

Automated Application Delivery with OpenStack Heat Patterns

Claudio Tagliabue

Esteban Arias

Ashok Iyengar

John L Hatfield



 **Cloud**

PureSystems



International Technical Support Organization

**Automated Application Delivery with OpenStack Heat
Patterns**

November 2016

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

First Edition (November 2016)

This edition applies to IBM PureApplication Software V2.2.2.

This document was created or updated on November 17, 2016.

© Copyright International Business Machines Corporation 2016. 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

Notices	v
Trademarks	vi
Preface	vii
Authors	vii
Now you can become a published author, too	ix
Comments welcome	ix
Stay connected to IBM Redbooks	ix
Chapter 1. Automated application delivery with OpenStack Heat patterns	1
1.1 Patterns Overview	2
1.2 Technologies for pattern creation	6
1.2.1 OpenStack	6
1.2.2 IBM Blue Box	9
1.2.3 Docker	10
1.2.4 SaltStack	12
1.2.5 OpenStack Heat	13
1.2.6 Zeron	15
1.3 IBM PureApplication System products for OpenStack	16
1.4 OpenStack Heat patterns	18
1.4.1 OpenStack Heat patterns on IBM Bluemix Local System with PureApplication Software	21
1.4.2 OpenStack Heat patterns on IBM Bluemix	22
1.5 Conclusions	23
Chapter 2. Setting up IBM Bluemix Local System for OpenStack Heat patterns	25
2.1 Setting up OpenStack on Bluemix Local System	26
2.1.1 Preparing Bluemix Local System	26
2.1.2 Enabling OpenStack services	28
2.2 Configuring Blue Box	30
2.2.1 Verifying OpenStack services	31
2.3 Sample OpenStack Heat patterns	33
2.4 Using an alternative Git repository	33
2.4.1 Stand up a new Git repository on Bluemix Local System	34
Chapter 3. Deploying sample OpenStack Heat patterns	35
3.1 Verifying the installation of the sample patterns	36
3.2 Installing OpenStack Heat pattern samples	36
3.2.1 Deploying the WebSphere Application Server OpenStack Heat pattern	37
3.2.2 Deploying the DB2 Server OpenStack Heat pattern	40
3.3 Managing the OpenStack Heat pattern deployments	42
Chapter 4. Importing OpenStack Heat patterns	47
4.1 Importing a simple Heat Orchestration Template	48
4.2 Importing from a Git repository	50
Chapter 5. Creating an OpenStack Heat pattern	53
5.1 Creating a blueprint in UCD Blueprint Designer	54
5.2 Defining a new OpenStack Heat pattern	57

5.3	Adding SaltStack formulas to the OpenStack Heat pattern	58
5.4	Committing and pushing the OpenStack Heat pattern to Git	60
5.5	Deploying the OpenStack Heat pattern	62
Chapter 6. Creating automation for OpenStack Heat patterns with SaltStack		65
6.1	SaltStack concepts	66
6.1.1	Master and minion communication	66
6.2	General SaltStack examples	67
6.2.1	Remote execution	68
6.2.2	Package installation	68
6.2.3	SaltStack states	69
6.2.4	Formula files	69
6.2.5	Git repositories and SaltStack files	70
6.3	Using SaltStack within OpenStack Heat patterns	70
6.3.1	Deploying an OpenStack Heat pattern	72
6.3.2	Starting Heat orchestration	72
6.3.3	Deploying virtual machines and install minions	72
6.3.4	Retrieving information and files from Git repositories	72
6.3.5	Heat and SaltStack operations	73
6.3.6	Running orchestration and running a formula	73
6.3.7	OpenStack Heat pattern deployed	74
Chapter 7. Zeron for OpenStack Heat patterns		75
7.1	Zeron architecture	76
7.1.1	Zeron terminology	76
7.1.2	Security in Zeron	77
7.1.3	Inside Zeron	77
7.2	Zeron user interface	78
7.2.1	Zeron UI for OpenStack Heat patterns on PureApplication Software	78
7.3	Zeron handlers	82
7.3.1	Inventory handler type	82
7.3.2	Logging handler type	82
7.3.3	Monitoring handler type	83
7.3.4	Operation handler type	83
7.4	Developer Roles in Zeron	85
Related publications		87
IBM Redbooks		87
Online resources		87
Help from IBM		88

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.

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.

Blue Box®	IBM®	Redpaper™
Bluemix®	IBM Blue™	Redbooks (logo)  ®
Box Panel®	IBM UrbanCode™	UrbanCode™
DB2®	PureApplication®	WebSphere®
developerWorks®	Redbooks®	

The following terms are trademarks of other companies:

SoftLayer, and SoftLayer device are trademarks or registered trademarks of SoftLayer, Inc., 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.

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

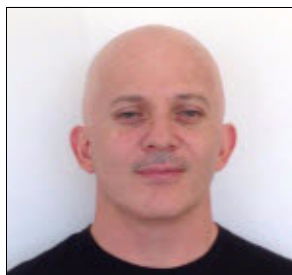
This IBM® Redpaper™ publication focuses on demonstrating how to build, deploy, and manage OpenStack Heat Patterns on IBM PureApplication® Systems. This paper addresses the topic of automated application deployment and delivery through patterns. In particular, it focuses on patterns based on open source technologies, primarily OpenStack.

Authors

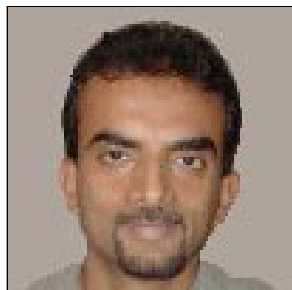
This paper was produced by a team of specialists from around the world, working at the International Technical Support Organization (ITSO), Raleigh Center.



Claudio Tagliabue is a certified IBM technical specialist, focusing on software as a service (SaaS) and business transformation (BT). He is excited by business change in the 21st Century and the consequent demands on technology. Transforming the “complicated” into “business as usual” has been Tag’s focus for the past 5 years. Through his work with clients as a domain expert for IBM’s ground-breaking PureApplication System, platform as a service (PaaS), software as a service, and business process management (BPM) technologies, Tag has gained significant insight into today’s senior IT stakeholder motivations. Tag presented his work at several conferences around Europe. Tag’s experience goes outside the IBM Software world, with work on innovative projects that involve open source technologies, such as OpenStack and Docker.



Esteban Arias is a Software engineer with over 18 years of experience, currently working for Data Analytics and Insights as a service, and formerly for Advanced Cloud solutions. All of this work was within IBM, mostly focused on leveraging current technology to build environments for clients. He is a subject matter expert (SME) in areas that include networking, storage, servers, and integrating many products, such as VMware, SoftLayer®, virtual storage area network (VSAN), VMware NSX, IBM Cloud Orchestrator (IBM ICO), OpenStack in general, and big data tools, using hybrid cloud models or private clouds.



Ashok Iyengar is an executive information technology (IT) specialist at IBM based in San Diego, and has worked in the IT industry for more than 30 years. He holds an MS degree in Computer Science from North Dakota State University, Fargo. In his spare time, Ashok loves to write. Among his works are the popular IBM WebSphere® Portal Primer and WebSphere Business Integration Primer. For the past several years, Ashok has worked on cloud-based projects, completing proofs of concept (POCs), pilots, architecture design, and implementations.



John L Hatfield is a Managing Hybrid Cloud Consultant in the United States. He has four years of experience in the cloud computing field. He has worked at IBM for 38 years. His areas of expertise include PureApplication System, IBM MQ, WebSphere Message Broker, WebSphere Transformation Extender, electronic data interchange (EDI), and data mapping.

This project was led by Margaret Ticknor, an IBM Technical Content Services Project Leader in the Raleigh Center. She primarily leads projects about WebSphere products and IBM PureApplication System. Before joining IBM Technical Content Services, Margaret worked as an IT specialist in Endicott, NY. Margaret attended the Computer Science program at State University of New York at Binghamton.

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

Joe Wigglesworth

STSM, PureApplication Software Development, IBM Cloud

Majeed Arni

Senior Software Engineer, IBM PureApplication System development, IBM Cloud

Kenneth Chu

Software Developer, IBM Cloud

Radu Mateescu

Senior Software Engineer, IBM Cloud

Eugen Postea

Advisory Software Developer, IBM Cloud

Seth Peterson

Advisory Software Engineer, IBM Cloud

Thanks to the following people for their support of this project:

- ▶ LindaMay Patterson, IBM Redbooks® Technical Writer
- ▶ Ann Lund, IBM Redbooks Residency Administrator

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 two - six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Learn more about the residency program, browse the residency index, and apply online:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

- ▶ Use the online **Contact us** review Redbooks form:

ibm.com/redbooks

- ▶ Send your comments in an email:

redbooks@us.ibm.com

- ▶ Mail your comments:

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>



Automated application delivery with OpenStack Heat patterns

This paper addresses the topic of automated application deployment and delivery through *patterns*. In particular, it focuses on patterns based on open source technologies, and primarily OpenStack.

In this context, a pattern completely defines the architecture of an application, and the underlying infrastructure topology that is needed for the application to meet its non-functional requirements. Typically, an enterprise application requires a set of core components to run, such as an application server, a database server, and a web server.

A pattern is in essence a *deployable artifact* that is expected to take care of some key high-level aspects of your development lifecycle:

- ▶ Install the operating system (OS) for each component of the application
- ▶ Integrate the application components
- ▶ Configure and tune the application
- ▶ Secure the application
- ▶ Monitor the application
- ▶ Manage the lifecycle of the application

Patterns bring together various pieces, such as operating system configurations that meet organizational security checklists, integrated software components, and any needed agents or configuration scripts. With knowledge of the roles that each component plays within the pattern, monitoring and lifecycle management can be used to keep the deployment running effectively.

Taking the manual work out of deployments speeds them up and makes it possible to quickly create new deployments. Additionally, patterns can be extended and improved to capture efficiencies, to lower costs and reduce the required resources. The repeatable nature of *patternized* deployments reduces errors due to manual steps, and lowers the risk of failed deployments.

1.1 Patterns overview

Patterns provide the capabilities that were described previously, along with pre-configured, pre-integrated, and pre-tested tasks in a deployable form, enabling repeatable deployments with full lifecycle management. Figure 1-1 defines the building blocks of a pattern.

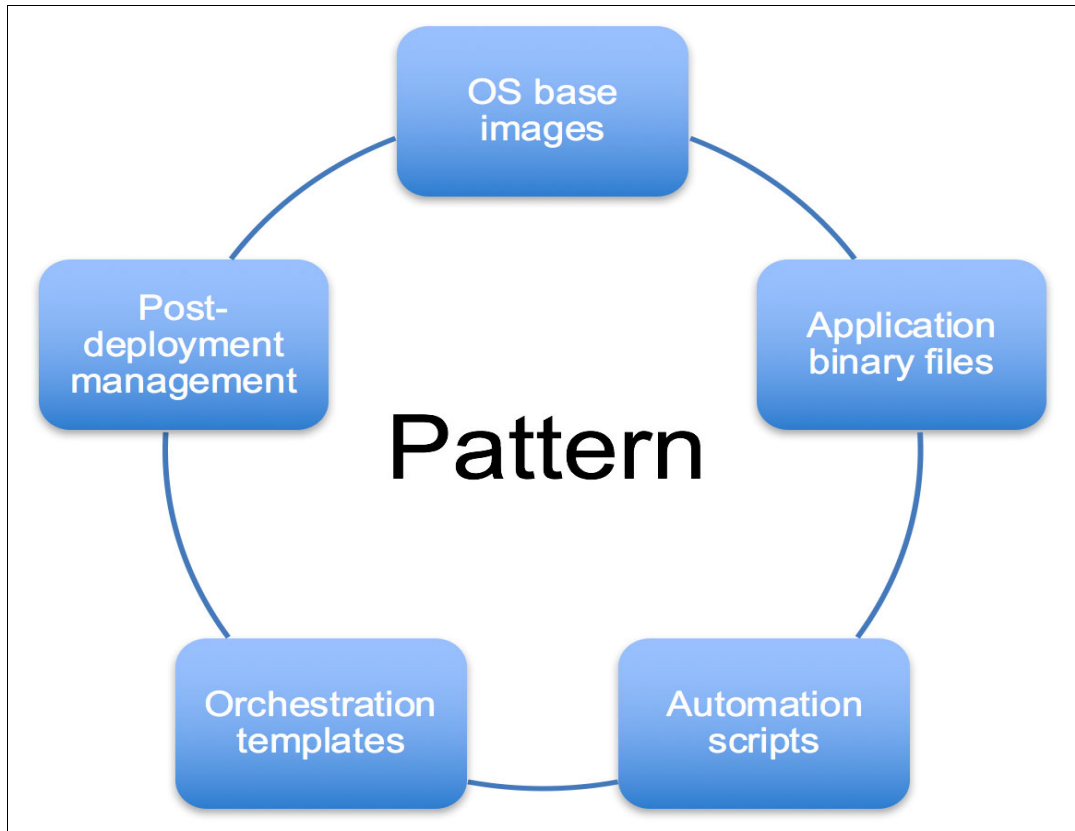


Figure 1-1 Building blocks of a pattern

Historically, the *Patterns of Expertise* that were introduced by IBM were the first enterprise implementation of this concept, and are today the primary deployment choice for complex IBM middleware. Patterns of Expertise can be deployed using products that are part of the IBM PureApplication System family of product offerings (described in 1.3, “IBM PureApplication System products for OpenStack” on page 16), which are the orchestrators and target for the deployment of this type of patterns.

A detailed description of the IBM Patterns of Expertise can be found in the IBM developerWorks® article *Discover PureApplication System patterns of expertise* at the following web address:

<http://ibm.biz/PatternsOfExpertise>

However, patterns based on open technologies could provide some key advantages over Patterns of Expertise:

- ▶ Higher levels of portability, because open technologies are adopted widely and are available for use on many systems.
- ▶ Higher speed of innovation, because open technologies are supported by a vast community.

- ▶ Lower cost of entry, because open technologies are typically free to use.
- ▶ Easier integration between open technologies, because they are often developed to work together.
- ▶ Consistency to the user experience for open technologies.

Since the introduction of IBM Patterns of Expertise, several technologies, primarily open source, have become popular, tackling one or more of the aspects that a pattern should address. The most popular of those technologies can be grouped in the seven layers that are depicted in Figure 1-2.

