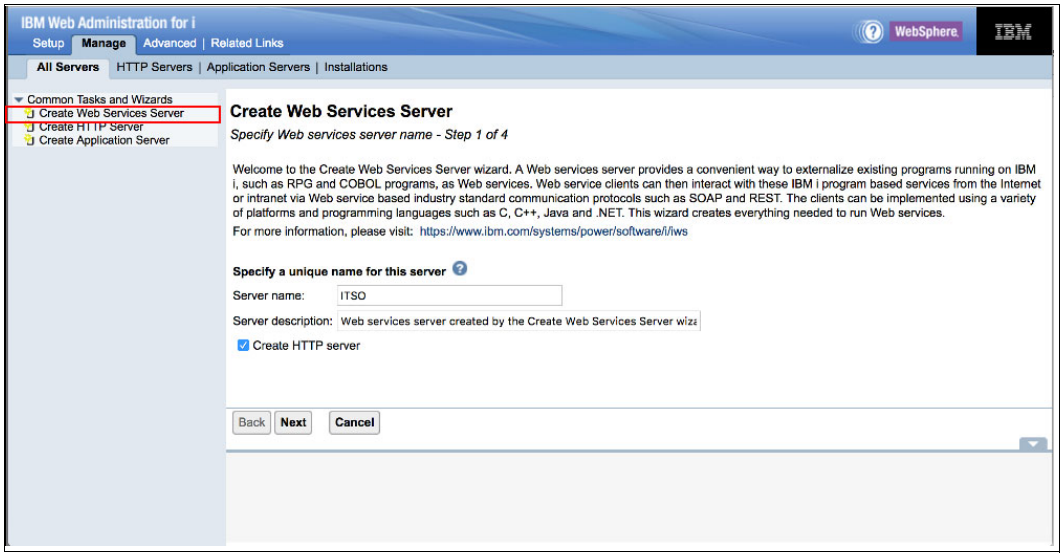

Figure 7-5 Create Web Services Server: First step

Note: The Uniform Resource Locator (URL) to IBM i Web Administration for IBM i is:

`http://<your.ibmi.i.server.ip.address.or.domain>:2001/HTTPAdmin`

Use your IBM i profile credentials to access the tool.

2. Create a Web Application Server. This server is what listens to the REST API call, and parses the input and output values between the web request and the back-end RPG application.

   The first requirement for the application server is the IP address and port on which it listens. By default, the IWS support specifies to listen on all IP addresses that are defined to this IBM i system and listen on a default port that the wizard selects. The wizard reviews the IBM i system and selects a port range that is not being used by anything else, which ensures that the server starts and works correctly initially. It is easy to change to a different IP address or port later. Click the **View HTTP server** link, and start the Change HTTP Port wizard to update at any time. Specify the addresses and ports for the server, as shown in Figure 7-6.

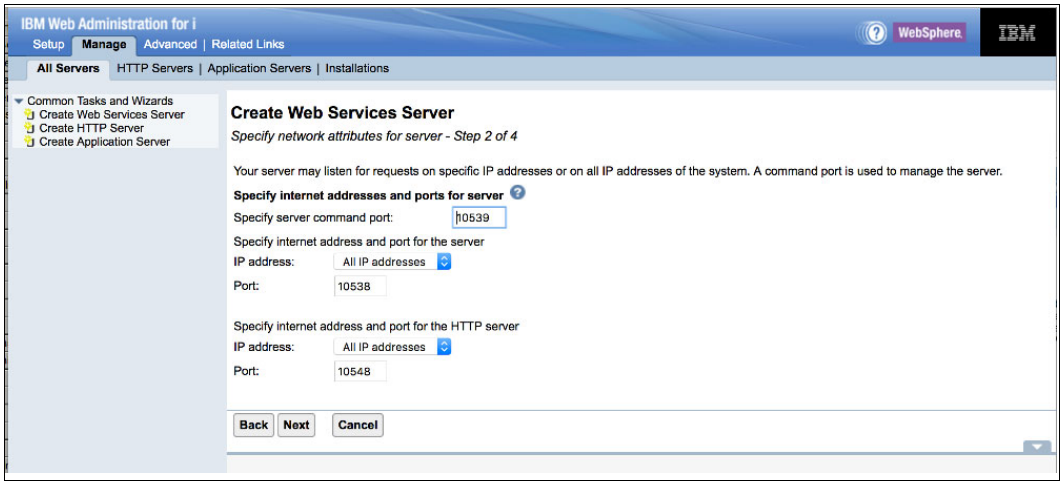

Figure 7-6 Specifying addresses and ports for the server

3. Specify the user ID for the server.

   The application server is just another job on the IBM i system. Every job requires a user profile that is used for this job. This is the user ID that is used to run the job for the application server.

   Use a profile that has no special authorities. A preferred security practice is to grant the user only the minimal authority that is needed. This user ID is granted the authority to all of the server objects, such as files and directories by the wizard.

   Specify a special user ID to run only the server job. The integrated web service server requires that the IBM i user ID status is set to *ENABLED and have a password value other than *NONE. There are three options:

   a. Use the default user ID.

      Specifies that the system-supplied profile QWSERVICE is used to run the server jobs. The QWSERVICE profile is shipped with a status of *DISABLED and the password is set to *NONE. To use the default user profile, ensure that the user profile status is set to *ENABLED and the password is set to a value other than *NONE. If the password for the user profile is set to *NONE, certain functions of the web service do not run correctly. Use the **CHGUSRPRF** CL command to modify the setting for this user ID.

   b. Specify an existing user ID.

      Specifies a specific user ID on the system to run the server jobs.

c. Create a new user ID.

Enables administrators of this wizard to create a user ID to run the server jobs without having to start a new 5250 session. This option is available only when the user that is signed on to this Web Admin GUI session has *SECADM special authority.

Click **Next** to continue (Figure 7-7).

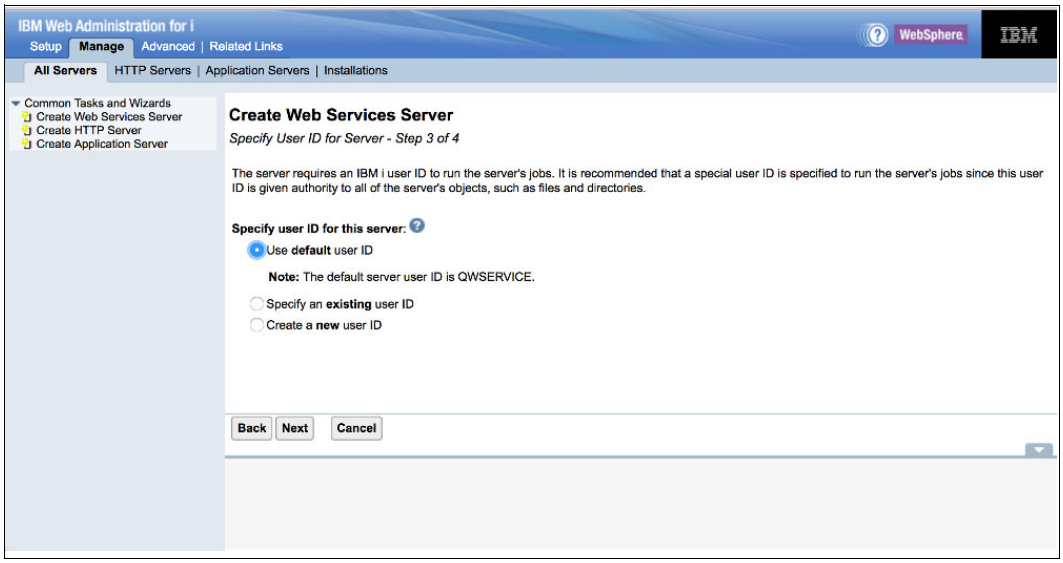

Figure 7-7 Specifying a user ID for the server

4. Review the information, and click **Finish** to complete the process (Figure 7-8).

The wizard creates both an HTTP Apache server and the Integrated Web Application Server that is used to contain all the web services that you want to create.

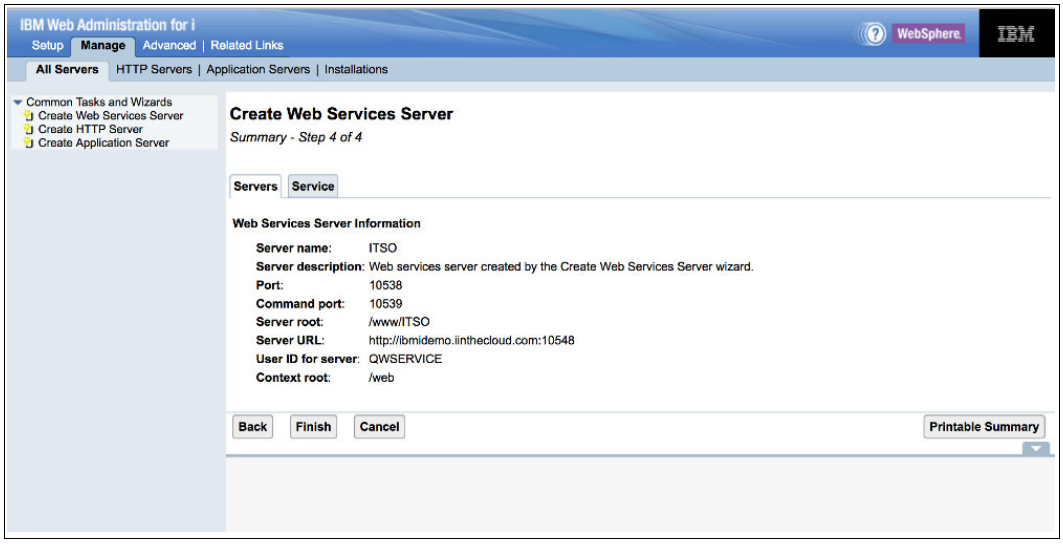

Figure 7-8 Review summary

5. After the server is created, it automatically starts. After it is running, you should see something like Figure 7-9. Otherwise, you might need to go back to the previous steps to verify the input information, or do troubleshooting.

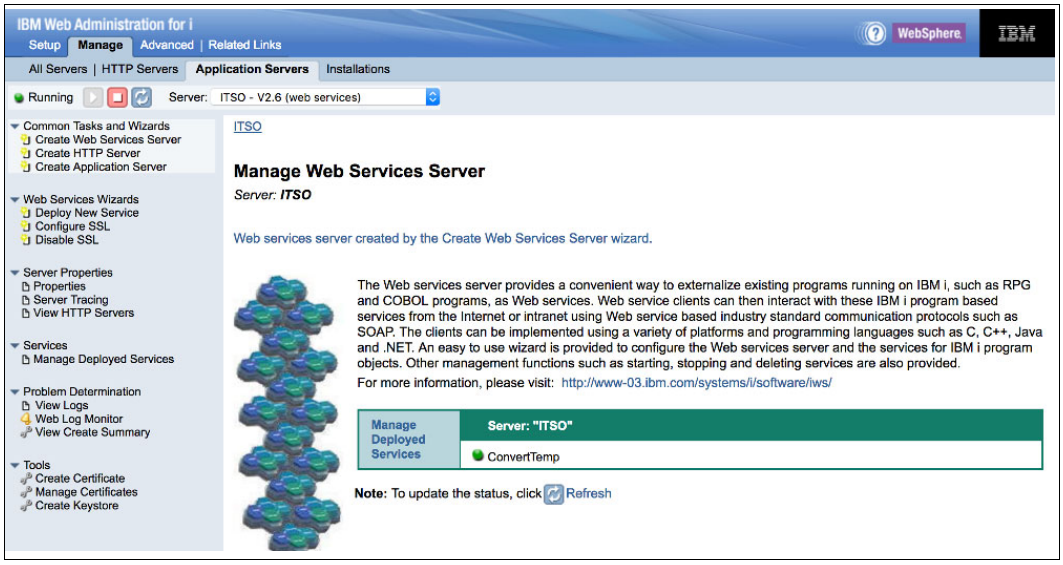

Figure 7-9 Server is created and ready to use

6. Now that your server is created and running, click **Deploy New Service** in the left navigation tree to start the wizard to proceed with the steps that are needed to create a web service (Figure 7-10).

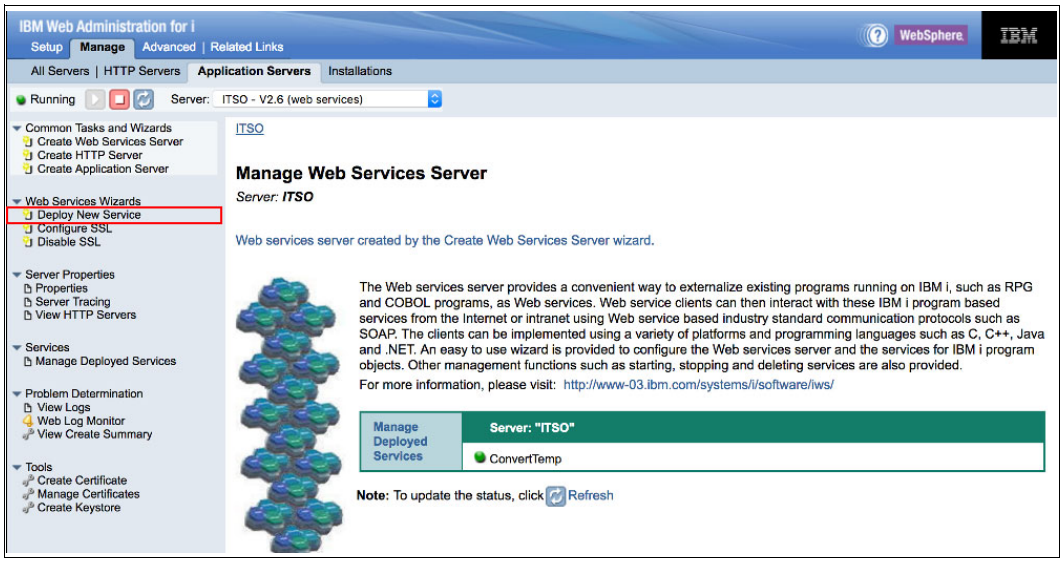

Figure 7-10 Deploy New Service

7. The IWS server supports the creation of both SOAP- and REST-based web services. Both can be deployed to a single server. You can create both SOAP and REST services for the same back-end program.

   Specify the web service type as **REST** for this example.

Click **Next** to continue (Figure 7-11).

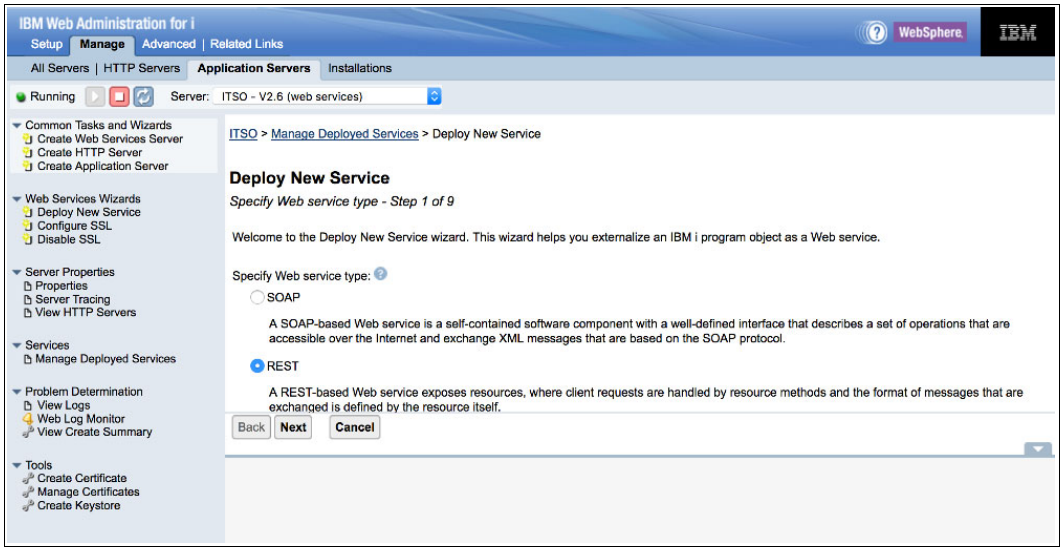

Figure 7-11 Specifying the REST web service type

8. The IWS support uses the compiled ILE object for creating the web service. The program must be a stateless, loosely coupled program that takes input, runs a task, and returns output. It must be compiled with the PCML option turned on, which likely was handled if the program object was compiled recently because compile with PCML is the default.

   The compiler creates a special object that is stored inside the compiled object. This PCML object defines the input and output parameters for the program. This is the support that the wizard uses to understand the signature for the ILE object. Specify the program object for which to create web service.

   Click **Next** to continue (Figure 7-12).

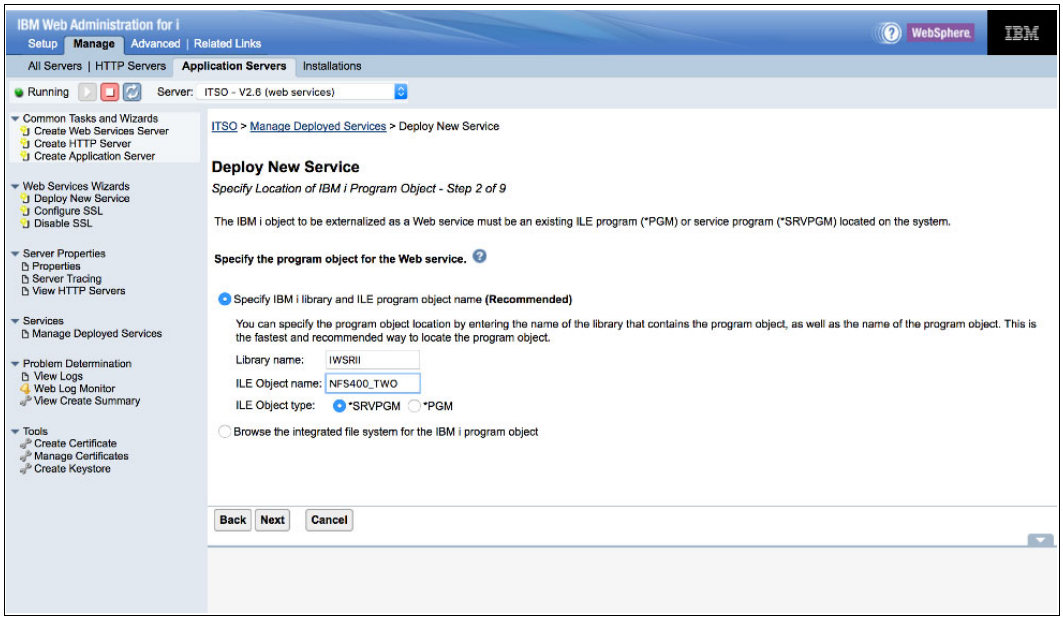

Figure 7-12 Specifying the program object for which to create the web service

9. Each web service that is created for this server must have a unique name. The important part is the Uniform Resource Identifier (URI) path value, especially when creating a REST API. The user can specify the URI identifier for the API and the Path parameter values.

   When you create a REST API, you have a web-based request that is sent to the IBM i system, and the web service must parse that web request and call the back-end ILE program. There are many ways that input values can be passed, and one common method is a Path Parameter value, where the input that is passed to the back-end program is passed as a parameter on the HTTP path.

   In our example, we pass the input as a path value. Specify a name for the service and define the URI path, as shown in Figure 7-13. Click **Next** to continue.

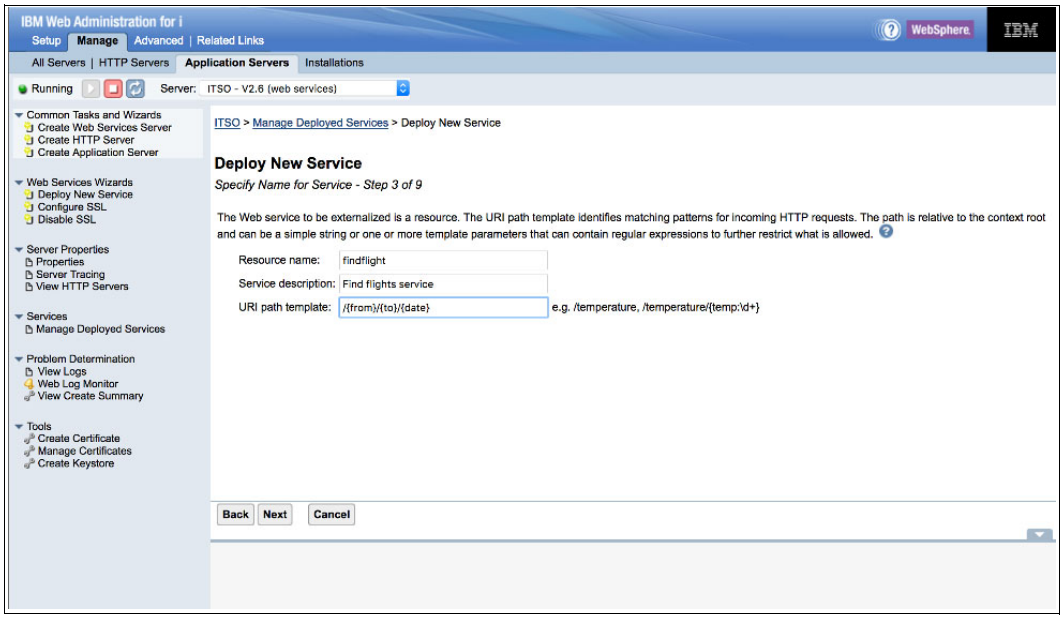

Figure 7-13 Naming the service and defining the URI for the template

10. Now, the wizard has read the PCML object for the compiled program or service program. All the programs and procedures that were found along with the input/output parameters for each one are displayed. When you create a REST API, connect *only* a single back-end procedure to a web request. This action simplifies support. Select **Export Procedures** to externalize a procedure as a web service.

    For this use case, we export only the FINDFLIGHTS procedure with three input parameters and two output parameters. Click **Next** to continue (Figure 7-14).

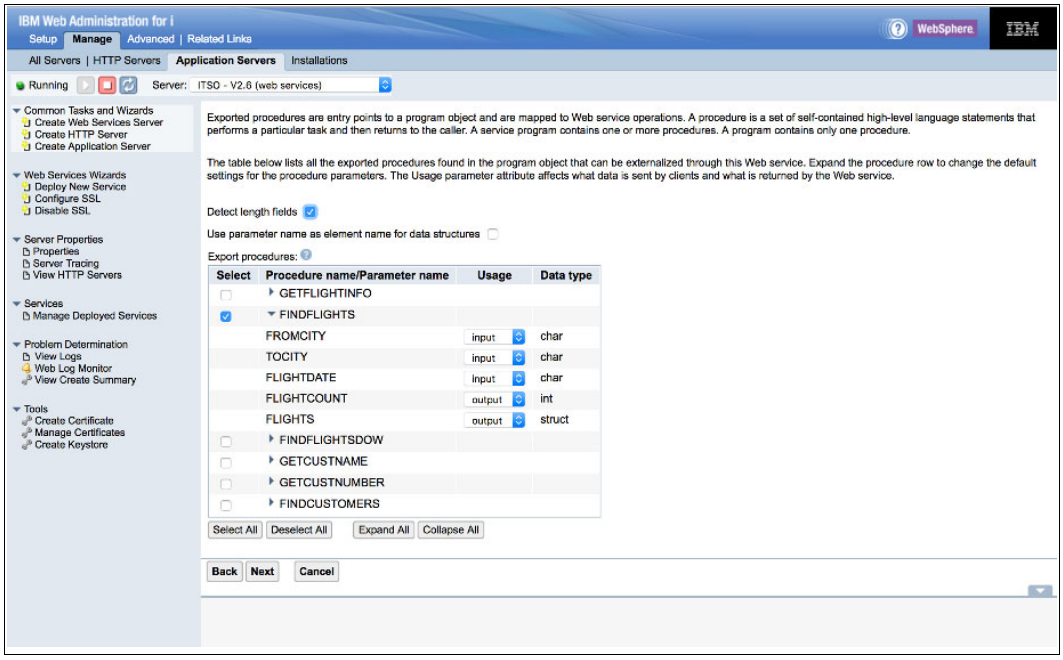

Figure 7-14 Selecting Export Procedures to externalize a procedure as a web service

11. This step in the wizard is the most important step when creating a REST API. You define the HTTP method that is used for this request: **`GET`**, **`POST`**, **`PUT`**, or **`DELETE`**. You also define the output format, either XML or JavaScript Object Notation (JSON). You must map the input values of the web request to the back-end program/procedure input values. There are many different places that the input can be passed on to the web request: path parm, matrix value, form field, HTTP header, or cookie.

The bottom table in Figure 7-15 is where the wizard identifies the input values for this procedure and you specify where the input values come from. The wizard supports setting a default value. Click **Specify resource method information**, and map the input parameters. Configure the parameters as shown in Figure 7-15. Click **Next** to continue.

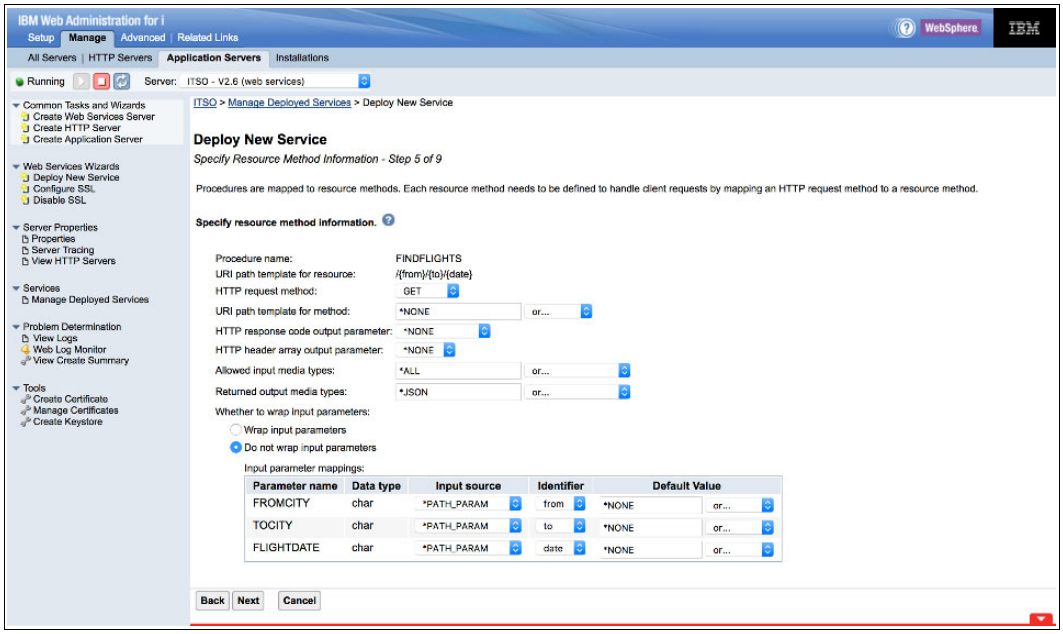

Figure 7-15 Specifying resource method information and map the input parameters

12. When a web service is called, a new job is started that is used to call the ILE program/procedure. This a pre-start job that is already running, but this job requires a user profile to be assigned to it. Specify a user profile that is used when running this ILE program. Do not specify a profile that has elevated authority or any special authorities. Specify a user ID for the service. Click **Next** to continue (Figure 7-16).

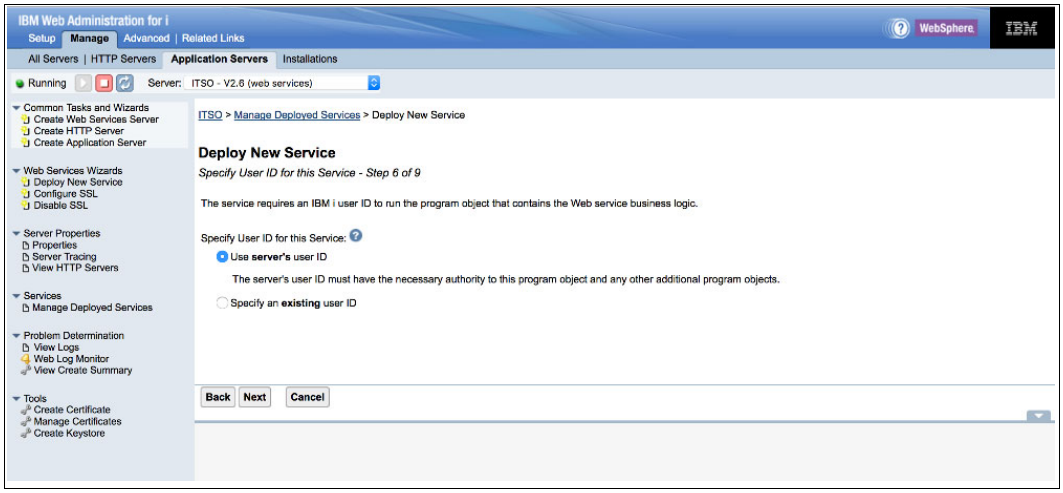

Figure 7-16 Specifying the user ID for the service

13. When a job on the IBM i system runs, it uses a library list to find the various resources that it needs. If something that is required is not found in the library list, the request fails, which includes other programs, databases, and other important objects. A list can be useful when you develop code because you can have your test database in one library, and change the library list later to point to your production database when you are ready.

    This setup can simplify the management and administration of your web services artifacts.

    Add a new library that is named FLGHT400 in to the library list. Click **Next** to continue (Figure 7-17).