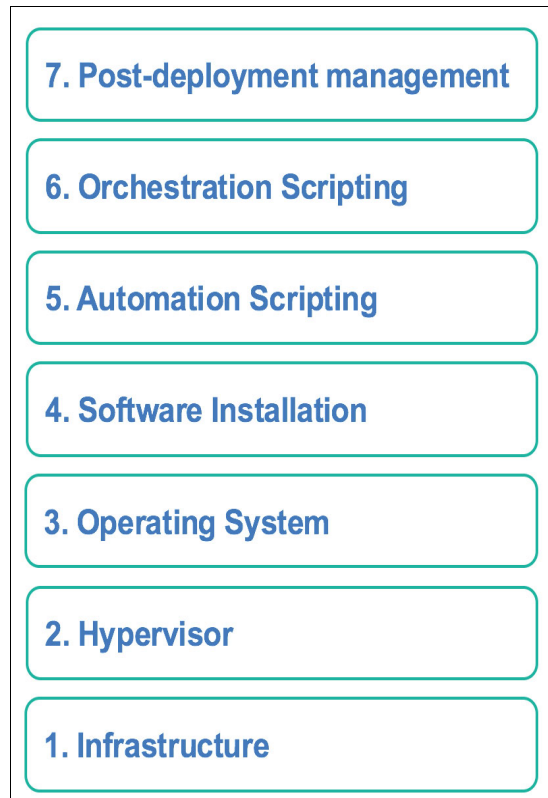


Figure 1-2 Seven layers of capabilities that deliver key aspects of a pattern

Figure 1-2 defines seven layers of technologies that can be used for pattern creation, deployment, and management, based on the role that they play within the pattern components and orchestration. The numbers in the following list correspond to the ones in Figure 1-2. The seven layers are described from the bottom up:

- ▶ Infrastructure (layer 1)

Although typically a physical component, software-defined infrastructure and infrastructure as a service (IaaS) are today the main building blocks for private and public clouds. The most popular open source technology for software-defined, server-based computing is *OpenStack*.¹ OpenStack provides a set of software tools for building and managing cloud computing platforms.

¹ OpenStack <https://www.openstack.org/>

Note: As opposed to server-based IaaS, so called server-less computing provides a new paradigm to use infrastructure and platform as a service (IaaS and PaaS). The following players are the main players in this space:

- ▶ OpenWhisk:
<https://developer.ibm.com/openwhisk/>
- ▶ Amazon Web Services (AWS) Lambda:
<https://aws.amazon.com/lambda/>

Server-less computing is not covered in this paper.

▶ Hypervisor (layer 2)

A layer that manages virtual machines, essentially enabling multiple operating systems to share infrastructure hosts:

<https://en.wikipedia.org/wiki/Hypervisor>

The following examples of hypervisors are well-known:

- VMware ESXi:
<http://www.vmware.com/>
- KVM:
<http://ibm.biz/Kernel-basedVM>

▶ Operating System (layer 3)

Open source operating systems have become significantly popular with the extensive adoption of Linux. Linux is available across a wide variety of processor architectures, and there are several popular open source distributions to choose from:

- Red Hat Linux is widely adopted in many enterprises:
<https://www.redhat.com>
- openSUSE Linux is based on the SUSE Linux professional distribution:
<https://www.opensuse.org>
- Ubuntu Linux has become the reference OS for both OpenStack and Docker development:
<http://www.ubuntu.com>

▶ Software installation (layer 4)

Delivering software applications has been drastically changed by the increasing use of container-based solutions. Conventional methods of delivering software in binary files or standard software packages, requiring an installation program, are still popular and widely adopted. However, delivering software in containers is simpler and faster. Additionally, the results of container-based software delivery are usually more reliable, because of the isolation enforced by container boundaries:

- Docker is the dominant container technology in the market:
<https://www.docker.com>
- Rkt (pronounced “rocket”) is a container technology that uses the Go programming language:
<https://coreos.com/rkt/>
<https://golang.org/>

However, Docker and Rkt are not the only players in this space. In order to prevent the container world from fracturing into incompatible technologies, many organizations joined forces to create the *Open Container Initiative*, and to establish standards regarding container formats and runtimes:

<https://www.opencontainers.org>

► Automation scripting (layer 5)

Operating systems and applications usually need configuration before they become what an organization needs. Traditionally, this automation is accomplished with batch files and shell scripts, which are essentially simple lists of operator commands. Today, open source technologies exist that raise the sophistication of scripting to become true *desired-state* management systems.

The desired state of a system can be described in code, and if the system drifts away from that state, commands run to bring it back to the desired state. Popular open source technologies in this space include the following platforms:

- SaltStack:

<https://saltstack.com>

- Chef:

<https://www.chef.io>

- Puppet:

<https://puppet.com>

- Ansible:

<https://www.ansible.com>

► Orchestration scripting (layer 6)

An extra layer to orchestrate end-to-end the discrete components that are described in the previous layers is usually required. This role is performed by orchestration scripts, or *templates*, and the orchestration engines that interpret them. Orchestration scripts and engines essentially replace instruction books and hours of manual labor, which would go into connecting together all of the systems that make up the deployment of a complex software application:

- One of the primary players in this space is OpenStack Heat, which provides both a language to create orchestration templates and an orchestration engine to deploy them onto the OpenStack infrastructure:

<https://wiki.openstack.org/wiki/Heat>

- When the only software delivery technology chosen by an enterprise is Docker, Docker Compose can be used as the orchestration scripting technology. Conversely, OpenStack Heat is able to work across different types of containers, virtual machines, and bare metal deployments:

<https://docs.docker.com/compose>

► Post-deployment management (layer 7)

After a pattern is orchestrated and deployed, several actions can be performed to automatically manage operations and monitoring of the pattern and its lifecycle. Examples of these actions are monitoring, log collection, and application lifecycle management. In essence, post-deployment management provides the *operations* part of pattern DevOps.²

² For more information about the DevOps (development and operations) movement, watch the video at <http://ibm.biz/Dev-Ops> and also so <http://ibm.co/devopsfordummies>

Although automation scripts, orchestration scripts, and endpoint management systems can be stretched to provide this capability at single virtual machine (VM) or container level, not many mature end-to-end open source capabilities for post-deployment management are available at the time this paper is written. For this reason, IBM created a framework to manage post-deployment actions through the automation scripting technologies that were described previously. This framework is called *Zeron*.

The remainder of this chapter describes the following topics:

- ▶ 1.2, “Technologies for pattern creation” provides information about some of the technology choices behind the seven layers described in Figure 1-2 on page 3.
- ▶ In 1.3, “IBM PureApplication System products for OpenStack”, we describe the PureApplication family, and how IBM Bluemix® Local System implements OpenStack.
- ▶ Finally, in 1.4, “OpenStack Heat patterns”, an alternative to IBM Patterns of Expertise is introduced, based on open technologies: *OpenStack Heat patterns*.

1.2 Technologies for pattern creation

This section introduces in more detail some of the technologies that were mentioned so far, in particular:

- ▶ 1.2.1, “OpenStack” and its IBM distribution: IBM Blue™ Box®
- ▶ 1.2.3, “Docker” on page 10
- ▶ 1.2.4, “SaltStack” on page 12
- ▶ 1.2.5, “OpenStack Heat” on page 13
- ▶ 1.2.6, “Zeron” on page 15

1.2.1 OpenStack

OpenStack is an open source software platform for building and managing cloud computing platforms for public and private clouds, with the main goal of providing users with a system that scales massively, runs in commodity hardware, and is loosely coupled across its components, provided by services and can be deployed with simplicity. All of this functionality is backed up by one of the largest open source communities, with over 50,000 individuals that work actively in different areas of the platform lifecycle, including development, testing, deployment, and integration.

OpenStack controls large pools of compute resources, storage endpoints, and networking capable devices, all distributed in the data center, presented in a pay-as-you-order fashion that enables the user to select what they need from the pool, use it, and release.

The OpenStack release cycle is approximately six months. A new version is published after a series of upgrades and improvements, which might include new capabilities or integrate current ones. At the time of the writing of this paper, the code name for the current stable release is *Mitaka*.

OpenStack is in essence an infrastructure as a service (IaaS) solution, based on a model of decentralized components and services. This model *enables* all of OpenStack’s components to provide a single unit that can be deployed as part of a *stack* installation.

Each project within the community is developed and controlled individually, providing a functionality that complements and integrates with the rest of the OpenStack projects. OpenStack projects can also be identified by the service that they provide. The main projects are listed and described in Table 1-1.

Table 1-1 OpenStack projects and services

Name	Service	Use	Description
Nova	Compute	Core	It handles the lifecycle of virtual machines, with various operations, such as creation, deletion, and resource scheduling.
Neutron	Networking	Core	It handles networking operations and provides application programming interfaces (APIs) for other OpenStack projects to interact with the IaaS networking components. It typically integrates with third-party vendors, such as Brocade or Cisco, to provide software defined switches and routers.
Swift	Object Storage	Core	It provides a Representational State Transfer (REST) interface to store objects in virtual containers. It supports a horizontal scaling architecture, and provides immediately available redundancy mechanisms to replicate Object Storage.
Cinder	Block Storage	Core	It provides persistent storage to the virtual machines, with a pluggable architecture that interacts with major storage vendors, such as EMC and VMware.
Keystone	Identity Service	Core	It provides the main authentication mechanism for OpenStack. Exposes APIs that allow the creation of users, projects, groups of users and roles.
Glance	Image Service	Core	It provides image cataloging capabilities, defining a registry and repository where administrators can register the OS base images, which are used at provisioning time.
Horizon	Dashboard	Optional	It provides a web-based console for accessing and interacting with all of the different components, mostly intended for administrators and regular users.
Ceilometer	Telemetry	Optional	It handles metering and telemetry services. These services enable administrators to retrieve fine-grained information about resource usage within the OpenStack cloud. Ceilometer can typically be used for IaaS billing.
OpenStack Heat	Orchestration	Optional	It orchestrates multiple composite cloud applications by using either the Heat Orchestration Templates (HOT) format or the Amazon Web Services (AWS) CloudFormation (CFN) template format, through either an OpenStack-native REST API or a CloudFormation-compatible Query API. OpenStack Heat is further described in 1.2.5, "OpenStack Heat" on page 13.

The number of OpenStack projects is constantly expanding as new projects are being developed. The list of OpenStack projects and services is not exhaustive. The complete list can be found at the following website:

<https://www.openstack.org/software/project-navigator/>

Each OpenStack service provides an API that enables you to deploy infrastructure components that combine only the projects that are required. Typically, infrastructure deployments are controlled through a graphical user interface (GUI) called the OpenStack Dashboard (provided by the Horizon project), shown in Figure 1-3, or a command-line interface (CLI), shown in Figure 1-4.

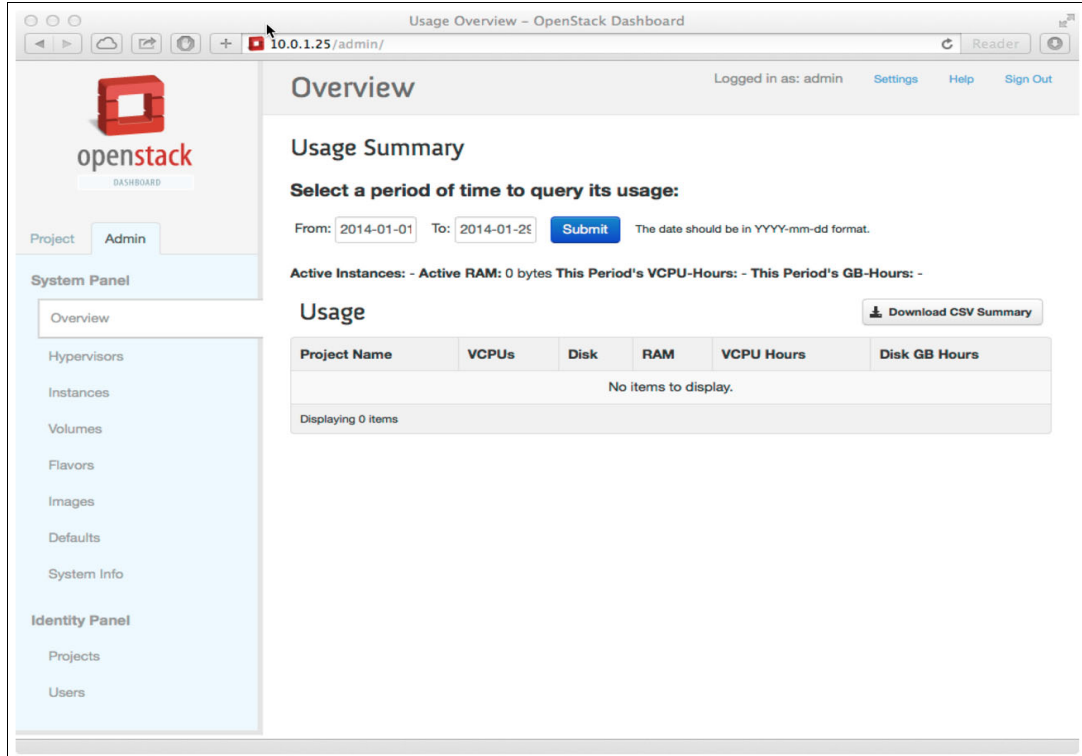


Figure 1-3 OpenStack Dashboard GUI

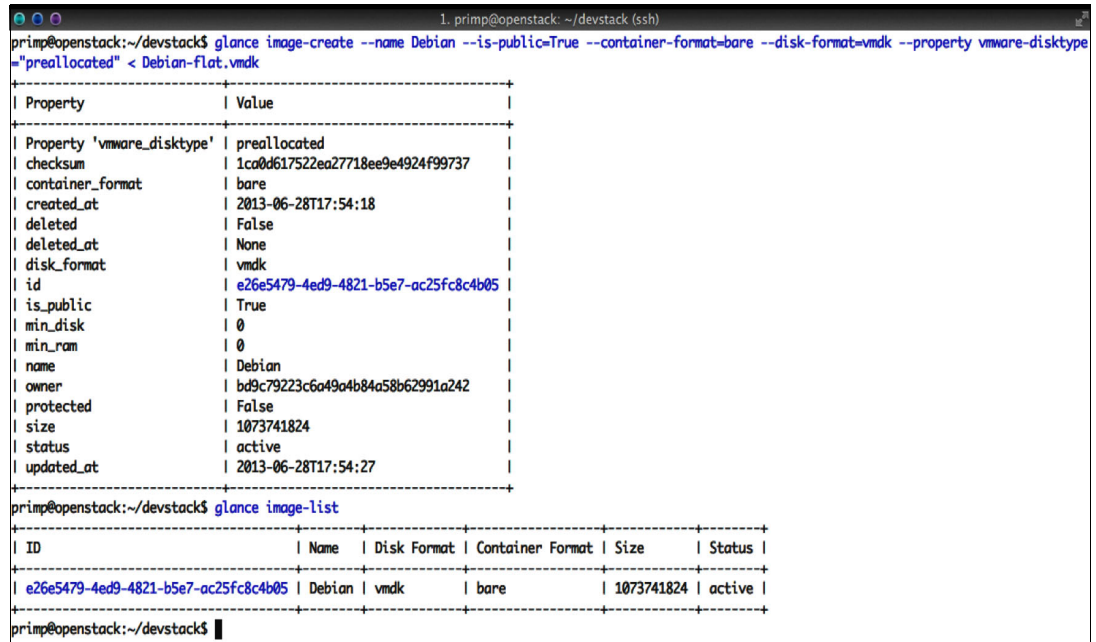


Figure 1-4 OpenStack CLI

1.2.2 IBM Blue Box

When working with an OpenStack cloud, one of the main architectural decisions involves the deployment method:

- ▶ The version that is selected requires expertise to be installed properly and has a learning curve for the organization.
- ▶ There are several *turnkey* solutions that include all that needs to be done to run OpenStack as a private cloud as a service (PCaaS).