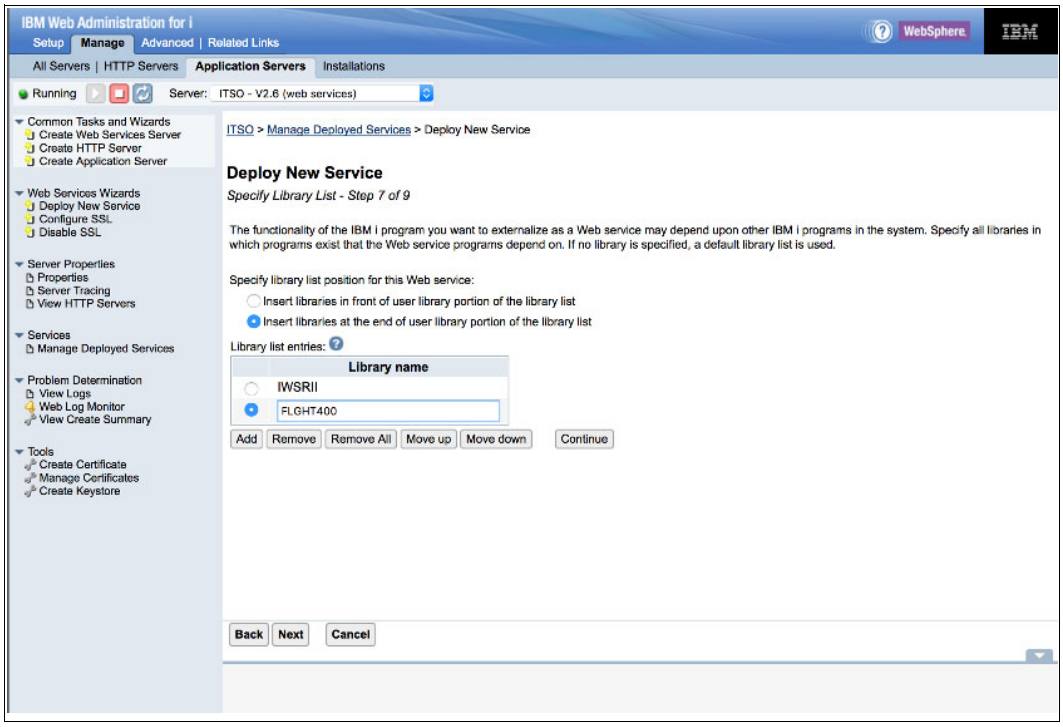

Figure 7-17 Adding a new library that is named FLGHT400 in to the library list

14. One of the more powerful features that is available is passing the HTTP header and transport information directly to your ILE program. As you move forward with using services, you have the ability for the back-end RPG to have the information that it needs in the web-based environment.

For this example, we are not using the transport or HTTP header values. You can keep them as the default, as shown in Figure 7-18. Click **Next** to continue.

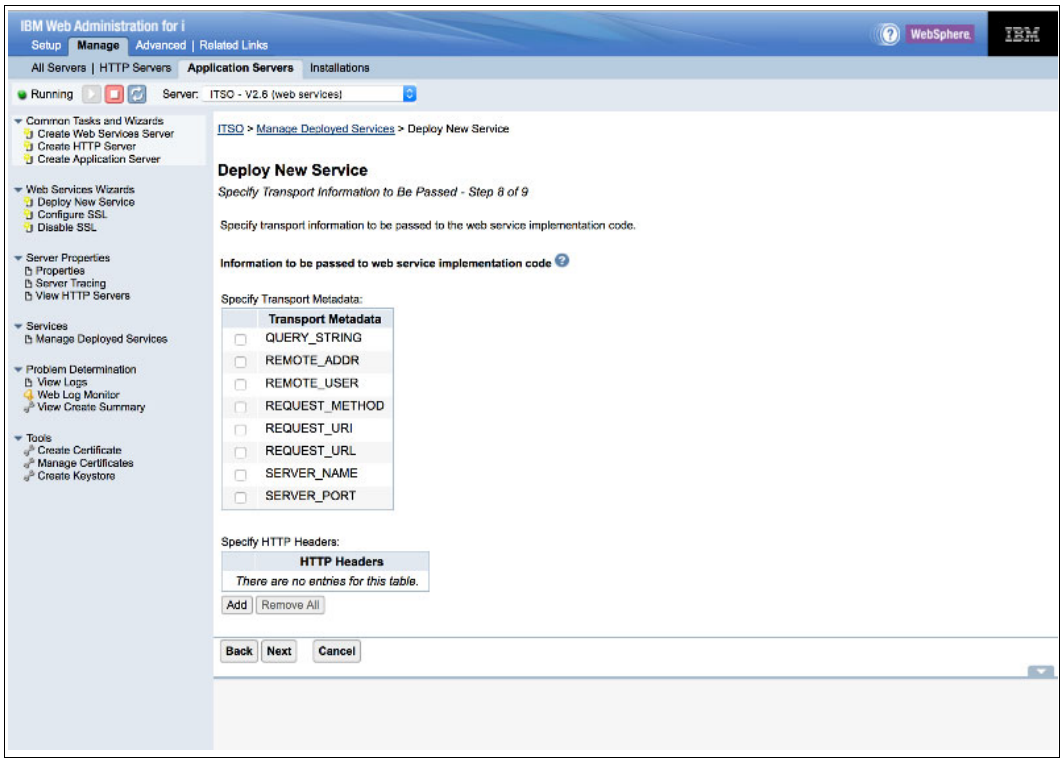

Figure 7-18 Specifying the transport information

15. Review and confirm the summary information and select **Finish** to complete the process (Figure 7-19). The service installation takes few moments to compete, and you see that the service is installed and ready after the installation completes.

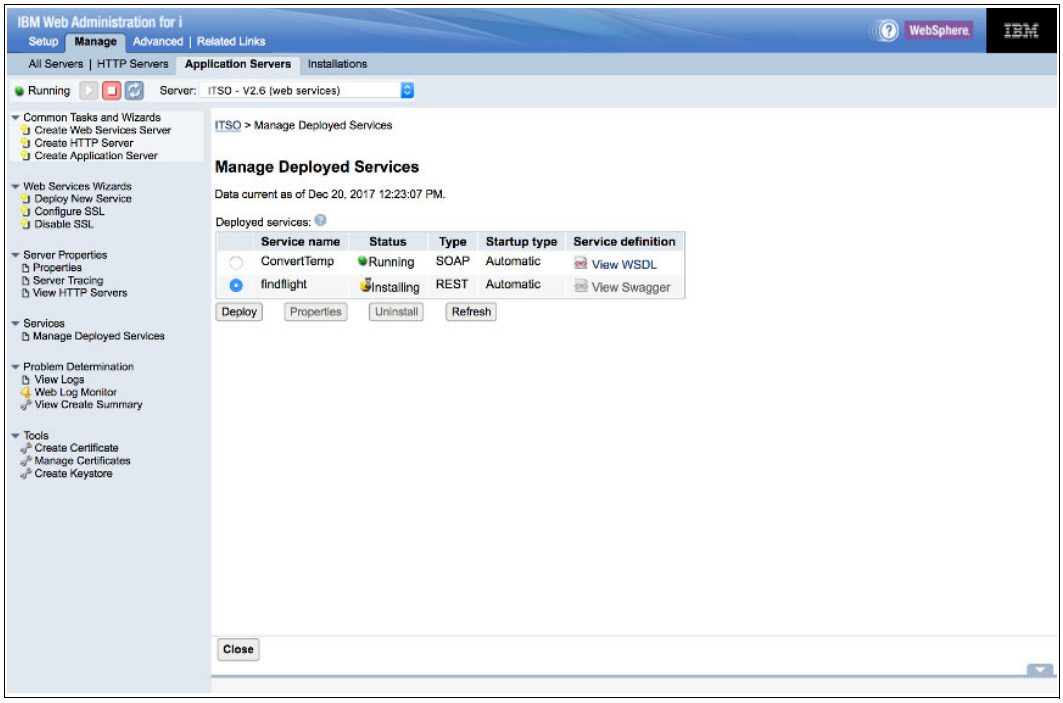

Figure 7-19 Service installation is complete

The REST service endpoint is ready to be used. To test the web service, use any client tool, such as Postman, SOAPUI, or HTTP Requester. In this example, we simply use a CURL command to verify the web service. The format of the URI is as follows:

```
http://<your.ibmi.i.address.or.domain.name>:<configured_port>/web/services/findflight/{From}/{To}/{Date}
```

For this example, you must run the command from your favorite shell. You can use qsh on your IBM i system. Change the host name and port value in the example to use the one that you just created. The example URL is as follows:

```
curl
http://common1.iinthecloud.com:10548/web/services/findflight/Toronto/Montreal/08032017
```

Example 7-6 shows the output from the query.

Example 7-6 Representational State Transfer query output example

```
{
  "FLIGHTCOUNT": 9,
  "FLIGHTS": [{
    "AIRLINE": "NWA",
    "FLIGHT": "5141604",
    "DOW": "Fr",
    "DEPARTCITY": "TOR",
    "ARRIVECITY": "MTL",
    "DEPARTTIME": "07:33 AM",
    "ARRIVETIME": "09:33 AM",
```

```
    "PRICE": "169"
  }, {
    "AIRLINE": "NWA",
    "FLIGHT": "5241605",
    "DOW": "Fr",
    "DEPARTCITY": "TOR",
    "ARRIVECITY": "MTL",
    "DEPARTTIME": "08:36 AM",
    "ARRIVETIME": "10:36 AM",
    "PRICE": "179"
  }, {
    "AIRLINE": "NWA",
    "FLIGHT": "5341606",
    "DOW": "Fr",
    "DEPARTCITY": "TOR",
    "ARRIVECITY": "MTL",
    "DEPARTTIME": "09:38 AM",
    "ARRIVETIME": "11:38 AM",
    "PRICE": "199"
  }]
}
```

Note: For the current updates and enhancements and details for deploying web services on IBM i, see the IWS for IBM i product page.

7.5.2 Creating Watson services on the IBM Cloud platform

To create a Watson service, you must have an IBM Cloud account. If you do not have one, create one by following the instructions in 5.1, “Signing up for an account at IBM Cloud” on page 84 before moving forward with this part. Then, complete the following steps:

1. Create the NLU service:
   a. Log in to the IBM Cloud platform and create the NLU service from the catalog (Figure 7-20).

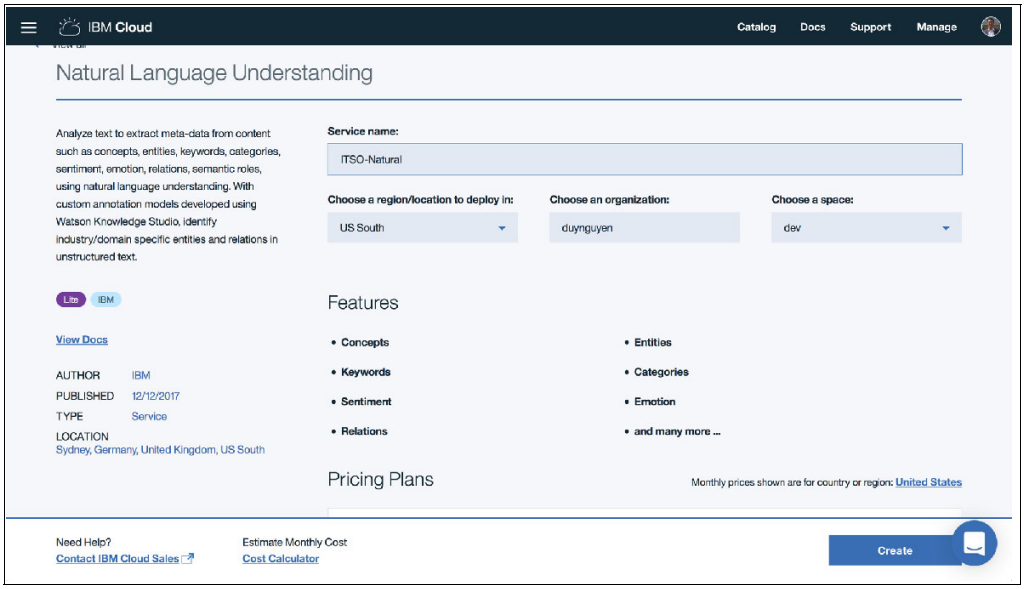

Figure 7-20 Creating the Watson NLU service

   b. Add a new credential to access the service.
   c. View and record the credential to be used in the application later.

2. Create the Weather Company Data service:

   a. Log in to IBM Cloud platform and create the Weather Company Data service from the catalog (Figure 7-21).

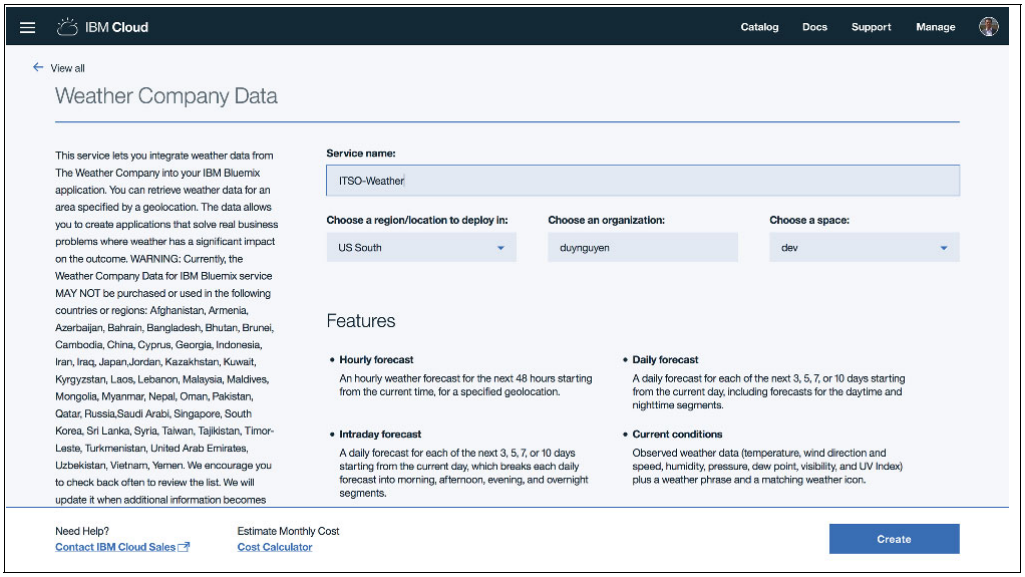

Figure 7-21 Creating the Weather Company Data service

   b. Add a new credential to access the service (Figure 7-22).

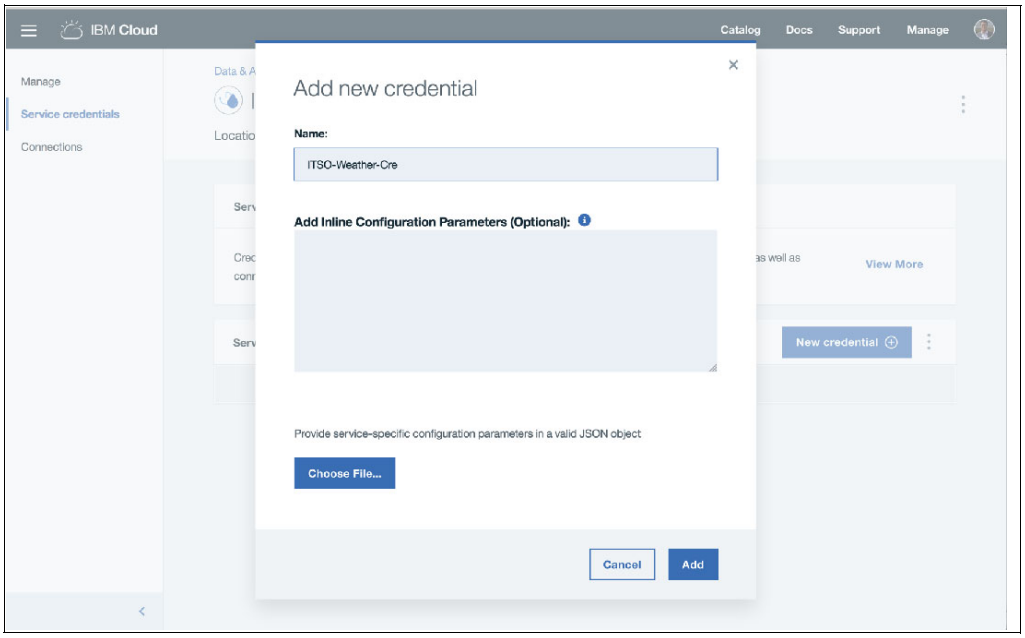

Figure 7-22 Adding a weather service credential

c. View and record the credential to be used in the application later (Figure 7-23).