The latter is the case of IBM Blue Box, which provides a streamlined set of processes and tools to deliver an enterprise-grade OpenStack runtime, on top of any layer of hardware and data center type, with emphasis in the highest standards for security, redundancy, compliance, and ease of use. An example of the Blue Box console that is used to manage an OpenStack cloud is shown in Figure 1-5. For more information about IBM Blue Box, see the following website:

<https://www.blueboxcloud.com/>

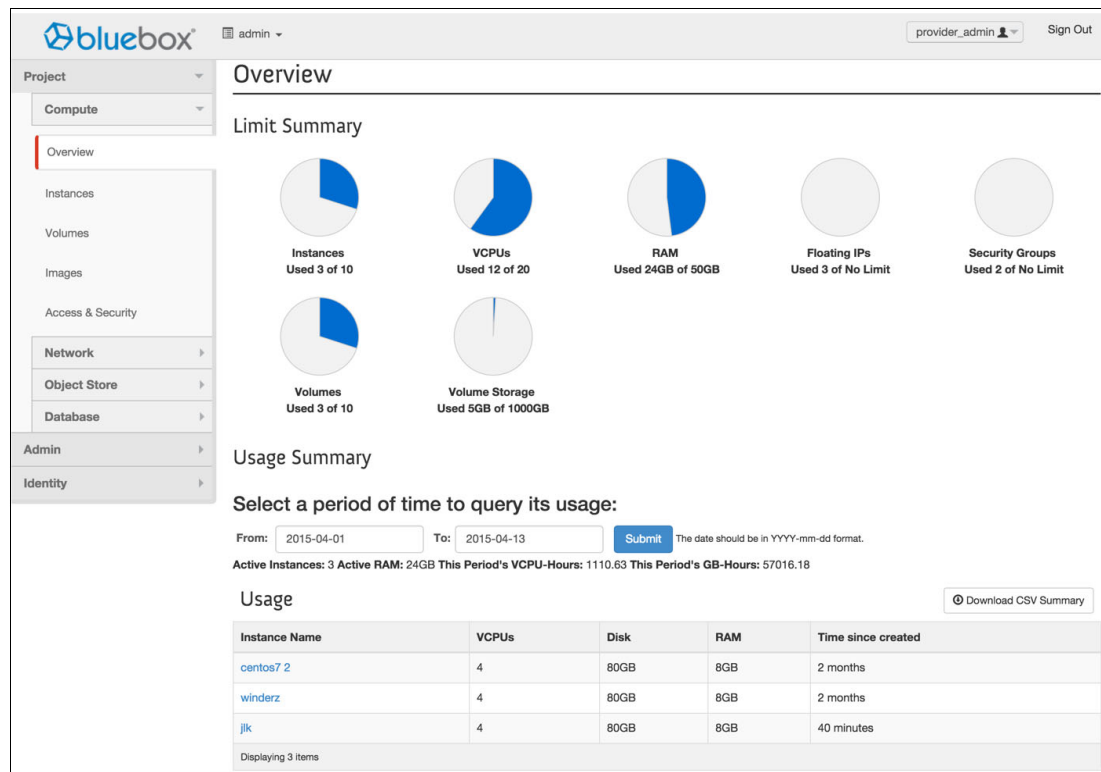


Figure 1-5 Blue Box OpenStack console

Figure 1-6 shows IBM Blue Box support for the following elements:

- ▶ OpenStack projects
- ▶ Hypervisors
- ▶ Operating systems

OPENSTACK API COVERAGE	
Block Storage API & Extensions	Cinder v2.0
Compute API & Extensions	Nova v2.0
Identity Service API & Extensions	Keystone v3.0
Image Service API	Glance v2.0
Networking API & Extensions	Neutron v2.0
Object Storage API & Extensions	Swift v1.0
Orchestration Service	Heat v2.0
Telemetry Service	Ceilometer v2.0
SUPPORTED HYPERVISORS KVM, QEMU	
SUPPORTED GUESTS Linux, Windows	

Figure 1-6 Blue Box supported OpenStack projects, hypervisors, and OS

IBM Blue Box is delivered in two main form factors:

- ▶ Blue Box dedicated

A deployment model aimed to provide the fastest path to get into an OpenStack cloud. It is a single, dedicated customer environment, preconfigured and automatically deployed, that provides resources for compute, storage, network, and security. Blue Box dedicated is hosted in an IBM SoftLayer data center:

<http://www.softlayer.com>

- ▶ Blue Box Local

Deployed on dedicated hardware and networking infrastructure that is controlled by customers within their premises. Blue Box local is remotely managed, meaning that IBM can access and maintain the Blue Box component, leaving customers in control of the deployed workloads.

The configurations for the Blue Box local deployment model are specific to the chosen hardware. The main hardware choice for Blue Box local described in this publication is Bluemix Local System, and information about it is provided in 1.3, “IBM PureApplication System products for OpenStack” on page 16.

1.2.3 Docker

Docker is an open source container deployment and management technology. It enables developers to build, send, and run applications, abstracting them from their runtime, such as a notebook or a virtual machine. Docker can manage multiple software stacks, such as a data center virtual machine (VM) or public or private cloud on PureApplication System offerings.

During the build phase, developers can create an application by using a Docker container, send it in a standard way (in Docker application format), and run workflows. Docker containers can also be shared with other people and communities.

Docker is a lightweight alternative for running software in a portable, though less isolated, virtual environment. Docker can run on a hypervisor or directly on the host OS. It is an optional component. Figure 1-7 shows a high-level comparison of the traditional virtualization approach with the Docker container approach.

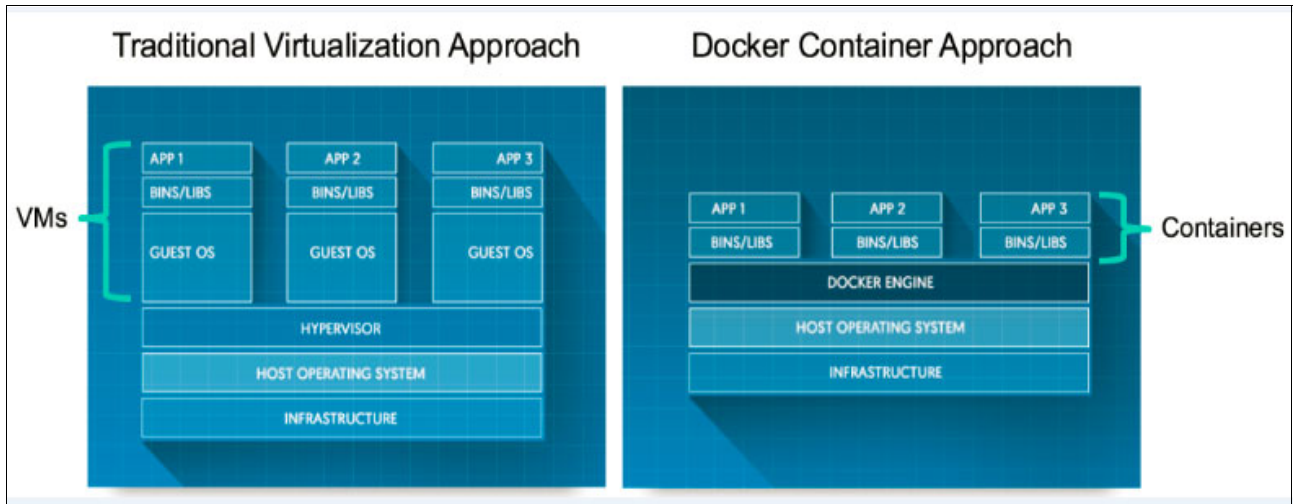


Figure 1-7 Comparison between virtualization with virtual machines and Docker containers

Table 1-2 shows a high-level comparison of the traditional virtualization approaches with the Docker container approach.

Table 1-2 High-level comparison between virtual machine and Docker containers

Attribute	Virtual machine	Docker container
Startup time and performance	Slow (minutes to start) because of the hypervisor processing requirements	Fast (seconds to start): No hypervisor processing
Footprint	<ul style="list-style-type: none"> ▶ Large (nothing is shared in terms of software) ▶ One guest OS for each VM 	Small (OS kernel is shared)
Portability	Low	High
Isolation and security	High	Medium
Resource constraints	Yes	Yes (CPU and memory)

Table 1-2 is further explained in the following list:

- ▶ **Startup time and performance**
Faster in Docker because containers do not need a hypervisor. VMs always include the necessary dependencies, such as the VM’s binary files, libraries, and the guest OS.
- ▶ **Footprint**
Docker does not have an OS dependency on deployment because it is on Linux Containers (LXC) low-level functions. This approach yields a smaller footprint than VMs, where middleware and application software link to the OS of each VM instance.

► Portability

Docker has a greater level of portability than VMs. VMs need to be customized by separating the software installation from the configuration steps on a full operating system, because applications and middleware are tied to the OS image in those cases.

► Isolation and security

Isolation by using VMs, based on hypervisor technology, is more secure than isolation provided by Docker containers, because no OS-level or middleware-level sharing of resources occurs.

► Resource constraints

Full OS and memory management are installed in VMs with the associated resources required for virtual device drivers. This architecture causes more resource constraints than Docker because Docker containers run with the Docker Engine rather than with the hypervisor. Docker has resource constraints that relate to memory and CPU.

For more information about Docker and its applications in PureApplication *Patterns of Expertise*, see the IBM Redbooks publication *Establishing a Secure Hybrid Cloud with the IBM PureApplication Family*, SG24-8284:

<http://www.redbooks.ibm.com/abstracts/sg248284.html>

1.2.4 SaltStack

Handling communications across enterprise systems is a complex task, and doing it effectively and quickly presents additional complexities. System administrators need tools that enable them to create automations for repeatable tasks.

For operations of this nature, open source tools, such as SaltStack,³ provide a remote execution engine that creates a bidirectional channel between a *master* server and a set of *slave* servers. This engine provides configuration and application management capabilities.

The current version at the time of writing of this paper is Beryllium (2015.8.0). Full documentation on this version is available at the following website:

<https://docs.saltstack.com/en/latest/contents.html>

To understand SaltStack, it is useful to define the concepts that are listed in Table 1-3.

Table 1-3 SaltStack concepts

Concept	Description
Master	Main server that controls the infrastructure and SaltStack application engine, working as a repository for data and control point for operations between the Minions.
Minion	System process that enables communication with the master and the extraction of instructions that are received from the master. It runs in a virtual machine or a container at the time of deployment.
State	Set of rules and configurations that describe a <i>desired state</i> for a particular minion.
Formula	A collection of states. For example, the steps that need to be followed to install a program. Each step is represented by a state.

³ SaltStack, <https://saltstack.com>

Concept	Description
Grain	Interface to derive information about the underlying system where a minion is hosted. This information is sent back to the master. Most grains are auto configured: For example, the host name and the default gateway of a system. It is also possible to create custom grains.
Pillar	Essentially the reverse of a grain: Information that is passed by a master down to a minion. Minions cannot get pillar data from other minions, Only from the master. There is no auto-configured pillar data, only custom.
Execution Functions	Commands running specific functions.
Execution Modules	Groups of execution functions that are grouped by affinity.
Reactor	A function that can be used to extend an existing execution function or module, and which can be exposed as a REST API. For example, a reactor to pass a key to a minion and make it accept it automatically.
Mine	A database that is running on the master to keep information directly in the master. It is used for the kind of information that is not yet available at the time of the minion deployment, which would otherwise be part of the pillar data.

Running individual functions is a manual process and doing it over and over in the entire infrastructure would be impractical. For this reason, SaltStack provides a mechanism to apply a desired configuration to a system. This mechanism is defined in Table 1-3 on page 12 as a *state*. A state is a data structure of variables and values to be applied into a system, to transition it into the desired configuration. The role of the minion is to *bring* the system where it runs into the desired state, by using execution functions and modules.

SaltStack provides a ready-to-use set of execution modules for the master to interact with the minions. Typical functionalities of these modules include the following tasks:

- ▶ Query information
- ▶ Apply configuration
- ▶ Install software on a minion
- ▶ Identify storage
- ▶ Setup network
- ▶ As mentioned, execution modules and functions can be extended by using reactors

SaltStack is further described in Chapter 6, “Creating automation for OpenStack Heat patterns with SaltStack” on page 65.

1.2.5 OpenStack Heat

Heat is an OpenStack project that handles *orchestration*, aiming to provide interfaces to control the entire lifecycle of infrastructure and application deployment. Heat allows users, clients, and other OpenStack projects to use resources in a controlled fashion, with special emphasis on *templates* that serve as the foundation for infrastructure deployment.

Orchestration entails a rich set of operations to be performed when deploying infrastructure and applications, such as creation and configuration of virtual machines, storage components, network components, and data injection.

To understand OpenStack Heat, it is useful to define the concepts that are listed in Table 1-4.