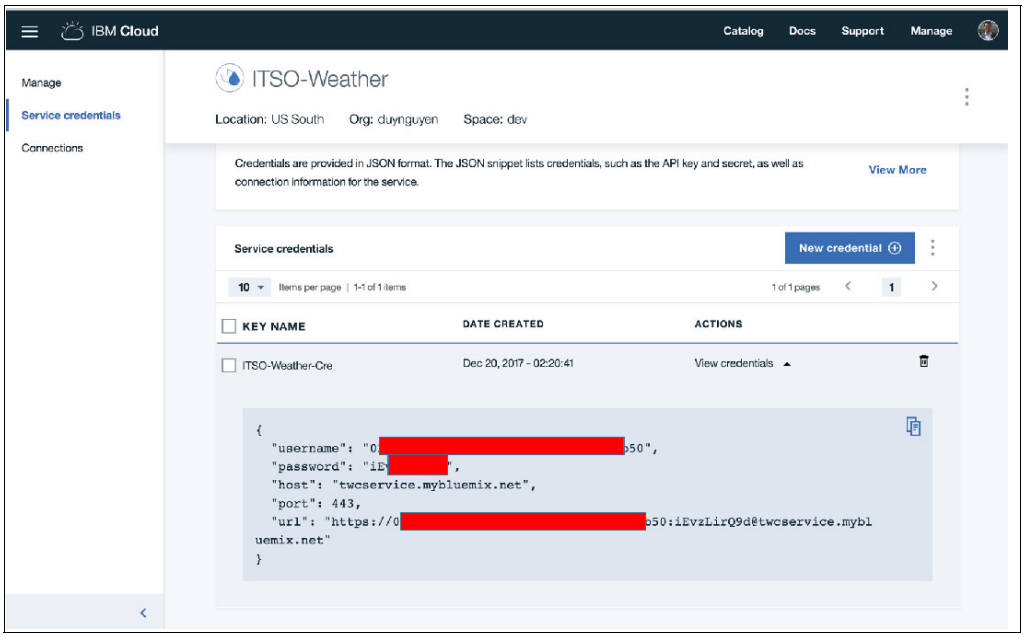

Figure 7-23 Viewing the credential

Note: The Weather Company Data service requires latitude and longitude to obtain weather information about a particular place. The API endpoint is in this format:

`https://<username>:<password>@twcservice.mybluemix.net:443/api/weather/v1/geocode/<latitude>/<longitude>/forecast/daily/xday.json`

The x is the number of future days for which you want to obtain weather information.

In this example, we use the Google API to convert a certain address into specific latitude and longitude pair. An API key to access Google API can be obtained from Google Maps.

3. Create the Watson Discovery service:

   a. Log on to the IBM Cloud platform, allocate the Watson Discovery service from the catalog, and create a service instance (Figure 7-24).

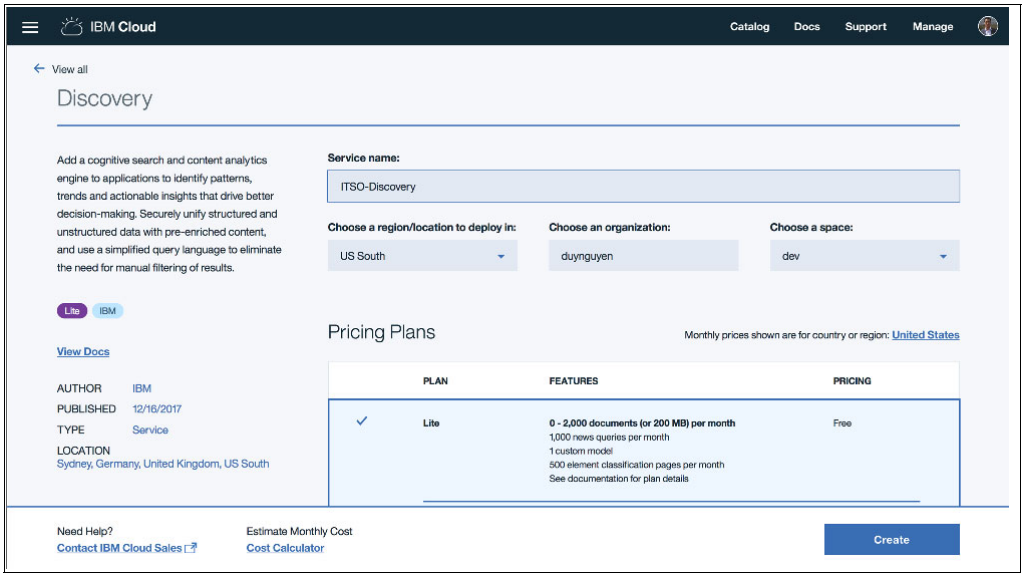

Figure 7-24 Creating the Watson Discovery service

   b. Add a new credential to access the service (Figure 7-25).

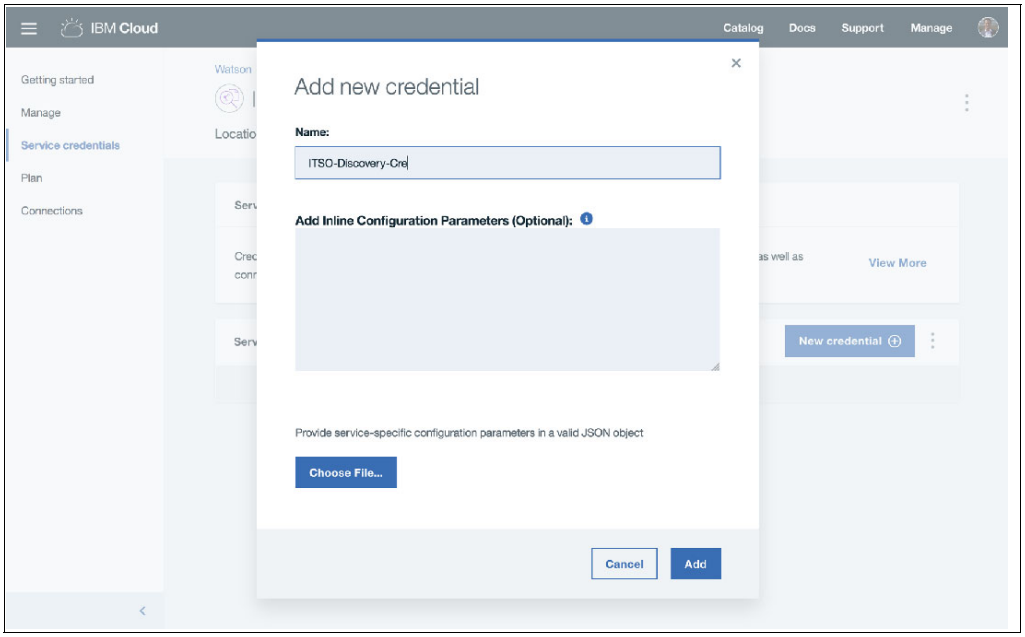

Figure 7-25 Adding a new credential to access the service

c. View and record the credential to be used in the application later (Figure 7-26).

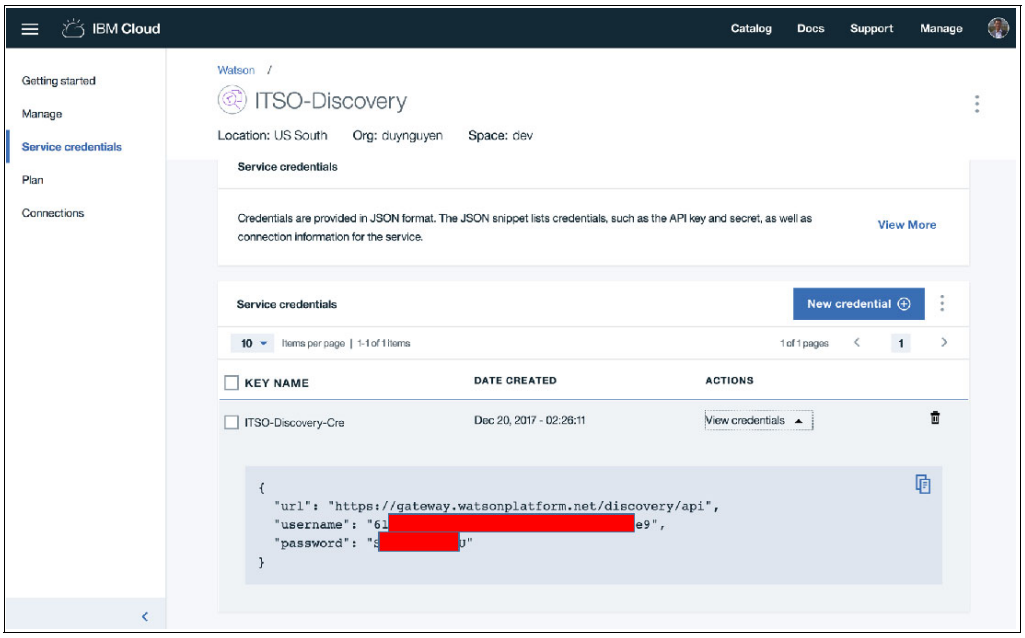

Figure 7-26 Viewing and recording the credential to be used in the application later

7.5.3 Creating the Findflight Node.js application

The following steps help to create the example Findflight Node.js application that coordinates all interactions with the IBM i IWS REST API and IBM Watson services to respond to a flight query and presents the results on a webpage:

1. Clone the example application from GitHub by running the following command:

   ```
   git clone https://github.com/ibm-redbooks-dev/flights-finder-power.gits
   ```

2. Provide the credentials to access the dependencies web services.

   After you have the source code cloned to your local workstation, open the `env.properties` file (Example 7-7) in the project root and input your services' credentials that you obtained in 7.5.2, "Creating Watson services on the IBM Cloud platform" on page 150.

Example 7-7 Providing the services' credentials

```
SECRET="secret123"

# google map api key
GOOGLE_MAP_API_KEY="<put-your-credentials-here>"

# watson configuration
WATSON_LANG_USER_ID="<put-your-credentials-here>"
WATSON_LANG_USER_PWD="<put-your-credentials-here>"
WATSON_LANG_VERSION_DATE="2017-02-27"

WATSON_DISC_USER_ID="<put-your-credentials-here>"
WATSON_DISC_USER_PWD="<put-your-credentials-here>"
WATSON_DISC_VERSION="v1"
WATSON_DISC_VERSION_DATE="2017-08-01"

# flight services, update with your IBM i webservice endpoint here
```

```
FIND_FLIGHT_URL="http://common1.idevcloud.com:10076/web/services/findflight"

# bluemix WEATHER
BLUEMIX_WEATHER_URL="https://twcservice.mybluemix.net/api/weather"
BLUEMIX_WEATHER_USER_ID="<put-your-credentials-here>"
BLUEMIX_WEATHER_USER_PWD="<put-your-credentials-here>"
```

3. Create a Cloud Foundry Node.js application in the IBM Cloud platform.

   Log in to the IBM Cloud platform, search for the Node.js SDK from the catalog, and create an application (Figure 7-27). Follow the instructions there to obtain the necessary CLI tools to deploy the app later.

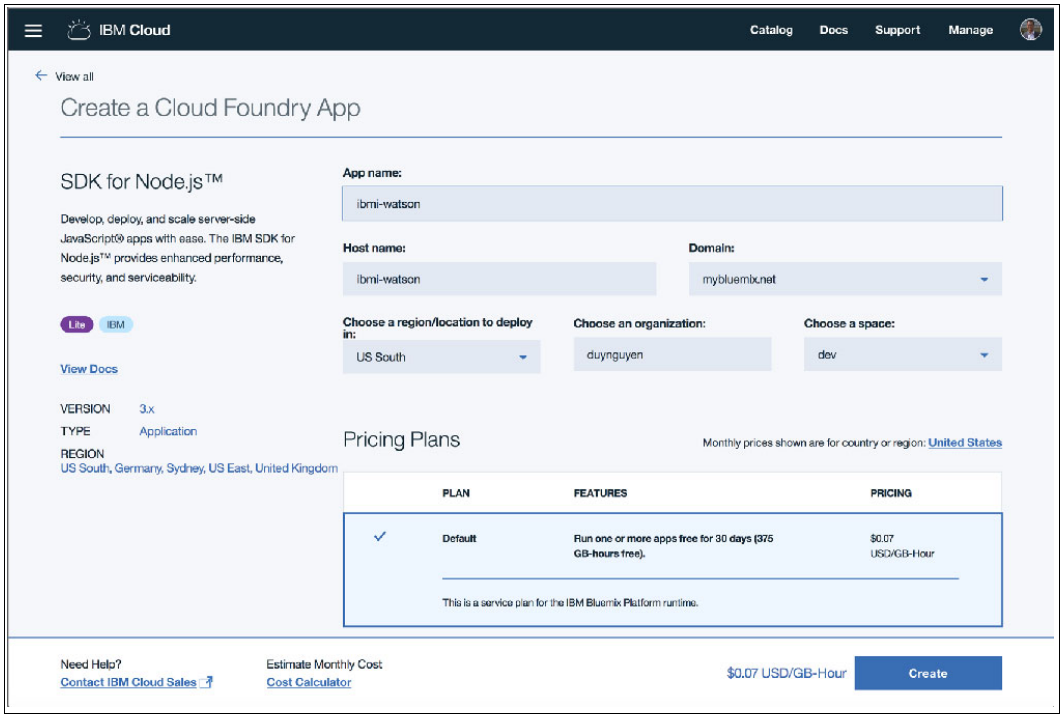

Figure 7-27 Creating a Cloud Foundry Node.js application in IBM Cloud

Note: Record the application name because you need it later to configure the metadata configuration file for the app before deploying it to IBM Cloud.

4. Provide the application configuration to deploy the app to IBM Cloud.

   Open the `manifest.yml` file in the project root, and input your application information, as shown in Example 7-8.

Example 7-8 Configuring the application deployment metadata

```
applications:
 - name: put-your-application-name-here
   random-route: true
   memory: 256M
```

5. Deploy the application on IBM Cloud by running the command that is shown in Example 7-9.

Example 7-9 Deploying the application to IBM Cloud

```
cf api api.ng.bluemix.net
cf push
```

If the application is deployed successfully, you see output that is similar to Example 7-10.

Example 7-10 The application is successfully deployed

```
0 of 1 instances running, 1 starting
1 of 1 instances running

App started

OK

App ibmi-watson was started using this command `./vendor/initial_startup.rb`

Showing health and status for app ibmi-watson in org duynguyen / space dev as
<your-email-goes-here>...
OK
requested state: started
instances: 1/1
usage: 256M x 1 instances
urls: ibmi-watson.mybluemix.net
last uploaded: Fri Dec 22 16:34:22 UTC 2017
stack: cflinuxfs2
buildpack: SDK for Node.js(TM) (ibm-node.js-6.12.0,
buildpack-v3.16.1-20171207-1520)

     state     since                    cpu    memory      disk      details
#0   running   2017-12-22 11:36:27 AM   0.0%   0 of 256M   0 of 1G
```

6. Test the application.

You now can test the application by sending requests to the following endpoint:

`https://your-application-name/findflight`

Use the following input parameters:

- **`searchtext`**: An input text that describes your query.
- **`date`**: The date on which you querying for flights.

Example 7-11 shows this use case's query.

Example 7-11 Query example

```
https://ibmi-watson.mybluemix.net/findflight?searchtext="Flights from Houston
to Chicago"&date=2017-12-22
```

The application provides list of flights according to your query, together with weather information about the destination in the next three days and top news about the destination (Example 7-12).

Example 7-12 Flight information example


```
{
  "fromCity": "Houston",
  "toCity": "Chicago",
  "date": "2017-12-22T00:00:00.000Z",
  "flights": {
    "FLIGHTCOUNT": 9,
    "FLIGHTS": [{
      "AIRLINE": "DLT",
      "FLIGHT": "5170956",
      "DOW": "Fr",
      "DEPARTCITY": "HOU",
      "ARRIVECITY": "CHG",
      "DEPARTTIME": "08:01 AM",
      "ARRIVETIME": "10:01 AM",
      "PRICE": "169"
    }, {
      "AIRLINE": "DLT",
      "FLIGHT": "5270957",
      "DOW": "Fr",
      "DEPARTCITY": "HOU",
      "ARRIVECITY": "CHG",
      "DEPARTTIME": "09:03 AM",
      "ARRIVETIME": "11:03 AM",
      "PRICE": "179"
    }, {
      "AIRLINE": "DLT",
      "FLIGHT": "5370958",
      "DOW": "Fr",
      "DEPARTCITY": "HOU",
      "ARRIVECITY": "CHG",
      "DEPARTTIME": "10:06 AM",
      "ARRIVETIME": "12:06 PM",
      "PRICE": "199"
    }, {
      "AIRLINE": "DLT",
      "FLIGHT": "5470959",
      "DOW": "Fr",
      "DEPARTCITY": "HOU",
      "ARRIVECITY": "CHG",
      "DEPARTTIME": "11:09 AM",
      "ARRIVETIME": "01:09 PM",
      "PRICE": "199"
    }, {
      "AIRLINE": "DLT",
      "FLIGHT": "5570960",
      "DOW": "Fr",
      "DEPARTCITY": "HOU",
      "ARRIVECITY": "CHG",
      "DEPARTTIME": "12:55 PM",
      "ARRIVETIME": "02:55 PM",
```

```
      "PRICE": "299"
    }, {
      "AIRLINE": "DLT",
      "FLIGHT": "5670961",
      "DOW": "Fr",
      "DEPARTCITY": "HOU",
      "ARRIVECITY": "CHG",
      "DEPARTTIME": "02:01 PM",
      "ARRIVETIME": "04:01 PM",
      "PRICE": "299"
    }]
}
```

Note: We provide only a basic HTTP endpoint to show the example and the raw data coming back for the query so that the reader decides how they want to display the data based on the output.

Figure 7-28 shows an example of a modern Node.js user interface that we created that calls the REST-based web service, and incorporates the Watson APIs that we described.

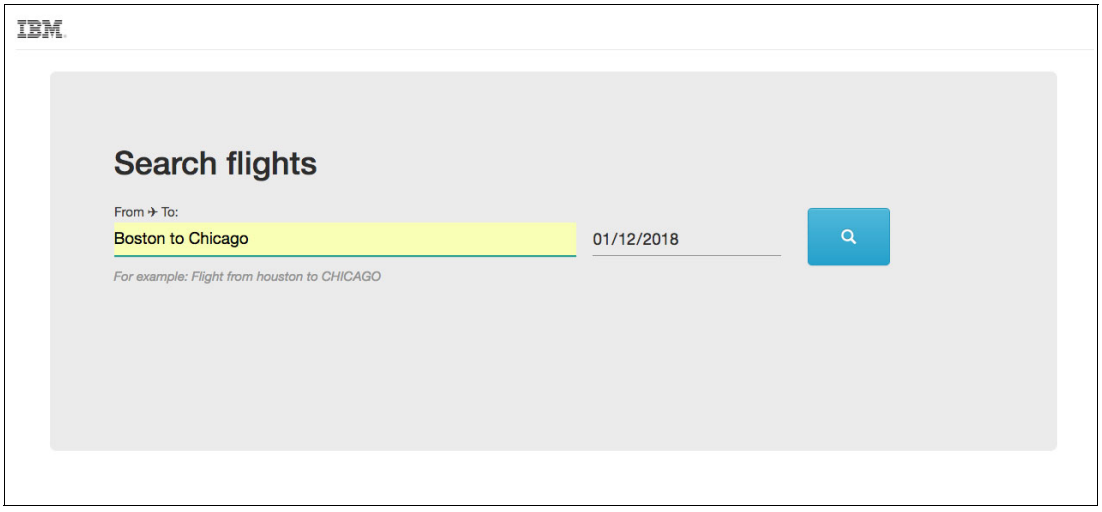

Figure 7-28 Findflights GUI Input page

This input page uses two interesting features:

- The From and To fields use the Watson service for language understanding. NLU can parse the words “Boston” and “Chicago” and convert them into the appropriate airport codes.
- The date field uses a calendar widget that is part of Node.js.

After you click the magnifying glass icon, the application calls the Rest web services for Findflights, which retrieves a list of possible flights that match the specified criteria (Figure 7-29).

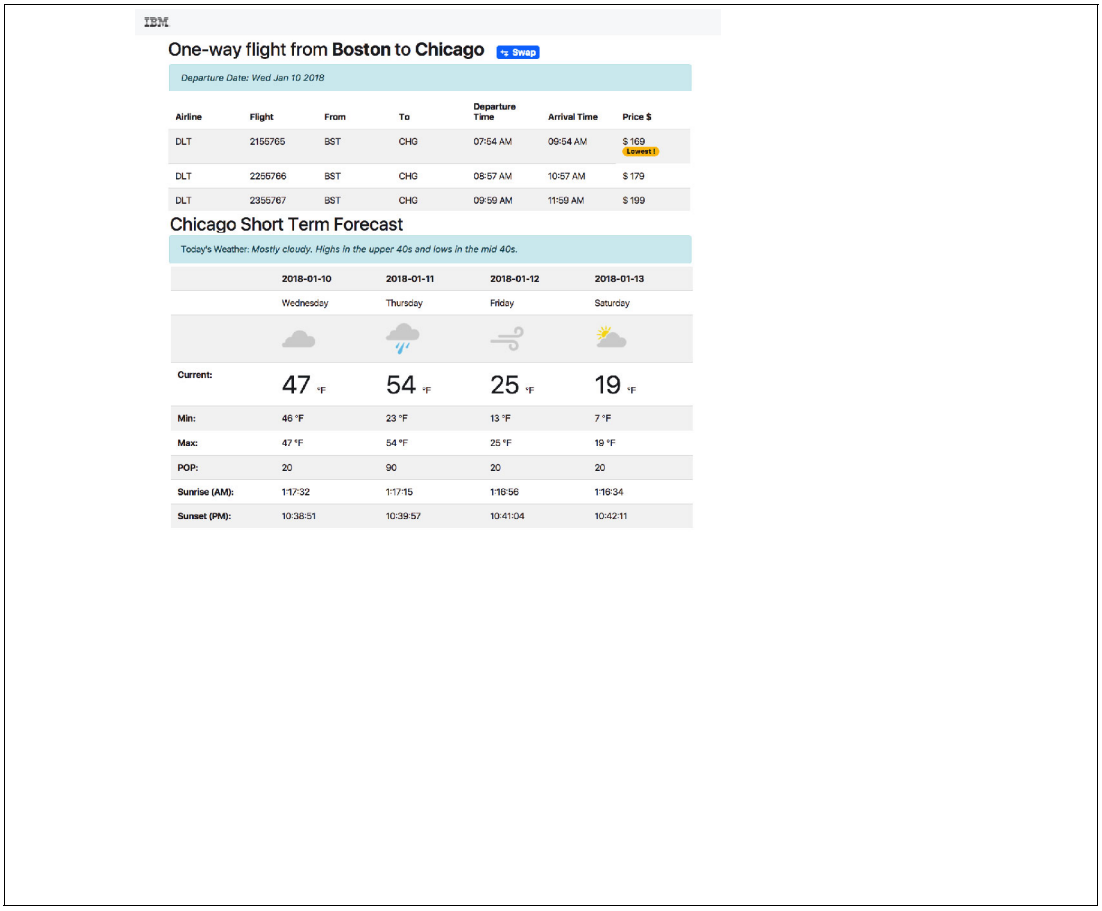

Figure 7-29 Findflights results page

In addition to displaying the flights information, we called the Watson weather API to retrieve the current weather information for the destination city.

8

Deploying the application into production

After your application is built and tested, you are ready to deploy it into production. You should not need to make any application changes to make it production ready. Design for application tuning, performance, availability, application maintenance, and system maintenance before you start coding.

In addition, the full software stack that your application depends on must be production ready. You must verify that the entire stack conforms to your production needs. You must set a Node.js runtime version and specify which library modules your developers may use. Your developers must produce code that is dependent only on your predefined run time and stack. When you deploy into production, you must manage the Node.js run time, ensuring that all applications that are running in the environment can run on the version of Node.js that you use by testing the application.

You must deploy your application with minimum intervention. You must provide Transport Layer Security (TLS) capability and be able to scale the deployment to satisfy peak usage demand.

After the application is in production, developer tools such as integrated development environments (IDEs), test harnesses, or the console are available to debug and correct any problems that might arise. Plan how you identify the root causes of application crashes, bottlenecks, performance issues, and race or concurrency conditions.

The chapter includes the following topics:

© Copyright IBM Corp. 2018. All rights reserved.

8.1 Standardized run time

Pick a version of Node.js that you want to use and ensure that all your applications run on this runtime version without problems. Almost all Node.js applications reuse modules that are created by the Node.js developer community and published to Node Package Manager (NPM), which generates a tree of dependencies. Some of these dependencies might be on modules that are *stale* and contain security vulnerabilities.

To reduce this risk, use tools that can check for vulnerabilities in modules that your application is both directly and indirectly dependent on. You should create a *white list* of NPM modules that your developers may use. Ensure that your standardized run time does not include deprecated or security vulnerable versions of modules. Plan to migrate away from a soon-to-be deprecated dependency stack before it is deprecated. Deprecated modules will not be fixed for security or performance issues.

When you must upgrade your Node.js runtime version, plan and control the migration. Verify which fixes and new and updated runtime features that you want available for your developers. Set up an environment where your developers can ensure that their applications continue to run in the new environment. Identify and apply all fixes that are needed for your applications to run on the new stack. Switch the production environment to the upgraded runtime and dependency stack only when all that necessary applications run without problems.

8.2 Designing for agile

Adapt an agile approach to your application development, build, test, and deployment environments. This approach enables you to deliver early and incrementally add features and improve your application as you receive feedback from the consumers of your application. Do not try to deliver the whole application in one go because you risk delivering functions that are redundant, not needed, or incorrectly specified.

A continuous delivery approach requires that you build run times and applications rapidly and automatically.

8.2.1 Application design

Ensure that the application design enables your developers to deliver functional code incrementally. Adapt a design that is flexible and does not need major modification to accommodate new features.

Decomposing your application into reusable services enables the following abilities:

- Share reusable components across multiple applications.
- Apply different scaling policies to each service appropriate to its usage profile.
- Develop, deploy, and maintain the services independently of the application.

8.2.2 Secrets

The application must not contain any hardcoded application secrets or service credentials. Any secrets or credentials should also be kept out of the source code repository. Secrets and credentials should be made available only to the application at run time through a configuration or settings file or through environment settings.

8.2.3 Testing

Test cases must be updated and validated for each incremental update. Tests should be on both existing and new code. Automate the testing process. All tests that are written for Node.js applications should run by using the **`npm test`** command, which enables the tests to be automated and scheduled to run after each code drop. Therefore, each code drop must contain updated tests.

Your testers must consider that applications that are written in interpreted languages such as Node.js can contain undetected syntax errors. To minimize the risk of application errors making it into production, ensure that the following tasks are met:

- Ensure 100% code coverage in test.
- Use code checking tools to identify coding style and syntax errors.
- Ensure that the code logic functions as expected.

8.2.4 Candidate release testing

In most cases, you group several updates into a candidate release. Your testers should do stress testing on the candidate application release and the system. Stress testing should be automatically run after a candidate release is identified. For a candidate release build to be successful, it must pass all tests, including stress tests. Stress tests should accomplish the following tasks:

- Perform load testing.
- Perform server crash testing.

8.2.5 Design for troubleshooting

Your developers should fix all issues that are identified in test. They have the failing test cases. They can reproduce and fix the issues.

Your application designers must consider how they debug any problems that might arise in a production environment without having to stop the application, if possible. It is much harder for developers to reproduce errors that occur only in production.

8.2.6 Application logging

Application logs can help pinpoint where and how an application is failing. Making log calls in an application can affect application performance. For Node.js applications, your developers should use Morgan with the debug module for logging and not the `console.log` file because then you can control when and which debug messages are logged. Application logging should be done as shown in Example 8-1.

Example 8-1 Sample application logging

```
app.listen(port, () => {
  debug(`Application is listening on port: ${port}`);
});
```

To turn on logging, set the **DEBUG** environment variable with a list of the modules that you want to log, as shown in Example 8-2.

Example 8-2 Application DEBUG option

```
DEBUG=redbook-app node index.js
```

In this case, the `redbook-app` module initializes debugging with the code snippet that is shown in Example 8-3.

Example 8-3 Application debug initialization

```
const debug = require('debug')('redbook-app');
```

Minimize logging to only those errors that are useful for debugging to achieve the following tasks:

- Identify the problem.
- Reproduce the problem.
- Fix the problem.
- Redeploy the application with the fix.

Strive for minimal downtime and impact on all the unaffected applications. Applications must be designed so that fixing them in production is a simple process.

Your developers can prevent application crashes by using techniques to catch, log, handle, and recover from exceptions so that the application keeps running while alerting application monitors of a problem. If the application can remain running while the root cause of the problem is identified, this situation reduces the perceived severity of the problem. A good technique is to wrap both synchronous and isochronous functions with *promises*. Example 8-4 shows an example of how to catch a coding error, which in this case is a missing function.

Example 8-4 Using promises to handle application errors

```
const express = require('express');
const app = express();
const debug = require('debug')('redbook-app');

const port = process.env.PORT || 3000;

function doSomething() {
  return Promise.resolve();
```

```
}

function doSomethingElse() {
  return Promise.resolve();
}

app.get('/dosomething', (req, res, next) => {
  doSomething()
    .then(() => {
      doSomethingElse();
    })
    .then(() => {
      doAnotherThing();
    })
    .then(() => {
      res.send('Hello Redbooks Reader');
    })
    .catch(next)
});

app.use((err, req, res, next) => {
  res.send('Something went wrong');
});

app.listen(port, () => {
  debug(`Application is listening on port: ${port}`);
});
```

8.2.7 Developer operations (DevOps)

Adapt a developer operations (DevOps) approach that accomplishes the following tasks:

- Build your runtime environment.
- Deploy your application.
- Monitor your application.
- Automatically restart if either the application or the server crashes.
- Reduce the number of outages.
- Enable you to react quickly to outages.

Figure 8-1 shows the DevOps lifecycle.