Table 1-4 OpenStack Heat concepts

Concept	Description
Template	Main building block of the Heat orchestration. It defines in text files, following the format that is defined in the next row, all of the items that create a <i>stack</i> , before its deployment.
Format	Heat supports two formats: Heat Orchestration Templates (HOT), which are written in Yet Another Markup Language (YAML), and AWS CloudFormation (CFN), which is written in JavaScript Object Notation (JSON).
Stack	Collection of objects that are being created by a heat template when deployed. It might include networks, routers, ports, security groups, and virtual machines.
Parameters	Parameters enable you to specify information specific to a single stack in order to customize a template. For example, if a stack uses a network, its definition would include the parameter ID for the network.
Environment	Collection of structured parameters that are grouped into a YAML file. This file is applied to a template to customize it, therefore affecting the deployment of a specific stack.
Resources	Each stack, when created, contains several resources, which are the items that have been provisioned, such as networks and virtual machines.
Scaling policy	Defines scaling properties for a stack, or a part of a stack (a <i>scaling group</i>).

The following list describes the main architectural components of OpenStack Heat:

- ▶ The *heat-api*
It provides a REST API that processes orchestration requests by sending them to the heat-engine.
- ▶ The *heat-api-cfn*
It provides a query API compatible with AWS CloudFormation, and processes orchestration requests by sending them to the heat-engine.
- ▶ The *heat-engine*
It performs the main work of orchestrating the launch of templates and providing events back to the API consumer.

Note: AWS CloudFormation support is not a focus of this paper.

OpenStack Heat also provides capabilities for the scalability of a stack. It provides these capabilities either *vertically*, where resources of a stack (such as memory or processors) are augmented in capacity, or *horizontally*, where more resources are added into the stack in quantity (such as increasing the number of virtual machines). This is attained through what in OpenStack Heat concepts is defined as *scaling policies* and *scaling groups*.

Note: Typically, automated scaling in OpenStack uses the OpenStack Ceilometer project, but this is not a requirement. For example, Zeron could be used instead, as detailed in 1.2.6, “Zeron” on page 15.

1.2.6 Zeron

In a typical *DevOps* paradigm, post-deployment management provides a way of running and administering *operations* on pattern instances and their components. Although performing this type of operation manually is not a significant resource cost for a single virtual machine, operations on complex topologies entail a high degree of complexity.

When dealing with patterns, the complexity and variety of the topologies that are specified by the patterns can result in a significant management burden, if operations are performed manually. The nature of this burden is in both the complexity of the manual operations themselves and in the number of specific tools that are needed to perform those operations on the single pattern components.

When dealing with OpenStack Heat patterns, the open source nature of the pattern element is another driver to the number of tools that are needed for post-deployment operations. As an example, in a pattern that runs a node.js application alongside an IBM DB2® component, two different tools would need to be installed and configured to manage operations on the two types of middleware.

To largely simplify the complexity of post-deployment operations, and to reduce the number of tools that are needed to manage those operations to a minimum footprint, IBM designed Zeron. Zeron is a lightweight, extensible, and pluggable lifecycle management framework designed for post-deployment management of patterns, virtual machines, and containers. It is specifically designed for OpenStack Heat patterns.

Zeron provides the following functionality for all virtual machines and containers in a pattern:

- ▶ Access to operations
- ▶ Consolidated logging
- ▶ Consolidated monitoring
- ▶ Fix management
- ▶ License tracking

The framework is designed to be extremely lightweight, with a footprint of about 5 megabytes (5 MB). The lightweight nature of Zeron is due to the fact that the framework does not directly provide the operations, but only a consolidated interface for those operations to be accessed and triggered. In order for the framework to be functional, operations need to be exposed at the pattern level.

Zeron is written in Python and can be extended through the creation of custom handlers. This makes the framework open in nature.

In addition, Zeron relies on OpenStack to provide security and authentication, by leveraging the OpenStack Keystone project. For more information about Keystone, see 1.2.1, “OpenStack” on page 6 and the following website:

<http://keystone.openstack.org>

Zeron is described in further detail in Chapter 7, “Zeron for OpenStack Heat patterns” on page 75.

1.3 IBM PureApplication System products for OpenStack

The IBM PureApplication System family of product offerings is designed to accelerate the deployment of enterprise applications. More information about the entire family of systems can be found on the following web address:

<http://www.ibm.com/software/products/en/pureapplication>

There are four members of the PureApplication family, as shown in Figure 1-8:

- ▶ IBM PureApplication Software
- ▶ IBM Bluemix Local System
- ▶ IBM PureApplication System
- ▶ IBM PureApplication Service

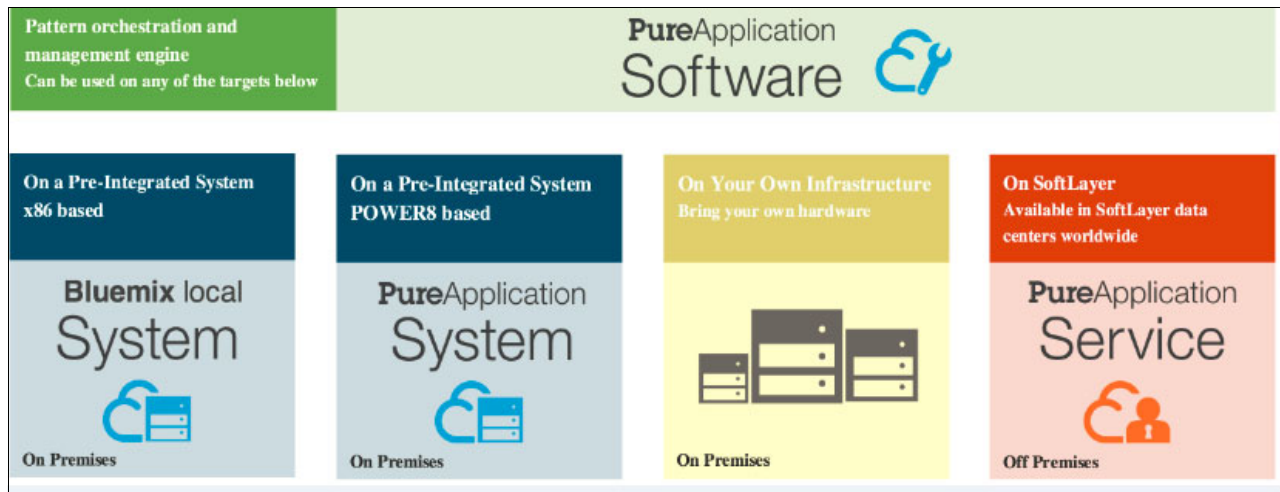


Figure 1-8 IBM PureApplication System family members

PureApplication Software is the orchestration and management engine, which can be used across different target hardware:

- ▶ On a pre-integrated system:
 - x86 (Intel): Bluemix Local System
 - Power8: PureApplication System
- ▶ On your own x86 or IBM Power hardware
- ▶ On SoftLayer's PureApplication Service infrastructure

Each product offering has its own strength and each product complements the others to provide a full approach to build a complex hybrid cloud solution. Whichever product you choose, the user experience that is provided by PureApplication Software remains the same, making the transition from one IBM PureApplication System product to another easy and straightforward. At the time of writing of this paper, the current version of the PureApplication Software is 2.2.2.

More information about the PureApplication family and its use can be found in *Establishing a Secure Hybrid Cloud with the IBM PureApplication Family*, SG24-8284:

<http://www.redbooks.ibm.com/abstracts/sg248284.html>

For the topics treated in this paper, we focus on PureApplication Software on Bluemix Local System. Using PureApplication Software on Bluemix Local System delivers an application platform in a rack form factor. It is a ready-to-use solution that merges the following components:

- ▶ Pre-integrated infrastructure
- ▶ System management capabilities
- ▶ *Patterns of Expertise*, such as IBM WebSphere Application Server and DB2, created by IBM and non-IBM vendors, and built-in tools to easily customize those patterns

At the infrastructure level, Bluemix Local System is a pre-integrated machine, which includes compute nodes, storage, and networking. Since the inception of PureApplication, IBM released three generations of pre-integrated systems, each with two choices of chipset technology: Intel-based and Power-based. IBM Bluemix Local System is the third generation of the Intel-based system.

In addition to the capabilities described previously, Bluemix Local System with PureApplication Software delivers an OpenStack runtime, also known as *OpenStack services*, using the IBM Blue Box technology described in 1.2.2, “IBM Blue Box” on page 9. It is in fact possible to dedicate several compute nodes on a Bluemix Local System to what essentially becomes a Blue Box local implementation of an OpenStack runtime.

Note: It is important to note that OpenStack services on Bluemix Local System are a managed deployment, completed in collaboration with Blue Box. This means that remote access to the system must be granted to the Blue Box operations team.

More information around the setup for OpenStack service on Bluemix Local System using PureApplication Software is provided in Chapter 2, “Setting up IBM Bluemix Local System for OpenStack Heat patterns” on page 25.

Note: At the time of writing of this paper, OpenStack services are only available on Intel-based Bluemix Local Systems, and not on Power-based PureApplication Systems.

In a similar way to OpenStack services, it is possible to reserve several compute nodes in a Bluemix Local System to run a dedicated Bluemix Local partition, as shown in Figure 1-9.

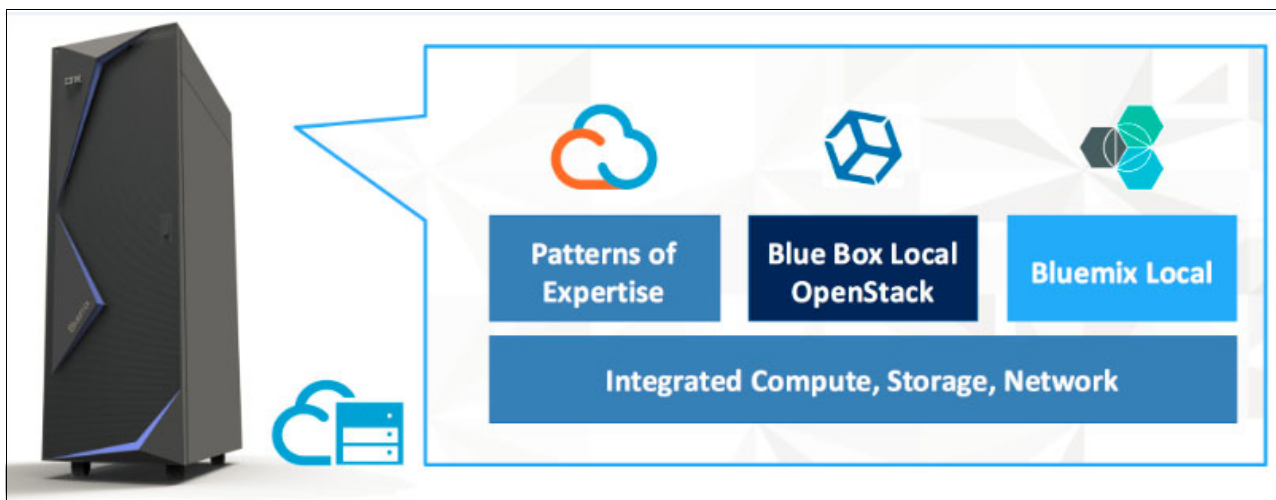


Figure 1-9 IBM Bluemix Local System: A single platform for three runtimes

Bluemix Local on IBM Bluemix Local System is not treated as part of this paper. For more information about this subject, see the following websites:

<http://www.ibm.com/bluemix>

<http://www.ibm.com/bluemix/local>

Note: At the time of writing of this paper, Bluemix Local is only available on Intel-based Bluemix Local Systems.

1.4 OpenStack Heat patterns

IBM *Patterns of Expertise* provide an enterprise-strong mechanism to deliver, deploy, and manage complex applications and their topology. However, the key components of the Patterns of Expertise are proprietary to IBM.

Based on the ever-growing popularity of open source technologies that deliver some of the capabilities that are needed by a pattern, IBM introduced *OpenStack Heat patterns*, which deliver the same capabilities, but are based on open source technologies.

OpenStack Heat patterns are based on OpenStack for the infrastructure layer. At the time of writing of this paper, the only choice for the hypervisor layer is KVM.

OpenStack Heat patterns provide an end-to-end design, deployment, and lifecycle management experience similar to the one available for the IBM Patterns of Expertise, which none of the open source technologies described in this chapter can currently deliver in isolation.

To run OpenStack Heat patterns, an *OpenStack Heat pattern runtime* is needed. The OpenStack Heat pattern runtime implementation that is described in this paper makes specific technology choices, alongside the OpenStack choice for the infrastructure layer and the choice for the hypervisor layer. Those choices are described in the following list and depicted in Figure 1-10 on page 19:

- ▶ Red Hat or Ubuntu for the OS layer
- ▶ Standard Software packages for software installations
- ▶ SaltStack for automation scripting
- ▶ OpenStack Heat for orchestration scripting on OpenStack infrastructure
- ▶ Zeron for post-deployment management

Note: The OpenStack infrastructure and the hypervisor layer are not part of the OpenStack Heat pattern runtime. However, they are part of the OpenStack-based *deployment target* for the OpenStack Heat patterns. *Deployment targets* supported by the OpenStack Heat pattern runtime are described in 1.4.1, “OpenStack Heat patterns on IBM Bluemix Local System with PureApplication Software” on page 21 and 1.4.2, “OpenStack Heat patterns on IBM Bluemix” on page 22.

Figure 1-10 shows the technologies used in this paper.