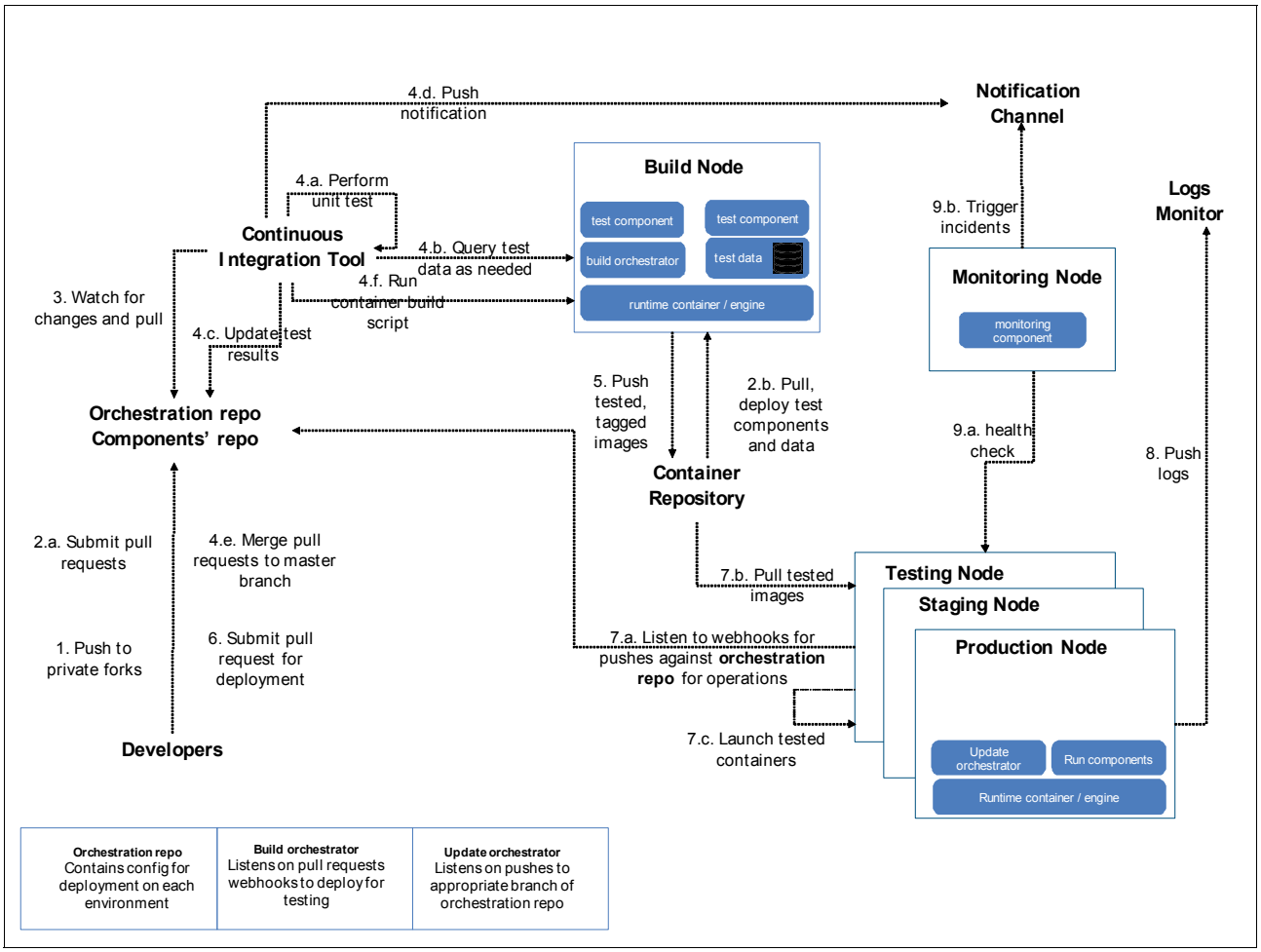

Figure 8-1 Developer operations lifecycle

The following list describes the steps in Figure 8-1:

1. A developer commits code changes and pushes updates to a private fork in the source code repository.
2. When the developer has makes all the changes, a pull request is created.
3. The continuous integration tool monitors for pull requests.
4. The continuous integration starts the following sequence of actions:
   a. Runs the unit tests.
   b. Uses the test data.
   c. Records the test results.
   d. Pushes notifications to inform the development leads of the success or failure of the build.
   e. In a successful unit test, runs the merge the code changes to the master branch.
   f. Builds the build run time containing the application with all dependencies.
5. Push the tested image containing the run time, application, and dependencies into the container repository.

6. After all the tests on the build node complete successfully, the lead developer submits a pull request for deployment.
7. The orchestrator triggers the following sequence of actions:
   a. Listens for deployment pull requests.
   b. Pulls a verified test image for rollout.
   c. Builds and starts the tested images for test, staging, and production.
8. Monitor the logs in test, staging, and production.
9. Actively monitor the application in test, staging, and production:
   a. Monitor the application and system health.
   b. Create notifications for any detected incidents.
10. Promote from test to staging to production when the relevant tests, including stress testing, are complete.

8.2.8 Production monitoring

Your production operators must monitor the application for the following items:

- Memory consumption
- CPU usage
- Bottlenecks
- Response times

Operators should use tools that enable dynamic reconfiguration of applications, for example, setting the `loglevel`. They should use tools such as **`logrotate`** to manage log files.

The monitor tools should raise alerts when they happen, enabling you to handle the situation before it becomes critical.

8.3 AIX system setup

This section provides information about how to set up the tools that are needed for log management, performance statistic, and monitoring tools on AIX.

8.3.1 Log management

By default, an AIX system does not perform “syslogging”. The default installation on an AIX system does not place entries in `/etc/syslog.conf`.

When comparing the syslogd on AIX to other UNIX like platforms, (such as Linux), you notice that `<facility>.*` does not work. So, make sure that your priority level is `never *`. A preferred configuration file is shown in Example 8-5.

Example 8-5 AIX log management

```
mail.debug      /var/log/mail
user.debug      /var/log/user
kern.debug      /var/log/kern
syslog.debug    /var/log/syslog
daemon.debug    /var/log/daemon
```

```
auth.debug      /var/log/secure
local2.debug    /var/log/sudo
```

It is standard on AIX to not have the logrotate daemon running, so you want to adapt `syslogd` to do the rotation on its own, as shown in Example 8-6.

Example 8-6 Logs management and rotation

```
mail.debug   /var/log/mail  rotate size 100k files 4 # 4 files, 100kB each
user.debug   /var/log/user  rotate files 12 time 1m #12 files, monthly rotate
kern.debug   /var/log/kern  rotate files 12 time 1m compress # 12 files, monthly
rotate, compress
```

When changes are made to `/etc/syslog.conf`, make sure to restart `syslogd`, as shown in Example 8-7.

Example 8-7 Refreshing the syslog daemon

```
[root@sys /] refresh -s syslogd
0513-095 The request for subsystem refresh was completed successfully.
```

For more information about syslog, see IBM Knowledge Center.

It is important that you monitor the automatic AIX reporting, which reports all errors, software failures, and hardware failures. AIX reporting is useful for monitoring Node.js software general errors.

8.3.2 Monitoring resources

Use the **topas**, **mmon**, **svmon**, and **ps** commands as monitoring tools, as shown in Example 8-8. and Example 8-9 on page 169

Example 8-8 The topas monitoring tool

```
> topas
Topas Monitor for host:AIX                          EVENTS/QUEUES    FILE/TTY
Wed Dec  6 11:31:30 2017   Interval:2               Cswitch      90  Readch        0
                                                    Syscall      98  Writech      86
CPU     User% Kern% Wait% Idle%   Physc  Entc%      Reads         0  Rawin         0
Total     0.0   0.3   0.0  99.7    0.01   1.13      Writes        0  Ttyout       86
                                                    Forks         0  Igets         0
Network    BPS  I-Pkts  O-Pkts    B-In   B-Out      Execs         0  Namei         0
Total    7.93K   162.5    0.50   7.82K   114.0      Runqueue      0  Dirblk        0
                                                    Waitqueue   0.0
Disk    Busy%      BPS     TPS  B-Read  B-Writ                      MEMORY
Total     0.0        0       0       0       0     PAGING           Real,MB   16384
                                                    Faults        0  % Comp     15
FileSystem          BPS    TPS  B-Read  B-Writ      Steals        0  % Noncomp  24
Total                 0      0       0       0     PgspIn        0  % Client   24
                                                    PgspOut       0
WLM-Class (Active)     CPU%    Mem%  Blk-I/O%       PageIn        0  PAGING SPACE
System                    0      12         0      PageOut       0  Size,MB    8192
Shared                    0       1         0      Sios          0  % Used      0
                                                                    % Free    100
Name           PID  CPU%  PgSp Class               NFS (calls/sec)
```

```
topas       4915212  0.0 2.15M System          SerV2        0  WPAR Activ    1
wlmsched    1638450  0.0  512K System          CliV2        0  WPAR Total    1
gil         1572912  0.0  960K System          SerV3        0  Press: "h"-help
lockd-1     9634044  0.0 1.19M System          CliV3        0         "q"-quit
clcomd      5570572  0.0 1.70M System          SerV4        0
rpc.lock    4980944  0.0 1.19M System          CliV4        0
```

Example 8-9 This svmon monitoring tool

```
> svmon -Pt5 | perl -e 'while(<>){print if($.==2||$&&&!$s++);$.=0 if(/^-+$/)}'
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
 4718800 rmcd             36265    19639        0    35845      N     Y     N
 5046426 aso              33761    10349        0    29983      Y     N     N
 4587732 IBM.ConfigRMd    32153    14915        0    31857      N     Y     N
 3866758 rpc.statd        32031    14900        0    31996      N     N     N
 5570572 clcomd           32012    14906        0    31734      N     Y     N
```

8.4 Linux on Power setup

This section provides information about how to set up the tools that are needed for log management, performance statistics, and monitoring tools for Linux on Power.

8.4.1 Log management

By default, Linux on Power servers do “syslogging”. The default installation places entries in `/etc/syslog.conf` and `/etc/rsyslog.conf`.

In `syslogd`, `<facility>.*` does not work, so make sure that your priority level is `never *`. A preferred configuration file is shown in Example 8-10.

Example 8-10 Log management on Linux

```
mail.debug      /var/log/mail
user.debug      /var/log/user
kern.debug      /var/log/kern
syslog.debug    /var/log/syslog
daemon.debug    /var/log/daemon
auth.debug      /var/log/secure
local2.debug    /var/log/sudo
```

Adapt `syslogd` to do the rotation on its own, as described in 8.3, “AIX system setup” on page 167.

When adaptations are made to `/etc/syslog.conf`, make sure to restart `syslogd`, as shown in Example 8-11.

Example 8-11 Syslogd on Linux

```
[root@sys /] service syslog start
[root@sys /] systemctl start rsyslog.service
```

For more information about syslog on Linux, see the man pages here and here.

It is important that you monitor the `/var/log/messages` file that reports all errors, software failures, and hardware failures because it is useful for monitoring Node.js software general errors.

8.4.2 Monitoring resources

Use the **`top`**, **`vmstat`**, **`ps`**, and **`lsof`** commands as monitoring tools, as shown in Example 8-12 and Example 8-13 on page 171.

Example 8-12 The topas monitoring tool

```
> top
top - 14:43:26 up 2 days, 12:02,  2 users,  load average: 0.48, 0.55, 0.58
Tasks: 303 total,   2 running, 301 sleeping,   0 stopped,   0 zombie
%Cpu(s):  5.7 us,  2.1 sy,  0.0 ni, 92.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 14111360 total,   843648 free, 11051712 used,  2216000 buff/cache
KiB Swap:  4194240 total,  4194240 free,        0 used.  2069376 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1561 neutron   20   0  299456 106688  20416 S   5.9  0.8  75:37.80
neutron-ibm-hea
 1586 nova      20   0  443840 191552  30592 S   5.9  1.4  80:09.57 nova-scheduler
 1589 glance    20   0  238912 122368  20288 S   5.9  0.9  19:25.36 glance-api
 1593 cinder    20   0  394688 169408  36096 S   5.9  1.2  72:22.70
powervc-cinder-
 1600 cinder    20   0  396352 170880  36096 S   5.9  1.2  85:57.40 cinder-backup
 1612 nova      20   0  384000 189760  24576 S   5.9  1.3  92:02.76
nova-ibm-health
 3110 gnocchi   20   0  730560 133952  19712 S   5.9  0.9   8:13.61
gnocchi-metricd
 4387 nova      20   0  369472 167616  16896 S   5.9  1.2  92:41.80 nova-conductor
    1 root      20   0  163072  14656   7232 S   0.0  0.1   1:33.77 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.32 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.58 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      rt   0       0      0      0 S   0.0  0.0   0:01.62 migration/0
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S   0.0  0.0   1:50.79 rcu_sched
   10 root      rt   0       0      0      0 S   0.0  0.0   0:00.77 watchdog/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:00.74 watchdog/1
   12 root      rt   0       0      0      0 S   0.0  0.0   0:01.61 migration/1
   13 root      20   0       0      0      0 S   0.0  0.0   0:00.62 ksoftirqd/1
   15 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:0H
   16 root      rt   0       0      0      0 S   0.0  0.0   0:00.77 watchdog/2
   17 root      rt   0       0      0      0 S   0.0  0.0   0:01.61 migration/2
   18 root      20   0       0      0      0 S   0.0  0.0   0:00.64 ksoftirqd/2
   20 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/2:0H
   21 root      rt   0       0      0      0 S   0.0  0.0   0:00.75 watchdog/3
   22 root      rt   0       0      0      0 S   0.0  0.0   0:01.60 migration/3
```

Example 8-13 The lsof monitoring tool

```
> lsof
COMMAND     PID   TID           USER   FD      TYPE             DEVICE  SIZE/OFF       NODE NAME
systemd       1                 root  cwd       DIR              253,4      4096         64 /
systemd       1                 root  rtd       DIR              253,4      4096         64 /
systemd       1                 root  txt       REG              253,4   2034824  100704643 /usr/lib
/systemd/systemd
systemd       1                 root  mem       REG              253,4      1006   16872732 /etc/sel
inux/targeted/contexts/files/file_contexts.local.bin
systemd       1                 root  mem       REG              253,4     44725   17046670 /etc/sel
inux/targeted/contexts/files/file_contexts.homedirs.bin
systemd       1                 root  mem       REG              253,4   1402771   17046668 /etc/sel
inux/targeted/contexts/files/file_contexts.bin
systemd       1                 root  mem       REG              253,4     69376   50332174 /usr/lib
```

8.5 Private cloud

A private cloud solution such as the IBM Cloud Private solution can provide a platform for developing and managing on-premises containerized applications, as shown in Figure 8-2. IBM Cloud Private provides a cloud platform that you can install behind your firewall. It provides cloud and application management capabilities.

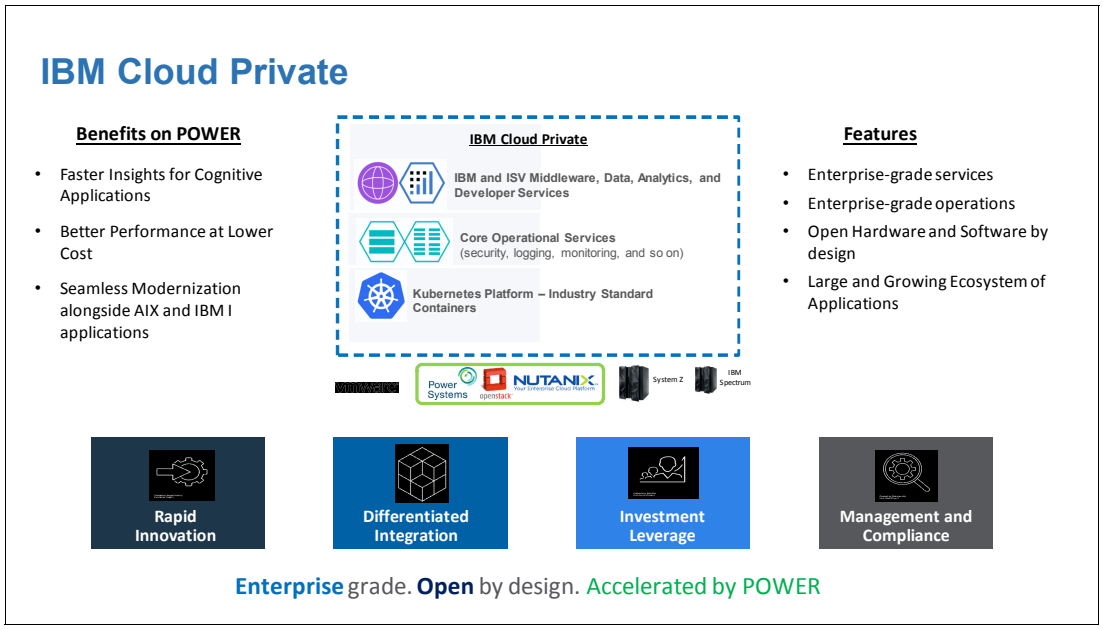

Figure 8-2 IBM Private Cloud

8.6 Deployment

A typical deployment into production involves the following tasks:

- Run **`git`** to fetch the code.
- Run **`npm install`** to fetch dependencies.
- Deploy the application with TLS support.
- Automate the start and stop of the application, including any dependent processes.

This process can be automated end to end, including release test on code, which reduces the potential for user error.

8.6.1 Environments

Your deployment process should be promoted from development, to testing, to staging, and then into production. Testing should be carried at each stage to verify the application in the environment. All testing, verification, and promotions should be automated, except the final promotion into production. This final promotion should be triggered manually.

8.6.2 Dependencies

Run the **`npm module npm-shrinkwrap`** command to lock down the versions of the dependencies that your application uses. This action ensures that the same module versions in the dependency tree are retrieved every time, including dependencies of dependencies. A slight change in a module that is a dependency of a dependency might cause your application to fail. This change in the dependency may occur anytime in your development process.

8.6.3 Reverse proxy server

An HTTP server that acts as a proxy for your applications enables all your applications to appear to be listening on the same 80 / 443 ports. The applications run with non-admin user privileges. The HTTP server is configured with directives to act as a proxy server to the applications, as shown in Figure 8-3.

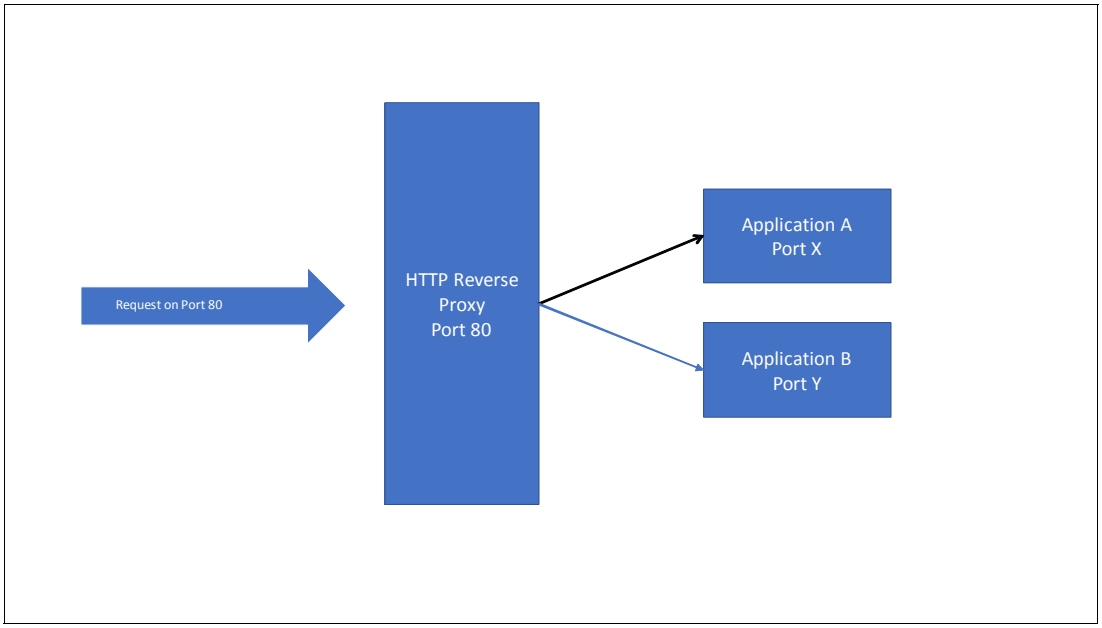

Figure 8-3 Port 80 / 443 Proxy

In most cases, TLS is needed between your application and the proxy server, in which case no changes to your applications are needed. If you must protect the data between your application and the proxy, then you must enable TLS. You must provide certificates for your application and modify your application to create a listener for HTTPS.

8.6.4 Compression

Use the proxy to compress responses. Although there is an NPM compression module for Node.js, the proxy server is better suited to apply compression when the requesting client supports compression.

Compression reduces the size of the data being transmitted over the network, which improves throughput.

8.6.5 Multicore

Depending on your elastic capacity settings, develop your applications to use all the CPU cores that are available to your applications. Node.js has a cluster library with which child processes can be created. The code listing in Example 8-14 reports how many CPU cores are available to the application.

Example 8-14 Counting cores

```
const numCPUs = require('os').cpus().length
console.log('There are %d cores available to this application', numCPUs)
```

Use a proxy server for load-balancing across multiple instances of your application, as shown in Figure 8-4. The proxy server can also monitor health and balance the incoming transactions across the servers. When clustering in this way, the application instances do not share data in memory, so it is not possible to maintain state by using application memory. Instead, use an in-memory data store to hold session data and maintain state.

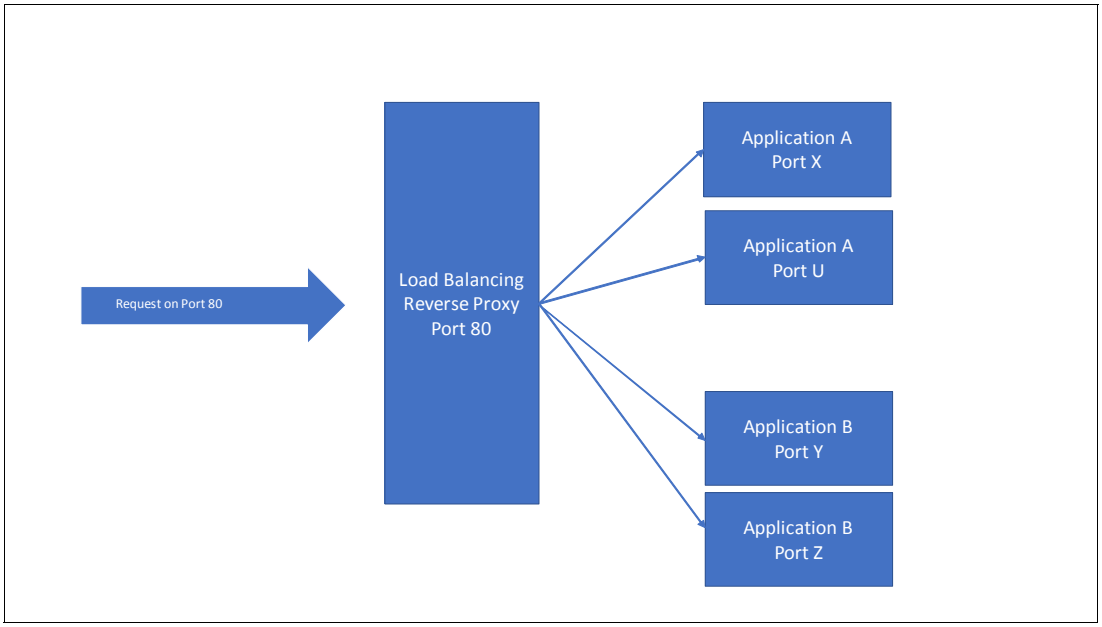

Figure 8-4 Proxy server as load balancer

8.6.6 HTTP caching

Develop your applications with an effective caching policy. The correct policy stops repeated downloads of the same static files. Your web applications have static content, some of which can be large. Set application cache policies to enable caching in the browser and in the HTTP proxy. Do not disable the cache in the HTTP server for static files so that they do not need to be reread for every user request. Applications may also share static files, such as JavaScript libraries. By default, Node.js Express applications enable view-templated caching in production, as shown in Figure 8-5.

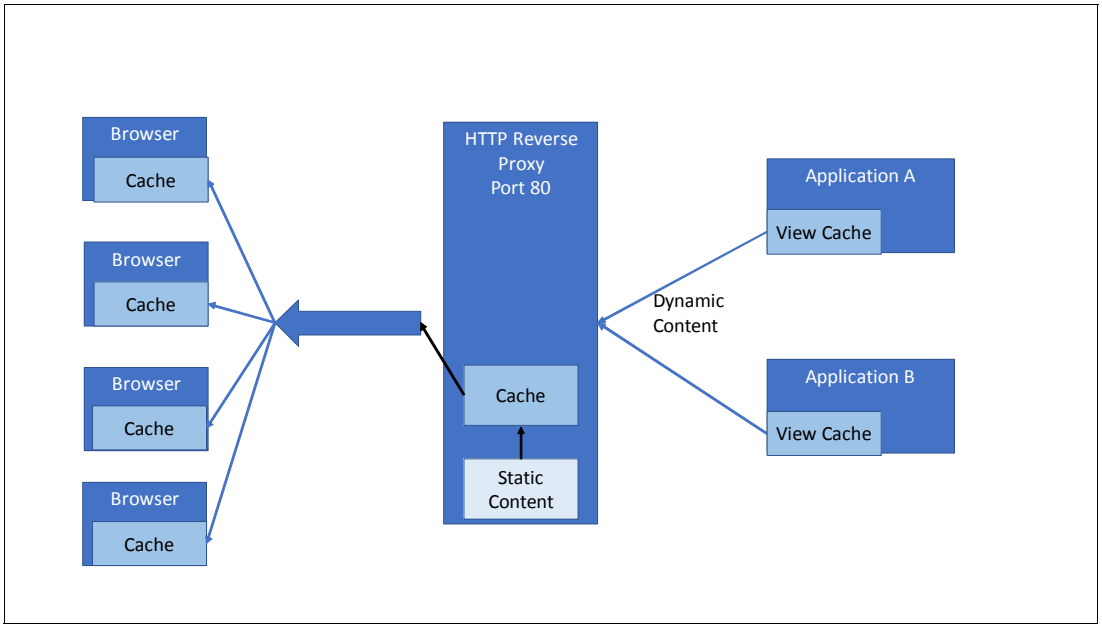

Figure 8-5 Caching policy for performance

8.6.7 Transport Layer Security

If your application uses sensitive data, then you must use TLS, which secures the connection and the data. Configure your reverse proxy server to handle TLS for your applications. If you must protect the data between your application and the proxy, then you must enable TLS between your application and the reverse proxy server, as shown in Figure 8-6.

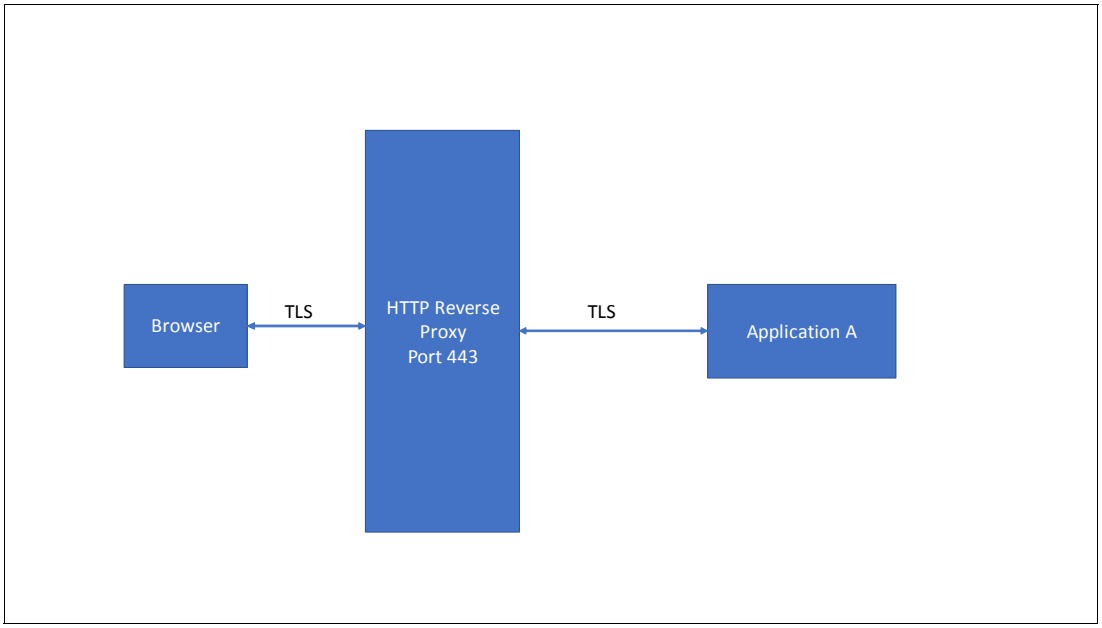

Figure 8-6 End-to-end Transport Layer Security

8.6.8 Proxy server for cloud services

If you use cloud-based services such as the Watson services, then your application needs access to the internet. Do not connect your application directly to the internet. Use a proxy server to handle the connection to any cloud-based services. Watson services by default use SSL for all communication, so you must enable TLS. If you must protect the data between your application and the proxy, then you must enable TLS for both parts of the communication link, as shown in Figure 8-7.

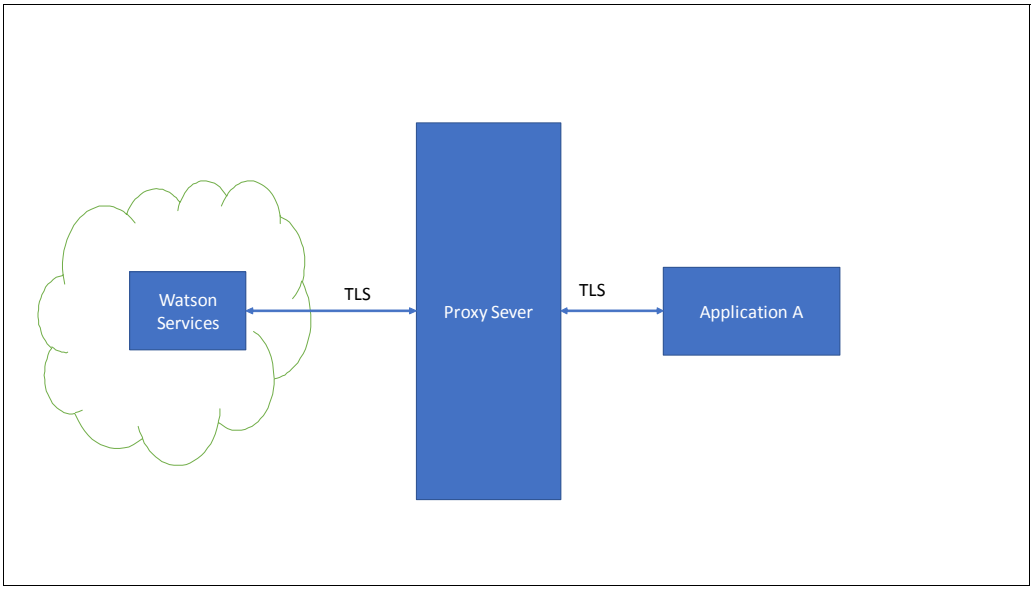

Figure 8-7 Proxy server to access cloud services

Your application communicates with the web services through the proxy server. If you are using the Watson Developer Cloud software developer kit (SDK), then you can configure the endpoint to be your proxy server, as shown in Example 8-15.