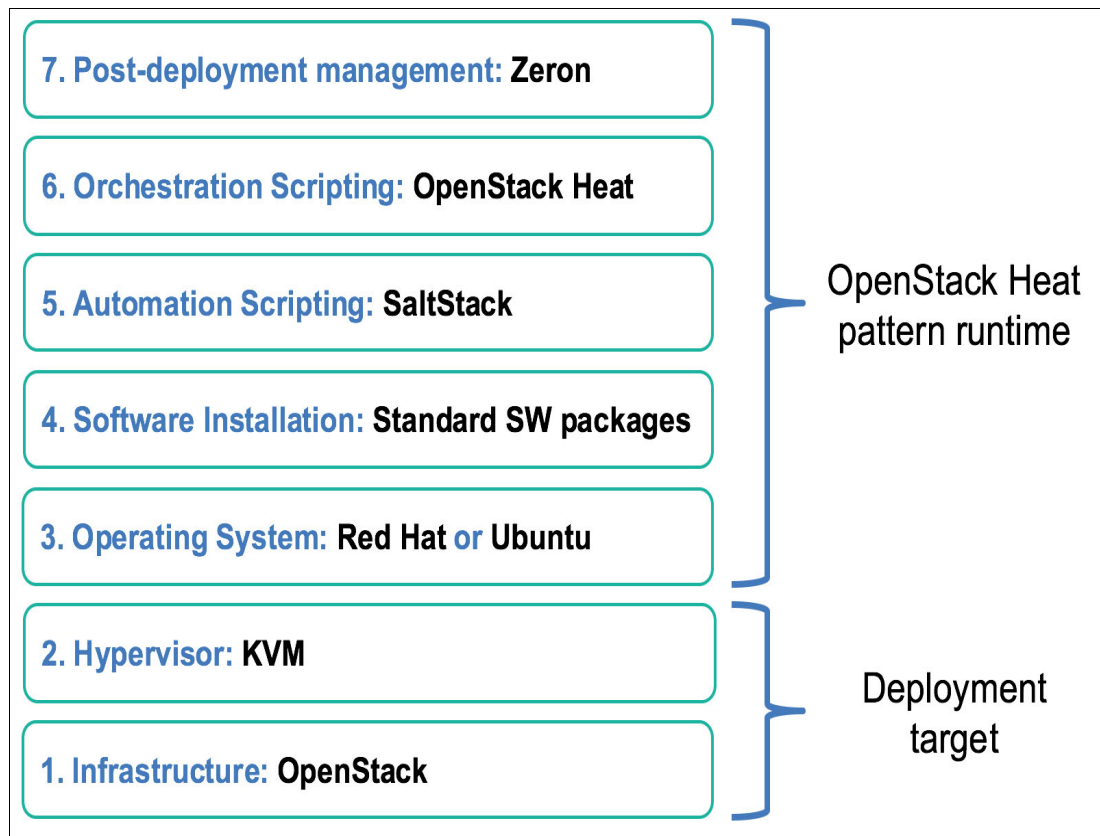


Figure 1-10 OpenStack Heat pattern runtime implementation on OpenStack and chosen hypervisor

The OpenStack Heat pattern runtime that is described in Figure 1-10 uses technologies that are tested, and are supported when using OpenStack Heat patterns. However, it might be possible to extend some of these choices, and to use alternative technologies. It is important to note that, although creating OpenStack Heat patterns using different technologies is possible, this choice is not tested for the described runtime.

The following list describes the main layers where the OpenStack Heat patterns could be extended:

- Operating System

Bringing Your Own OS (BYOOS), such as openSUSE and CentOS:

<https://www.centos.org/>

- Software installation

Using containers rather than standard software packages for the delivery of applications. Patterns of expertise already provide support for Docker, and this is an area of current development for OpenStack Heat patterns, as well.

- Automation scripting

In the same way that SaltStack agents communicate with the Salt master, different *desired-state* management system agents can be used within OpenStack Heat patterns, provided a central server is available in the organization. This could be done, for example, by using *desired-state* management systems, such as Chef and Puppet. The same applies to other systems, such as Ansible, which do not use agents.

In addition to the set of open technologies that are listed in Figure 1-10 for the OpenStack Heat pattern runtime, an overall *OpenStack Heat pattern engine* also needs to provide capabilities to create and manage OpenStack Heat patterns at *design time*. The OpenStack Heat pattern engine that is described in this paper provides two additional capabilities:

- ▶ A version control system for pattern artifacts, such as Heat templates and SaltStack scripts. The technology used to implement this system is Git:
<https://git-scm.com/>
- ▶ A graphical pattern editor, to enable pattern developers to create OpenStack Heat patterns that use all technology components. The pattern editor technology is based on the IBM UrbanCode™ Deploy (UCD) Blueprint Designer.

The resulting *OpenStack Heat pattern engine* is illustrated in Figure 1-11.

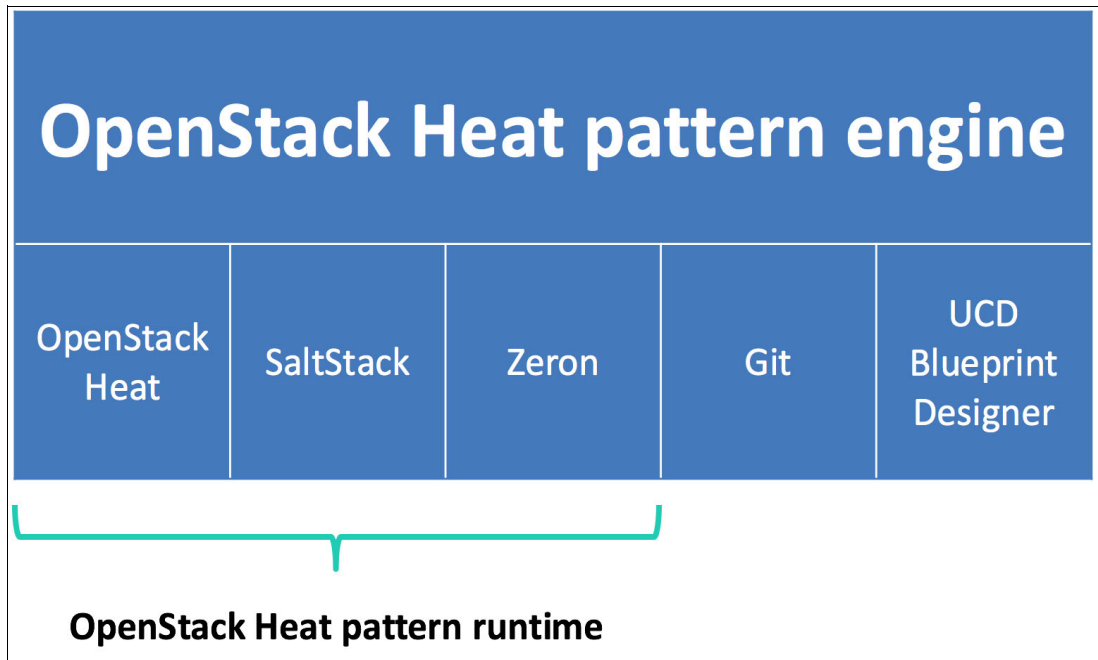


Figure 1-11 *OpenStack Heat pattern engine*

Note: The *OpenStack Heat pattern engine* includes the *OpenStack Heat pattern runtime* and the additional capabilities that are listed in 1.4, “OpenStack Heat patterns” on page 18.

After OpenStack Heat patterns are created, they can be deployed to a *deployment target*. The idea behind the choice of OpenStack infrastructure at the base of a deployment target is that OpenStack-based stacks can be created and deployed on any hardware. This gives the OpenStack Heat patterns a unique portability across every type of infrastructure component, and IaaS in general, that can be defined through OpenStack.

However, it is important to notice that for the OpenStack Heat patterns to be able to run, a pattern engine, such as the one described in Figure 1-12 on page 21, needs to be set up and configured. The way a pattern engine is implemented is not mandated by IBM. If the key technology choices described in Figure 1-10 on page 19 are followed, OpenStack Heat patterns can be run and managed through any pattern engine implementation that uses an OpenStack-based deployment target.

This is a key strength of the OpenStack Heat pattern paradigm: Both the deployment target and the engine are based on open source technologies. Therefore, if the open technologies present in the OpenStack Heat patterns are used, the patterns can be deployed anywhere.

At the time of writing this publication, IBM has two platforms that, in addition to running OpenStack-based stacks, provide a preconfigured pattern engine implementation, which follows the technology choices that are described in Figure 1-11 on page 20. Those platforms are described in the following list, and depicted in Figure 1-12:

- ▶ IBM Bluemix Local System with PureApplication Software
- ▶ IBM Bluemix (experimental service only)

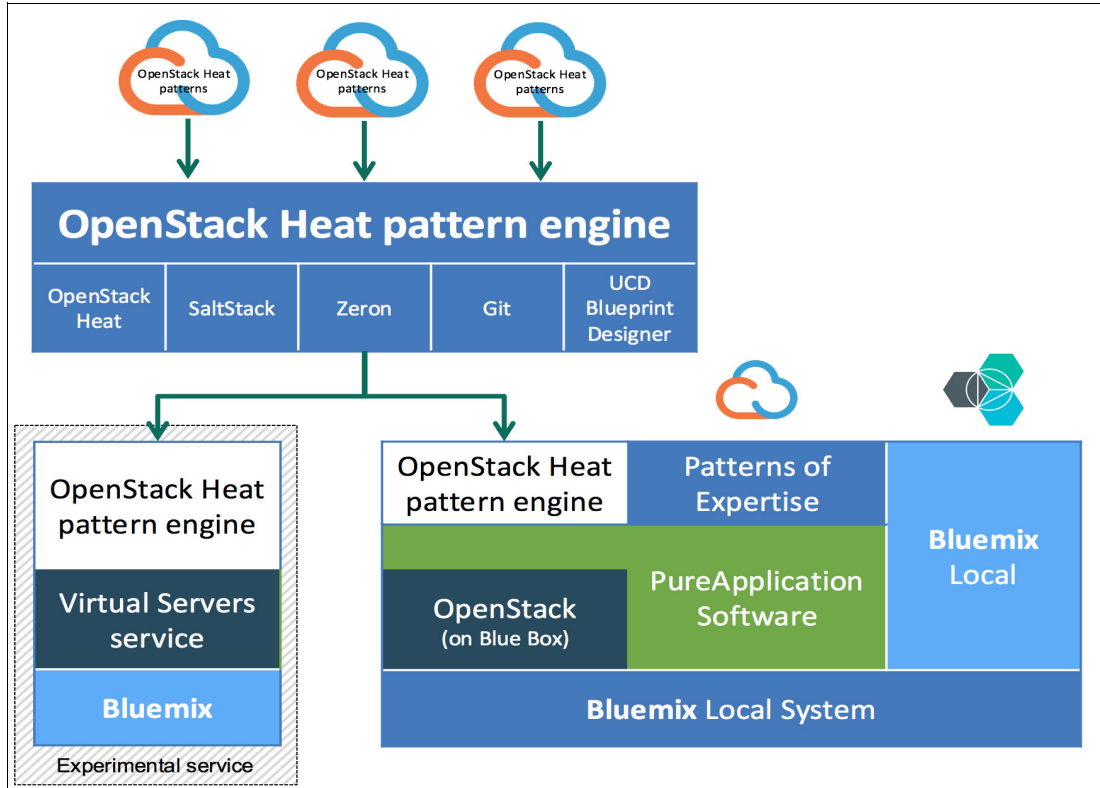


Figure 1-12 Platform running a preconfigured pattern engine

Note: The implementation of the pattern engine differs slightly for the two platforms that are shown in Figure 1-12.

These platforms are further described in the remainder of this chapter.

1.4.1 OpenStack Heat patterns on IBM Bluemix Local System with PureApplication Software

In 1.3, “IBM PureApplication System products for OpenStack” on page 16, we described the capability to run OpenStack services on IBM Bluemix Local System with PureApplication Software. In essence, an IBM Blue Box local implementation can be set up on a Bluemix Local System to create and manage an OpenStack runtime.

Because it delivers an OpenStack runtime, IBM Bluemix Local System can act as the deployment target for OpenStack Heat patterns, as shown in Figure 1-10 on page 19.

However, in addition to the OpenStack services, the PureApplication Software on Bluemix Local System also makes available an implementation of the pre-configured pattern engine described in Figure 1-12 on page 21.

The fundamental advantage of using this type of platform, such as Bluemix Local System for OpenStack Heat patterns, is that most of the infrastructure and pattern prerequisites are satisfied by the underlying converged infrastructure, enabling developers to focus on the build-out and deployment of the patterns and the applications.

The main focus of this paper and the following chapters is on the creation and management of OpenStack Heat patterns on Bluemix Local with PureApplication Software.

1.4.2 OpenStack Heat patterns on IBM Bluemix

Bluemix is the IBM cloud platform as a service (PaaS) that offers mobile and web developers access to IBM software for integration, security, transactions, and other key functions, and software from IBM Business Partners:

<http://www.ibm.com/bluemix>

Bluemix provides several *composable* services for developers and businesses to create, manage, and deploy their own applications in the cloud. Bluemix provides the following features:

- ▶ A range of services to build and extend web and mobile apps quickly
- ▶ Processing power to deliver app changes continuously
- ▶ *Fit-for-purpose* programming models and services
- ▶ Manageability of services and applications
- ▶ Optimized and elastic workloads

For more information about Bluemix, see *IBM Bluemix: The Cloud Platform for Creating and Delivering Applications*, REDP-5242:

<http://www.redbooks.ibm.com/abstracts/redp5242.html>

Bluemix offers two services that are particularly relevant to the topic described in this paper:

- ▶ The Bluemix Virtual Servers service:
<http://ibm.biz/VirtualServers>
- ▶ The Bluemix Pattern Engine service

The IBM Virtual Servers service enables you to create, run, and manage virtual servers and virtual machines in public private clouds. This service is built on OpenStack, and can use Heat templates.

Note: The Virtual Servers service on Bluemix is in beta at the time of writing of this paper.

Because it delivers an OpenStack runtime, the Virtual Server service on Bluemix can act as the deployment target for OpenStack Heat patterns, as defined in Figure 1-9 on page 17.

The IBM Bluemix Pattern Engine provides an implementation of the pre-configured pattern engine that is described in Figure 1-10 on page 19.

Note: The Pattern Engine service on Bluemix is an experimental service at the time of writing of this paper. An experimental service is introduced for testing purposes only and provides limited support.

It is important to note that the OpenStack Heat pattern engine that is provided by the Pattern Engine service is slightly different from the one provided by PureApplication Software, despite making the same technology choices.

The main differences from the PureApplication Software OpenStack Heat pattern engine on Bluemix Local System are listed as follows:

- ▶ Some of the OpenStack Heat and SaltStack configurations are different, because in the Virtual Server service no direct access to the OpenStack Heat engine is permitted.
- ▶ OpenStack Heat patterns cannot be uploaded to the engine, only the sample ones can be used.
- ▶ OpenStack Heat patterns cannot be edited.
- ▶ Because no pattern editing is possible, the IBM UrbanCode Deploy (UCD) Blueprint Designer is not provided as part of the pattern engine.

OpenStack Heat patterns on the IBM Bluemix Pattern Engine are not described in further detail in this paper.

1.5 Conclusions

In this chapter, we describe the key elements that are needed to create a *pattern*, together with what makes a pattern reusable and portable. We provide a primary example of a good pattern incarnation, in the IBM *Patterns of Expertise*.

Various technology choices, primarily open source, are provided for the implementation of the key elements defined previously. Throughout the chapter, the most relevant of those technologies are further described.

Therefore, we have introduced *OpenStack Heat patterns*: Patterns created primarily by using open technologies, using OpenStack. We also define various concepts, such as a pattern deployment target, an *OpenStack Heat pattern runtime*, and an *OpenStack Heat pattern engine*. Finally, we describe the two IBM platforms that provide a deployment target and an engine implementation for OpenStack Heat patterns: Bluemix Local System and Bluemix.

In the remainder of this paper, we describe in more detail OpenStack Heat patterns, focusing in particular on PureApplication Software on Bluemix Local System:

- ▶ In Chapter 2, “Setting up IBM Bluemix Local System for OpenStack Heat patterns” on page 25, we provide guidance about how to set up Bluemix Local System to provide an OpenStack runtime by using IBM Blue Box.
- ▶ In Chapter 3, “Deploying sample OpenStack Heat patterns” on page 35, the deployment of a sample OpenStack Heat pattern is described.
- ▶ In Chapter 4, “Importing OpenStack Heat patterns” on page 47, we describe how to import a Heat template (HOT) and transform it into an OpenStack Heat pattern.
- ▶ In Chapter 5, “Creating an OpenStack Heat pattern” on page 53, the steps to create a new OpenStack Heat pattern are detailed and described.
- ▶ In Chapter 6, “Creating automation for OpenStack Heat patterns with SaltStack” on page 65, we provide further detail about SaltStack, and describe how it operates in the context of PureApplication Software.
- ▶ Chapter 7, “Zeron for OpenStack Heat patterns” on page 75 provides an in-depth review of Zeron and how it can be used to manage operations on OpenStack Heat patterns after they are deployed.



Setting up IBM Bluemix Local System for OpenStack Heat patterns

In Chapter 1, “Automated application delivery with OpenStack Heat patterns” on page 1, we define the *OpenStack Heat pattern runtime* as the runtime implementation able to run OpenStack Heat patterns based on specific technology choices. Additionally, we define the *OpenStack Heat pattern engine* as a combination of the pattern runtime and a specific set of tools for OpenStack Heat pattern editing, such as Blueprint Designer in IBM UrbanCode Deploy (UCD), and version control system, such as Git.

When setting up IBM Bluemix Local System with IBM PureApplication Software, all the constituents of the *OpenStack Heat pattern engine* are created, including the OpenStack runtime, the Zeron framework, the SaltStack master, UCD Blueprint Designer, and the necessary Git repositories. It is worth noting that, at the time this paper was written, the OpenStack implementation that is used in Bluemix Local System, IBM Blue Box, does not support the OpenStack Cinder project. Therefore, adding storage volumes to OpenStack Heat patterns is not possible.

The *OpenStack Heat pattern engine* implementation in PureApplication Software gets created at setup time on the IBM PureApplication System Manager (PSM) virtual machine (VM). This VM contains the SaltStack master for the OpenStack Heat patterns, a web server for the software binary files, and four Git repositories, which are as follows:

- ▶ Pattern repository, where OpenStack Heat patterns are stored
- ▶ Module repository, where SaltStack execution modules are stored
- ▶ Pillars repository, where SaltStack pillars are stored
- ▶ Formulas repository, where SaltStack formula are stored

The pattern repository is described in more detail in Chapter 5, “Creating an OpenStack Heat pattern” on page 53, and the three repositories that are related to SaltStack are described in 6.2.5, “Git repositories and SaltStack files” on page 70.

This chapter describes the steps that are needed to set up the OpenStack Heat pattern engine on Bluemix Local System for the first time.

2.1 Setting up OpenStack on Bluemix Local System

As mentioned in Chapter 1, “Automated application delivery with OpenStack Heat patterns” on page 1, PureApplication Software V2.2.1 and later on Bluemix Local System provides an OpenStack runtime. OpenStack Services are accessible from the System menu. Go to **System** → **System Settings** to see the OpenStack Services, as shown in Figure 2-1.

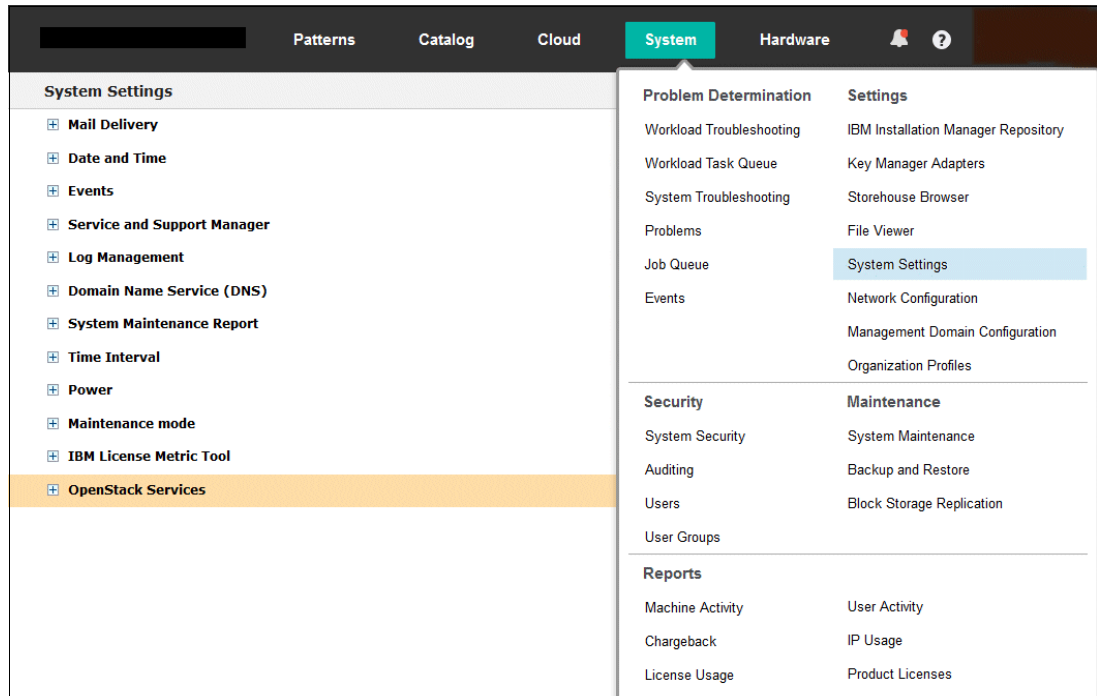


Figure 2-1 System → System Settings → OpenStack Services menu option

This section describes the steps to enable OpenStack services on Bluemix Local System.

2.1.1 Preparing Bluemix Local System

In this section, we detail the steps that are needed to prepare Bluemix Local System before enabling OpenStack services. To learn more about networking, and in particular virtual local area networks (VLANs), Internet Protocols (IPs), and IP Groups, on Bluemix Local System and PureApplication System, see the developerWorks articles at the following web address:

<http://ibm.biz/NetworkDesign>

Also, see *Integrating IBM PureApplication System into an Existing Data Center*, SG24-8285:

<http://www.redbooks.ibm.com/abstracts/sg248285.html>

Network VLAN Preparation

The following VLANs are needed, and are added to PureApplication Software by selecting **System** → **Network Configuration**:

- ▶ Blue Box Public Network
- ▶ OpenStack External Management Network
- ▶ OpenStack Internal Management Network
- ▶ OpenStack Heat Patterns External Management Network
- ▶ Customer data (the same as PureApplication data VLAN)

Figure 2-2 shows some of the VLANs listed.

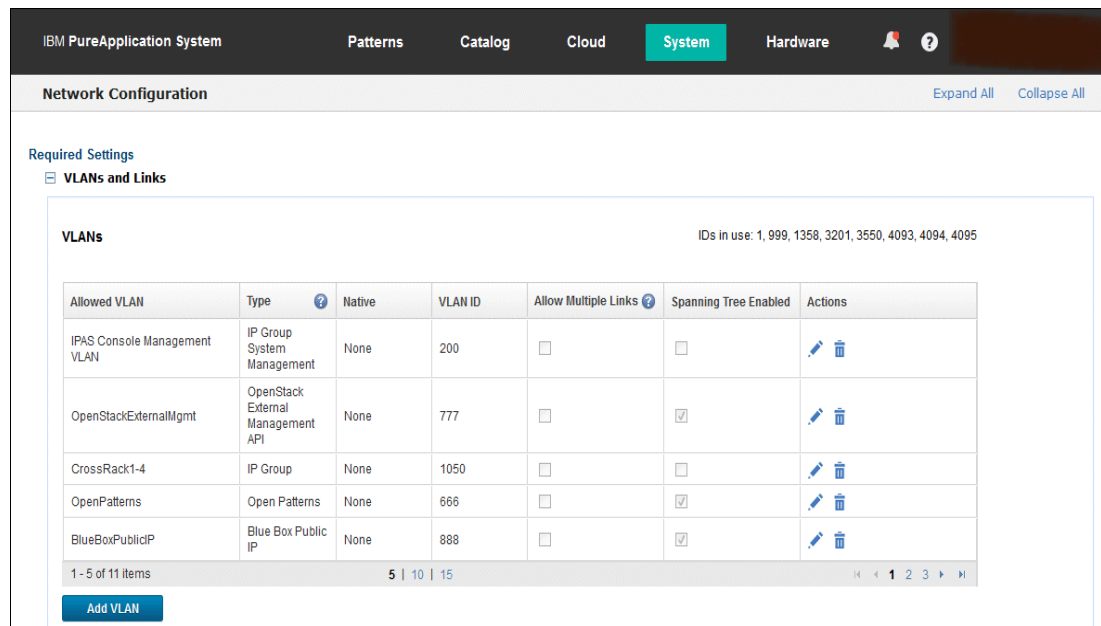


Figure 2-2 Network Configuration

Additional resources

The following additional resources are needed to complete the setup:

- ▶ A dedicated IP group
- ▶ Dedicated compute nodes
- ▶ Additional network information

Dedicated IP Group

One of the requirements for running OpenStack Heat patterns is to have a dedicated IP group, and then associate the data VLAN with this IP group and add at least 32 contiguous IPs to that IP group.

Compute Nodes

OpenStack requires three compute nodes. Make sure that the Bluemix Local System has at least three compute nodes available before configuring OpenStack.

Note: Currently, the requirement is to have exactly three compute nodes. This requirement will be relaxed in the future to accommodate more compute nodes in the OpenStack cloud group.

Network information

The VLANs listed in the beginning of, “Network VLAN Preparation” on page 26 should be configured with the network information that is related to your network environment. The specifics depend on your environment and network.

Table 2-1 on page 28 shows an example with the following configuration:

- ▶ The network team provides the VLAN IDs.
- ▶ The Gateway information depends on your network environment.
- ▶ The subnet mask remains as shown in Table 2-1 on page 28.

Table 2-1 shows sample VLAN network information.

Table 2-1 VLAN network information

VLAN	VLAN ID	Subnet MASK	Gateway
Blue Box Public Network	909	255.255.255.0	9.42.68.1
OpenStack External Management Network	2251	255.255.255.0	172.19.251.1
OpenStack Internal Management Network	999		
OpenStack Heat patterns External Management Network	2250	255.255.255.0	172.19.250.1
Customer data VLAN	629	255.255.248.0	172.17.56.1

2.1.2 Enabling OpenStack services

A user must be assigned the hardware administration role with permission to manage hardware resources (full permission) to perform the steps in the remainder of this section. After logging in to the Console, go to the **System** menu to complete the following tasks to enable OpenStack services:

1. Click **System** → **System Settings**.
2. To expand **OpenStack Services**, click the Plus sign (+).
3. Initially, the Services status should be **Disabled**, as seen in Figure 2-3 on page 29.
4. Click **Deploy** to initialize the OpenStack infrastructure.

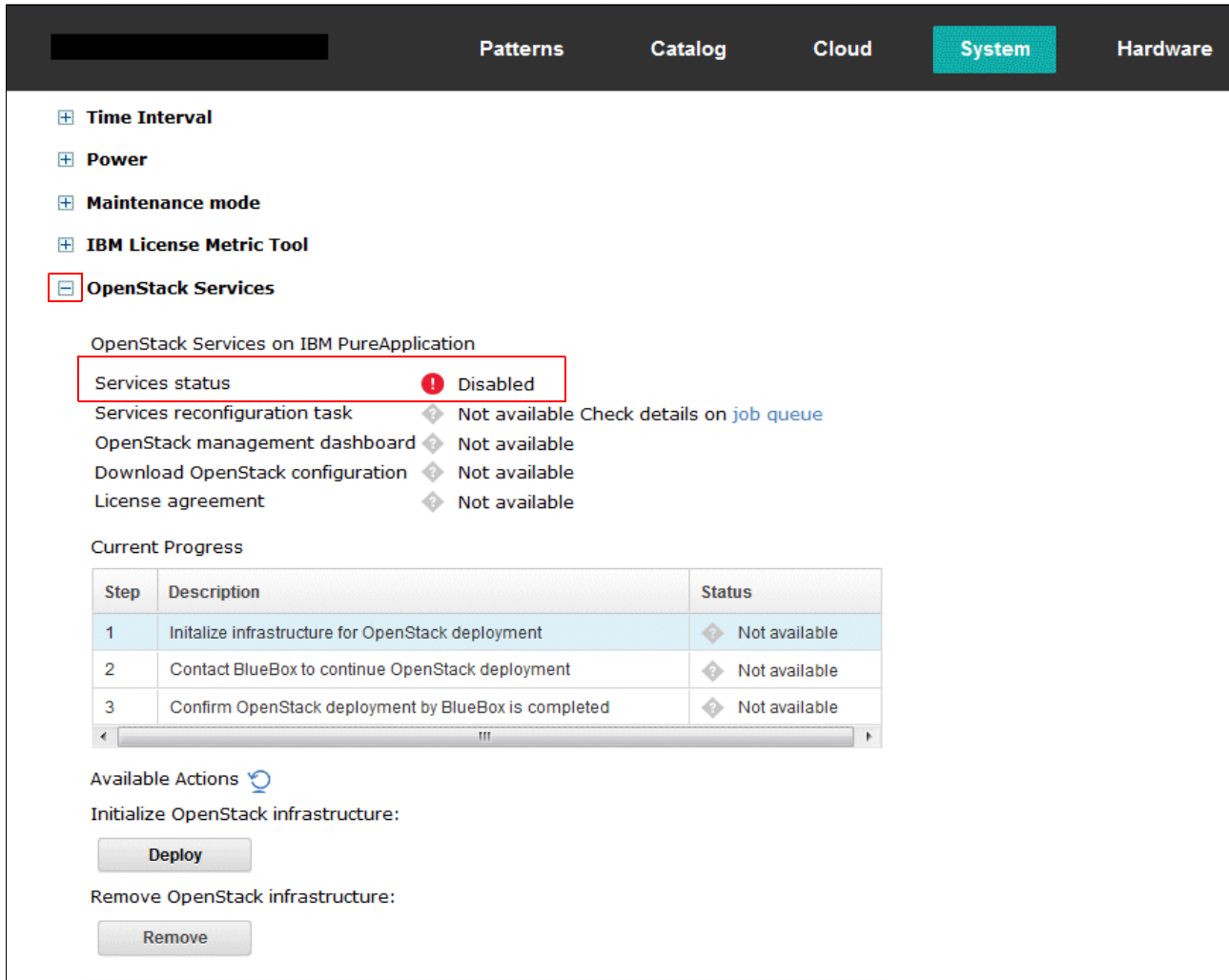


Figure 2-3 System menu showing OpenStack Services

5. Accept the license agreement for each of the images.
6. Enter the network information for each of the VLANs by using the information collected.
7. Select three available compute nodes from the list that is used for OpenStack Services.
8. Add the Data VLAN that is used for OpenStack Heat pattern deployment.
9. Enter the LDAP parameters. There are six entries, which are shown in Table 2-2, that need to be modified. The rest of the parameters should already have values.