Example 8-15 Setting a proxy as an endpoint for Watson Developer Cloud

```
const ConversationV1 = require('watson-developer-cloud/conversation/v1'),

const proxyEndpoint =
'https://watsonconversation.myproxyserver.com/conversation/api';

var serviceSettings = {
  'url' : proxyEndpoint,
  'username' : serviceUserName,
  'password' : servicePassword
};

var watsonService = new ConversationV1(serviceSettings);
```

8.7 Application management on AIX

This section describes application considerations that are related to application build and startup automation on an AIX system.

8.7.1 Application building on AIX

Make sure that the Git client is installed so that the application code and dependencies can be fetched. For more information about installing the Git client, see Chapter 4, “Setting up the development environment” on page 71.

8.7.2 Automated application start on AIX

Automate the application start and restart after a crash of by adding an entry to the `/etc/inittab` file with a `respawn` action, as shown in Example 8-16.

Example 8-16 Automated application start on AIX

```
> mkitab “npm_id:2:respawn:npm_full_command”
```

8.8 AIX Application high availability

The application can be made highly available by setting up IBM AIX PowerHA and defining this application as an *application controller* inside a PowerHA resource group and the AIX file systems. For more information about PowerHA, see IBM Knowledge Center.

8.9 Setting up an HTTP proxy on AIX

The HTTP proxy server can be provided by either IBM HTTP Server (IHS) or Web Application Server.

In IHS, you must edit the configuration files that are based on the Apache services. However, in IBM WebSphere®, you modify or add server types and proxy servers from the GUI by clicking **Servers** → **Servers Types** → **WebSphere proxy servers**.

A proxy action cannot be performed unless it is associated with a proxy rule expression. A proxy rule expression is evaluated only when it is associated with a proxy virtual host. A proxy action can be created or administered without a proxy rule expression or a proxy virtual host, but it cannot be used without them.

Creating a caching proxy can be done by completing the following steps:

1. Enter the name of the proxy action in the Action name field. The Action name is used as the unique identifier for this proxy action configuration. The Action name must be unique within the cell and cannot include any of the following characters: # \ / , : ; " * ? < > | = + & or %.
2. Select the **Enable caching** check box to enable caching.

3. Enter the value in seconds for the Default expiration field. The Default expiration field specifies the amount of time, in seconds, before a cached response expires.
4. Enter the value in seconds for the Last modified factor. The Last modified factor field specifies the amount of time before a response is cached if the response does not have explicit HTTP expiration headers.

Configuring the HTTP server as a load balancer is a preferred practice. In a typical configuration that uses IHS for load-balancing, IHS is installed on a separate server. The full procedures are shown in IBM Knowledge Center.

The HTTP server maintains a file where its configuration is stored. The configuration file for HTTP server has the following default path:

```
/opt/IBM/HTTPServer/conf/httpd.conf
```

Any SSL configurations must go into this file. Example 8-17 for shows how the SSL definition supports the TLS protocol, where TLS is the successor of SSL.

Example 8-17 SSL / Transport Layer Security configuration

```
# Example SSL(TLS) configuration
#
# required due to GSKit8 library problem
LoadFile /usr/lib64/libcrypto.so
#
LoadModule ibm_ssl_module modules/mod_ibm_ssl.so
#
Listen 443
<VirtualHost wasnode1.net:443>
ServerName wasnode1.net
#
SSLEnable
SSLProtocolDisable SSLv2
SSLProtocolDisable SSLv3
# cipher suite TLS_RSA_WITH_AES_256_CBC_SHA(35b)
# remove all ciphers first
SSLCipherSpec ALL NONE
SSLCipherSpec ALL +TLS_RSA_WITH_AES_256_CBC_SHA
# symmetric offload (older GSKit versions)
# SSLAttributeSet 417 549
</VirtualHost>
# PKCS#11 configuration
KeyFile /opt/HTTPServer/ssl/key.kdb
SSLServerCert IBMICATOK:ihscert
SSLStashfile /opt/IBM/HTTPServer/ssl/ibmicatok.sth
SSLPKCSDriver /usr/lib/pkcs11/PKCS11_API.so64
SSLDisable
SSLCachePortFilename /opt/IBM/HTTPServer/logs/siddport
# End of SSL configuration
```

8.10 Multi-cores in AIX

In AIX, make sure that the CPU multi-cores feature is enabled, as shown in Example 8-18.

Note: If AIX has two assigned processors, either virtual or dedicated, the multi-cores features can be enabled so that each processor is represented as four logical CPUs.

Example 8-18 AIX CPU multi-cores feature

```
# lsdev -Cc processor
proc0 Available 00-00 Processor
proc8 Available 00-08 Processor

# smtctl -m on -w now
smtctl: SMT is now enabled.

# smtctl
This system is SMT capable.
This system supports up to 8 SMT threads per processor.
SMT is currently enabled.
SMT boot mode is not set.
SMT threads are bound to the same virtual processor.

proc0 has 4 SMT threads.
Bind processor 0 is bound with proc0
Bind processor 1 is bound with proc0
Bind processor 2 is bound with proc0
Bind processor 3 is bound with proc0

proc8 has 4 SMT threads.
Bind processor 4 is bound with proc8
Bind processor 5 is bound with proc8
Bind processor 6 is bound with proc8
Bind processor 7 is bound with proc8

# sar -u 1 1
AIX AIX 1 7 00FA44234C00    12/06/17
System configuration: lcpu=8 ent=1.00 mode=Uncapped
15:59:29    %usr    %sys    %wio   %idle   physc   %entc
15:59:30       0       0       0     100    0.01     1.1
```

8.11 Application management on Linux

This section describes application considerations that are related to application build and startup automation on an Linux on Power system.

8.11.1 Application building on Linux

Make sure that the Git client is installed so that application code and dependencies can be fetched. For more information about the environment setup, see Chapter 4, “Setting up the development environment” on page 71.

8.11.2 Automated application start on Linux

Automate the application start and restart after a crash by adding an entry to `/etc/inittab file` with a `respawn` action, or add the action to `/etc/rc.local` as a command, as shown in Example 8-19.

Example 8-19 Automated application start on Linux

```
> vi /etc/inittab
“npm_id:2:respawn:npm_full_command”
vi /etc/rc.local
```

8.12 Linux application high availability

The application can be made highly available by setting up Linux Global Filesystem and defining this application as a service inside the cluster and the global file systems. For more information about clustering, see Red Hat Global File System.

8.12.1 Setting up the HTTP proxy serve on Linux

The HTTP proxy server can be provided by either IHS (based on Apache) or Web Application Server.

In IHS, you must edit the configuration files that are based on the Apache services. However, in WebSphere, you modify or add server types and proxy servers from the GUI by selecting **Servers** → **Servers Types** → **WebSphere proxy servers**.

A proxy action cannot be performed unless it is associated with a proxy rule expression. A proxy rule expression is evaluated only when it is associated with a proxy virtual host. A proxy action can be created or administered without a proxy rule expression or a proxy virtual host, but it cannot be used without them.

Creating a caching proxy can be done by completing the following steps:

1. Enter the name of the proxy action in the Action name field. The Action name is used as the unique identifier for this proxy action configuration. The Action name must be unique within the cell and cannot include any of the following characters: # \ / , : ; " * ? < > | = + & or %.
2. Select the **Enable caching** check box to enable caching.
3. Enter the value in seconds for the Default expiration field. The Default expiration field specifies the amount of time, in seconds, before a cached response expires.
4. Enter the value in seconds for the Last modified factor. The Last modified factor field specifies the amount of time before a response is cached if the response does not have explicit HTTP expiration headers.

Configuring HTTP server as a load balancer is a preferred practice. In a typical configuration that uses IHS for load-balancing, IHS is installed on a separate server. The full procedures are shown in IBM Knowledge Center.

The HTTP server maintains a file where its configuration is stored. The configuration file for the HTTP server has the following default path:

`/opt/IBM/HTTPServer/conf/httpd.conf`

Any SSL configurations must go into this file. Example 8-20 for shows how the SSL definition supports the TLS protocol, where TLS is the successor for the SSL.

Example 8-20 SSL / Transport Layer Security configuration

```
# Example SSL(TLS) configuration
# required due to GSKit8 library problem
LoadFile /usr/lib64/libcrypto.so
LoadModule ibm_ssl_module modules/mod_ibm_ssl.so
Listen 443
<VirtualHost wasnode1.net:443>
ServerName wasnode1.net
SSLEnable
SSLProtocolDisable SSLv2
SSLProtocolDisable SSLv3
# cipher suite TLS_RSA_WITH_AES_256_CBC_SHA(35b)
# remove all ciphers first
SSLCipherSpec ALL NONE
SSLCipherSpec ALL +TLS_RSA_WITH_AES_256_CBC_SHA
# symmetric offload (older GSKit versions)
# SSLAttributeSet 417 549
</VirtualHost>
# PKCS#11 configuration
KeyFile /opt/HTTPServer/ssl/key.kdb
SSLServerCert IBMICATOK:ihscert
SSLStashfile /opt/IBM/HTTPServer/ssl/ibmicatok.sth
SSLPKCSDriver /usr/lib/pkcs11/PKCS11_API.so64
SSLDisable
SSLCachePortFilename /opt/IBM/HTTPServer/logs/siddport
# End of SSL configuration
```

8.13 Application updates

Application updates happen because of application changes, improvements, or fixes. If you adopt a continuous delivery approach, then there will be a regular stream of updates. Create a mechanism that enables your developers to submit updates to a code respiratory. This mechanism should automate the process of code review and test. Approved updates make up a release.

Adopt a Continuous Integration approach where incremental updates to code are made frequently so that you can test changes and fixes in smaller increments. To enable Continuous Integration, you must automate the build and test processes so that you can immediately evaluate the quality of code changes.

Use a tool to automate the process of building, testing, and deployment of your application when the build and test phases are successful. If possible, run tests in parallel.

There are several tools to manage open source developments:

- Travis CI offers a website for closed projects, but remains a hosted web service. Your corporate policy might not allow the usage of a hosted web service to manage your builds.
- Jenkins runs in a Docker container that you can run on your Power Systems server.
- There are Node.js alternatives that run from the command line that you can automate. For the current preferred practices for continuous integration build and test tools, see the Node.js website.

You should set up an application update mechanism that enables you to deploy releases with a minimum application downtime.

A

Setup and configuration of the IBM DB2 server

This appendix describes the IBM DB2 server installation and configuration that is required for the application environment that is described in this book.

The appendix includes the following topic:

- “Preparing the DB2 server”

© Copyright IBM Corp. 2018. All rights reserved.

Preparing the DB2 server

In this section, you prepare the DB2 server by creating a database-owning user and group by completing the following steps:

1. Create the user and group, and input a password for the user, as shown in Example A-1.

Example A-1 Creating a user and group

```
# mkgroup db2admg
# mkuser -a pgrp=db2admg nodedb2
# echo "nodedb2:nodedb2" | chpasswd
```

2. Make sure that the DB2 source is owned by the user `nodedb2` and the group `db2admg`, as shown in Example A-2.

Example A-2 Changing the ownership to the DB2 owning user

```
# chown -R nodedb2.db2admg /DB2_SOURCE
```

3. Log in with the `nodedeb2` user and perform the installation by running **`db2_install`**, as shown in Example A-3.

Example A-3 Running the DB2 installer

```
$ ./db2_install
DBI1324W  Support of the db2_install command is deprecated.
DBI1244I  Directory for non-root installation of DB2 - /home/nodedb2/sqllib
Specify one of the following keywords to install DB2 products.
  SERVER
  CONSV
  CLIENT
  RTCL
Enter "help" to redisplay product names.
Enter "quit" to exit.
************************************************************
SERVER
DBI1160I  Non-root install is being performed.
DB2 installation is being initialized.
Total number of tasks to be performed:43
Total estimated time for all tasks to be performed: 3770 second(s)

Task #1 start
Description: Checking license agreement acceptance
Estimated time 1 second(s)
Task #1 end
[..]
Task #45 start
Description: Updating global profile registry
Estimated time 3 second(s)
Task #45 end
The execution completed successfully.
```

4. Log in again with the `nodedb2` user to reread the new DB2 profile.

5. Make sure that the instance is created by running the command that is shown in Example A-4.

Example A-4 Listing the current DB2 instance

```
$ db2ilist
nodedb2
```

6. Update the communication port in DB2 and the UNIX server, as shown in Example A-5.

Example A-5 Updating the DB2 communication port

```
$ db2 update dbm cfg using SVCENAME 51500
DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command completed
successfully.

$ db2 get dbm cfg | grep SVCE
TCP/IP Service name                          (SVCENAME) = 51500
SSL service name                         (SSL_SVCENAME) =

$ vi /etc/services
Added the lines:
db2c_nodedb2            51500/tcp       # DB2 connections for nodedb2
db2c_nodedb2            51500/udp
```

7. Start DB2 by running **db2start**, as shown in Example A-6.

Example A-6 Starting DB2

```
$ db2start
12/14/2017 21:47:03     0   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.
```

B

Additional material

This book refers to additional material that can be downloaded from the internet, as described in the following section.

Locating the material on GitHub

The source code that is associated with Chapter 6, “Use case implementation for IBM AIX and Linux on Power” on page 107 is available at the following GitHub repository:

https://github.com/ibm-redbooks-dev/customer-feedback-power

The source code that is associated with Chapter 7, “Use case implementation for IBM i” on page 131 is available at the following GitHub repository:

https://github.com/ibm-redbooks-dev/customer-feedback-power

© Copyright IBM Corp. 2018. All rights reserved.

Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide more information about the topics in this document. Some publications that are referenced in this list might be available in softcopy only.

- *Building Cognitive Applications with IBM Watson Services: Volume 1 Getting Started*, SG24-8387
- *Building Cognitive Applications with IBM Watson Services: Volume 2 Conversation*, SG24-8394
- *Building Cognitive Applications with IBM Watson Services: Volume 3 Visual Recognition*, SG24-8393
- *Building Cognitive Applications with IBM Watson Services: Volume 4 Natural Language Classifier*, SG24-8391
- *Building Cognitive Applications with IBM Watson Services: Volume 5 Language Translator*, SG24-8392
- *Building Cognitive Applications with IBM Watson Services: Volume 6 Speech to Text and Text to Speech*, SG24-8388
- *Building Cognitive Applications with IBM Watson Services: Volume 7 Natural Language Understanding*, SG24-8398
- *Developing Node.js Applications on IBM Cloud*, SG24-8406
- *Essentials of Application Development on IBM Cloud*, SG24-8374

You can search for, view, download, or order these documents and other IBM Redbooks publications, Redpapers, web docs, drafts, and extra materials, at the following website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

© Copyright IBM Corp. 2018. All rights reserved.

Redbooks

Enhancing the IBM Power Systems Platform with IBM Watson Services

(0.2”spine)
0.17”<->0.473”
90<->249 pages

SG24-8419-00

ISBN 0738443026

Printed in U.S.A.

Get connected