Table 2-2 LDAP parameters

Parameter	Value
chase_referrals	false
tls_req_cert	never
page_size	10
user_pass_attribute	password
group_member_attribute	member
salt_master_access_group	salt

10. Click **OK** to start the OpenStack configuration task.

You can navigate to the System Job Queue to check whether the OpenStack job is completed. When the job has completed, you should see the OpenStack Service fully enabled, as seen in Figure 2-4. This screen is accessible from the **System** menu option (**System** → **System Settings** → **OpenStack Services**).

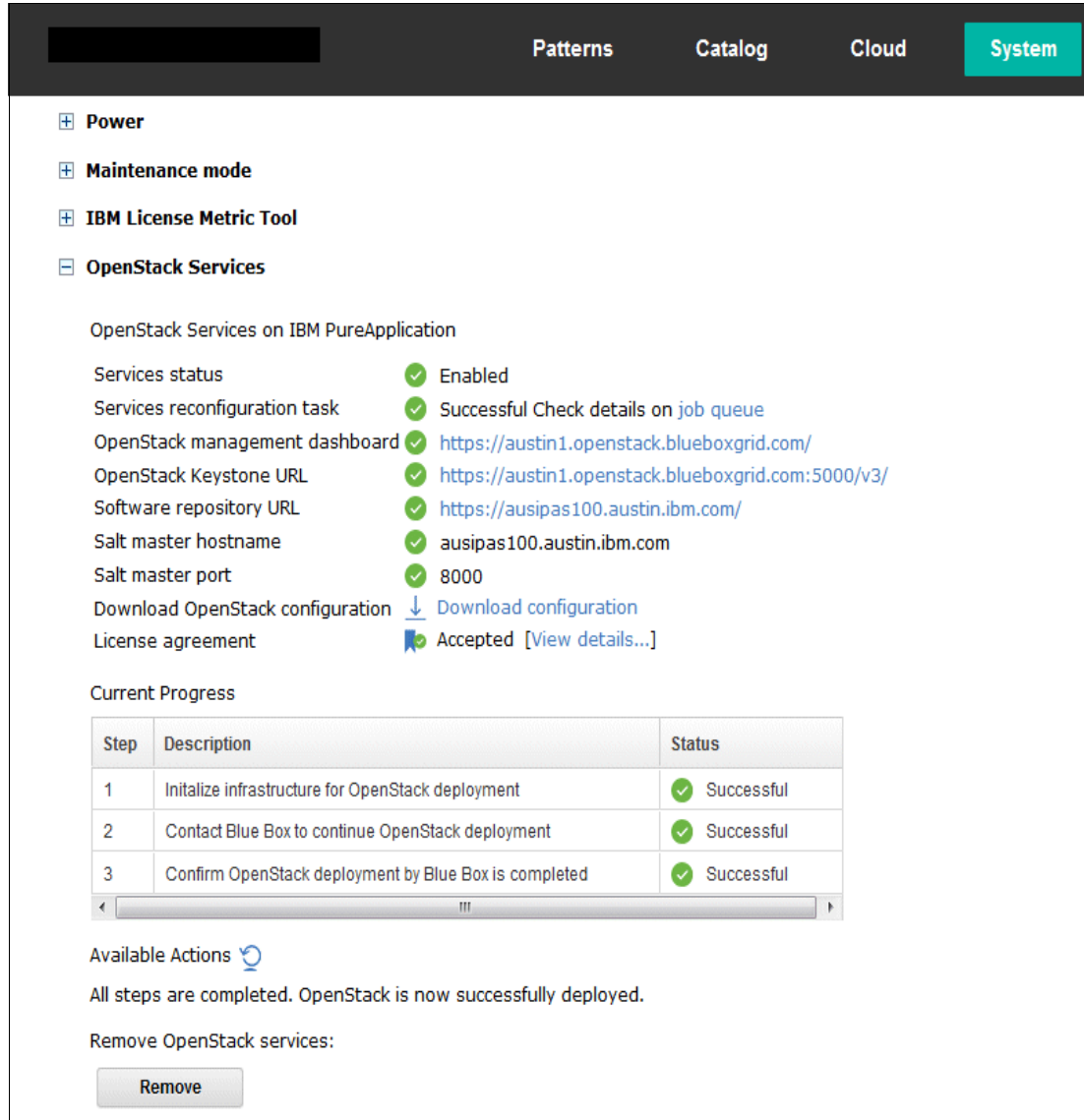


Figure 2-4 OpenStack services enabled

2.2 Configuring Blue Box

At this point, when the OpenStack services are in place, the environment is handed over to the IBM Blue Box team to set up Blue Box. That task is started by opening a support ticket from the management portal, which is called IBM Box Panel®. The ticket includes a JavaScript Object Notation (JSON) file with all of the details to deploy the environment.

The Box Panel login page can be accessed at the following web address:

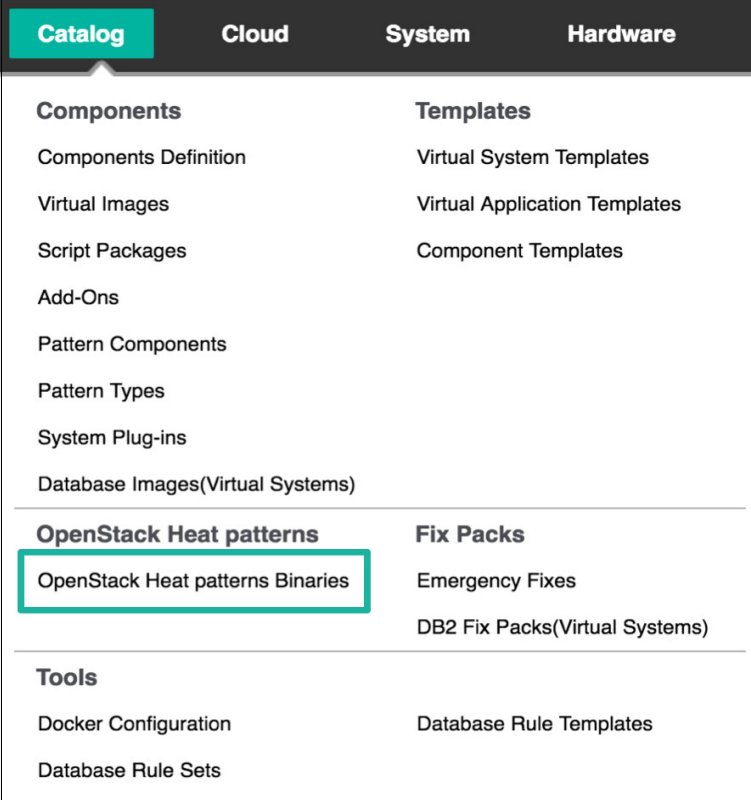
<https://boxpanel.bluebox.net>

2.2.1 Verifying OpenStack services

There are two menu items that can be viewed to verify that OpenStack services have been configured properly.

Catalog → OpenStack Heat Patterns

On the **Catalog** menu, under **OpenStack Heat patterns**, you should see **OpenStack Heat patterns Binaries**, as shown in the Figure 2-5.



Catalog	Cloud	System	Hardware
Components		Templates	
Components Definition		Virtual System Templates	
Virtual Images		Virtual Application Templates	
Script Packages		Component Templates	
Add-Ons			
Pattern Components			
Pattern Types			
System Plug-ins			
Database Images(Virtual Systems)			
OpenStack Heat patterns		Fix Packs	
OpenStack Heat patterns Binaries		Emergency Fixes	
		DB2 Fix Packs(Virtual Systems)	
Tools			
Docker Configuration		Database Rule Templates	
Database Rule Sets			

Figure 2-5 Catalog → OpenStack Heat patterns menu

Patterns → OpenStack Heat patterns

You should see **OpenStack Heat patterns** and **OpenStack Heat pattern Instances** under the **Patterns** menu, as shown in Figure 2-6.

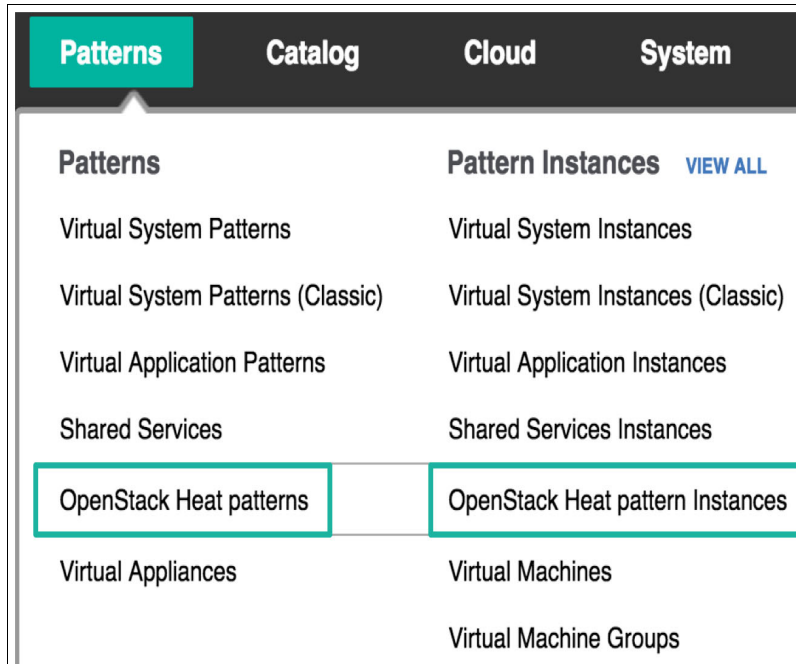


Figure 2-6 Patterns → OpenStack Heat patterns menu

If you click **OpenStack Heat patterns** you should see the sample OpenStack Heat patterns that are available, as shown in Figure 2-7. Deployment of sample OpenStack Heat patterns is described in Chapter 3, “Deploying sample OpenStack Heat patterns” on page 35.

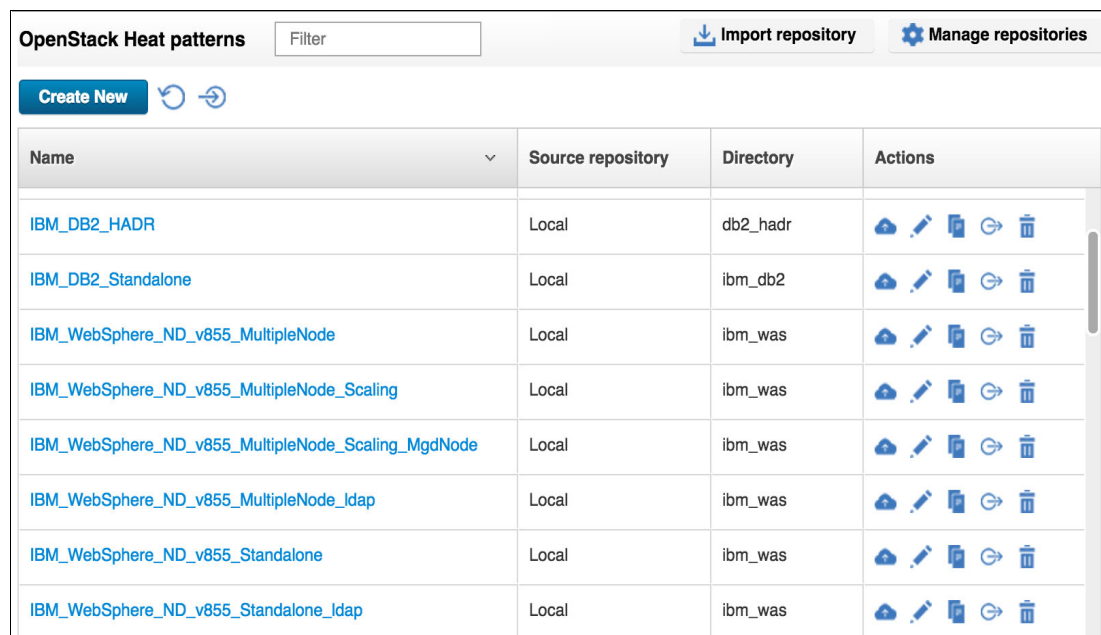


Figure 2-7 Available OpenStack Heat patterns on PureApplication Software

2.3 Sample OpenStack Heat patterns

As mentioned in Chapter 1, “Automated application delivery with OpenStack Heat patterns” on page 1, PureApplication Software V2.2.2 includes the sample OpenStack Heat patterns of IBM WebSphere Application Server and IBM DB2. After logging in to the console, go to the **Patterns** menu and select **OpenStack Heat patterns** to verify. If the sample patterns are not there, complete the following steps to import them:

1. Issue the following CLI command:

```
pure.cli/bin/pure -h <PureApp_IP> -u <USER_ID> -p <PWD> -a
deploy-er.openpatternbinaries.ImportBinaries("<openpattern_binary_folder>/rdsDB
2Binary.tar.gz", True)
deployer.openpatternbinaries.ImportBinaries("<openpattern_binary_folder>/rdsDB2
Binary.tar.gz", True)
```

2. Import the setup archive compressed file and the DB2, WebSphere Application Server, and Salt minion installation files. This is done from the **Patterns** → **OpenStack Heat patterns** menu.
3. Click **Import**, select `initial_import.zip`, and then click **OK**.
4. Click **Import**, select `ibm_db2_openpattern.zip`, and then click **OK**.
5. Click **Import**, select `ibm_websphere_openpattern.zip`, and then click **OK**.

Note: If you have PureApplication Software V2.2.1, the sample pattern binary files must be downloaded from the IBM support site and imported by using the command-line interface (CLI) command. The authors assume that readers are familiar with setting up the PureApplication CLI utility.

Deployment of the sample OpenStack Heat patterns is covered in 3.2, “Installing OpenStack Heat pattern samples” on page 36, the creation of new OpenStack Heat patterns is explained in Chapter 4, “Importing OpenStack Heat patterns” on page 47 and Chapter 5, “Creating an OpenStack Heat pattern” on page 53.

2.4 Using an alternative Git repository

Optionally you might choose to set up and use a Git repository for storing your OpenStack Heat patterns, alternative to the internal pattern repository available on PSM. This can be done using any of the following methods:

- ▶ Point to an existing remote Git repository.
- ▶ Stand up a new Git repository on PureApplication System.
- ▶ Stand up and point to a remote Git repository on another system.

In this IBM Redbooks publication, there are references to “standing up” a Git repository. This section describes how to set up one.

In 4.2, “Importing from a Git repository” on page 50, we describe how OpenStack Heat patterns stored in an alternative Git repository can be used.

2.4.1 Stand up a new Git repository on Bluemix Local System

To stand up a new Git repository, we need a Red Hat Linux VM:

1. Deploy a simple VM with IP tables disabled.
2. Next, use the Red Hat Satellite service or `yum` to install the software. Issue the following commands:

```
yum -y install git
yum -y install git-daemon
```

To start the Git Server, perform the following steps:

1. Create a folder, or a location, that will be the root of all repositories, for example `/git`.
2. Start the Git daemon as follows:

```
git daemon --reuseaddr --base-path=/git --export-all --verbose
--enable=receive-pack &
```

Because you disabled the firewall when you deployed your Red Hat instance, you do not need to open port 9418. If you did not stop the IP tables with the deployment, use the following commands to disable the firewall:

```
service iptables save
systemctl stop iptables.service
systemctl disable iptables.service
systemctl status iptables.service
```



Deploying sample OpenStack Heat patterns

IBM PureApplication Software v2.2.2 provides two sample OpenStack Heat pattern types: IBM WebSphere Application Server (WAS) and IBM DB2. This chapter will describe the deployment of these two sample patterns.

3.1 Verifying the installation of the sample patterns

To verify the sample pattern installations, complete the following steps:

1. Sign on to the IBM Bluemix Local System console, then go to **Patterns** → **OpenStack Heat patterns**, as shown in Figure 3-1.

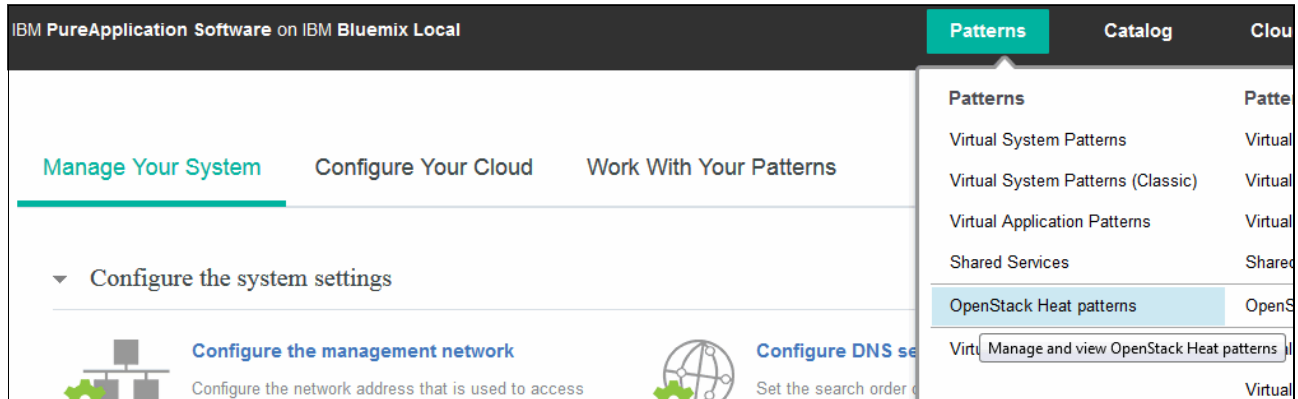


Figure 3-1 Opening the OpenStack Heat patterns

2. This action brings you to the OpenStack Heat pattern catalog, showing all of the sample patterns that are preloaded on Bluemix Local System using PureApplication Software, as shown in Figure 3-2.

The screenshot shows the 'OpenStack Heat patterns' catalog. It features a search filter, a 'Create New' button, and a table of patterns. The table has columns for Name, Source repository, Directory, and Actions. The following table represents the data shown in the screenshot:

Name	Source repository	Directory	Actions
ActiveDirectoryDomainController	Local	hotWindows/ActiveDirectoryController	[Cloud] [Edit] [Copy]
DHvanilla-RHEL_Link	Local		[Cloud] [Edit] [Copy]
IBM_DB2_HADR	Local	db2_hadr	[Cloud] [Edit] [Copy]
IBM_DB2_Standalone	Local	ibm_db2	[Cloud] [Edit] [Copy]
IBM_WebSphere_ND_v855_MultipleNode	Local	ibm_was	[Cloud] [Edit] [Copy]
IBM_WebSphere_ND_v855_MultipleNode_Scaling	Local	ibm_was	[Cloud] [Edit] [Copy]
IBM_WebSphere_ND_v855_MultipleNode_Scaling_f	Local	ibm_was	[Cloud] [Edit] [Copy]

Figure 3-2 Listing of OpenStack Heat patterns on Bluemix Local System

If the list is empty or it is not available, check with your Bluemix Local System administrator to see if the system has been “OpenStack enabled”.

3.2 Installing OpenStack Heat pattern samples

In this section, the installation of two sample OpenStack Heat patterns is described:

- ▶ WebSphere Application Server sample OpenStack Heat pattern
- ▶ DB2 sample OpenStack Heat pattern

3.2.1 Deploying the WebSphere Application Server OpenStack Heat pattern

Next, we deploy the `IBM_WebSphere_ND_v855_Standalone` OpenStack Heat pattern, which creates a stand-alone WebSphere Application Server Network Deployment cluster:

1. From the home page, select **Patterns** → **OpenStack Heat patterns**. You should be presented with a listing of all OpenStack Heat patterns.
2. Click **IBM_WebSphere_ND_v855_Standalone** to see the pattern information, as shown in Figure 3-3.

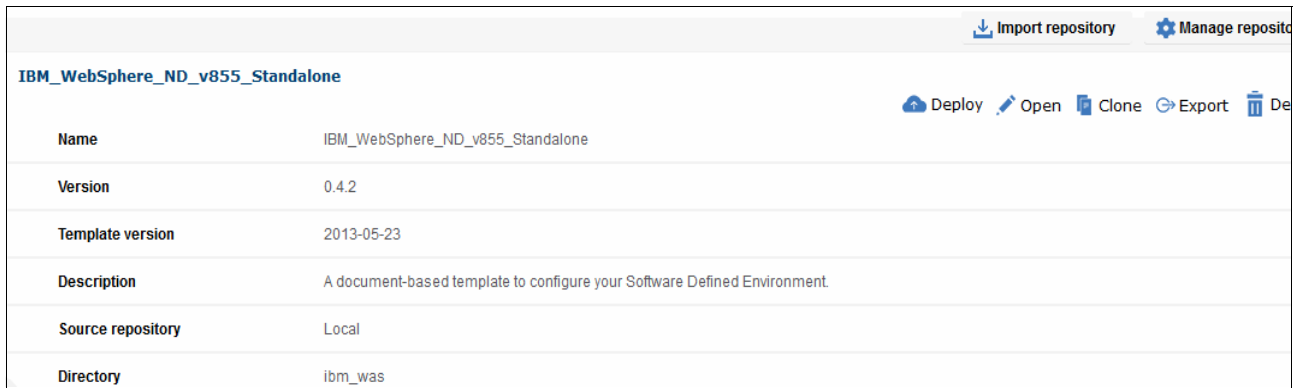


Figure 3-3 Select the `IBM_WebSphere_ND_v855_Standalone` OpenStack Heat pattern

3. When you have selected the pattern, you can select the **Deploy** button. This action opens the deployment page shown in Figure 3-4.

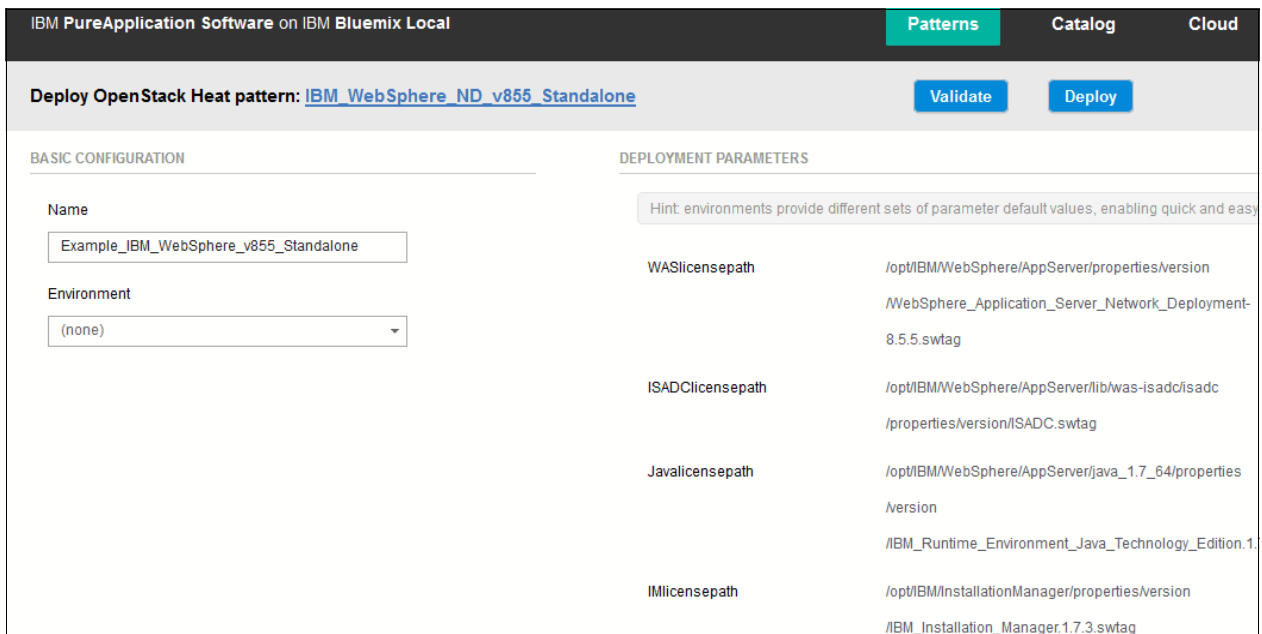


Figure 3-4 Deployment of WebSphere Application Server Standalone

The deployment page contains approximately forty deployment parameters, which need to be set prior to the deployment. These parameters can be saved in what is called an *Environment*, which is specific to a single OpenStack Heat pattern. Environments can be used to populate the required parameters for successive deployments of the same pattern.

If an Environment is available for the sample pattern, it can be used to pull in the values for this deployment. A description of all of the values needed for this sample pattern is shown in Table 3-1.

- Notice at the bottom of this list a **Save as a new environment** button is present. Click **Save as a new environment** if you want to save the parameter values for future use.

Table 3-1 Parameters for the WebSphere Application Server sample OpenStack Heat pattern

Host Name Parameters	Description
was_vm_name	Name of the virtual machine to deploy.
Image Parameters	
Flavor	OpenStack Nova concept to define the size of a virtual server. A full list of standard “flavors” can be found on the following website: http://docs.openstack.org/admin-guide/cli_manage_flavors.html
SSH Key name	A Secure Shell (SSH) key pair ^a to access the virtual machines’ operating system (OS) when deployed.
Availability	OpenStack Neutron concept, defined at the time Bluemix Local System is set up for OpenStack using IBM Blue Box. Additional availability zones can be defined in the OpenStack dashboard.
OS Image	OS Image to deploy onto.
Password (virtuser)	This is the password for the virtuser ^b user of the system.
Password (root)	This is the password for the root user of the system.
Yum Proxy URL	Uniform Resource Locator (URL) of the proxy to utilize. If no proxy is needed then leave blank.
Yum repository URL	URL to point to yum server.
Software Repository URL	Software repository URL.
Network	OpenStack Neutron concept, defined at the time Bluemix Local System is set up for OpenStack using Blue Box. Additional networks can be defined in the OpenStack dashboard.
Salt Parameters	
Saltmaster address	Host/Internet Protocol (IP) address of the Salt Master. It can be found in System → System settings → Openstack services .
Saltmaster API port	Salt Master port. It can be found in System → System Settings → OpenStack services .
Salt API user	The user connecting to Salt application programming interface (API).
Salt API password	Password for connection to Salt API.
WebSphere Parameters	
WAS OS user	OS Username for the WebSphere operation system user.
WAS OS group	OS Group for the WebSphere operation system user.

Host Name Parameters	Description
WebSphere Parameters (cont.)	
WAS OS Users password	OS Password for the WebSphere operation system.
WAS Console user	Admin user login to WebSphere Application Server.
WAS Console password	Admin password login to WebSphere Application Server.
WAS Install directory	Install directory of WebSphere Application Server.
WAS Profile directory	Profile directory of the WebSphere Application Server node.
Enable custom log location	Select enable for alternate directory for server logs.
Custom log directory	To specify an alternate directory for server logs.
Enable java	Select whether to enable Java 7.
WAS fixpack version	Was fixpack version, 2 digits only.
Java fixpack version	Java fixpack version.
Enable 32-bit SDK ^c	Software development kit (SDK) feature, can select 32-bit or 64-bit option.
Enable 64-bit SDK	SDK feature, can select 32-bit or 64-bit option.
WAS Standalone profile name	Standalone profile name.
WAS Standalone cell name	Standalone cell name.
WAS Standalone node name	Node name.
WAS Standalone server name	Server name.
IBM Installation Manager	
IBM Installation Mgr Install Mode	Instructs how Installation manager should be installed (admin, user, or group). Normally admin.
IBM Installation Mgr Install User	The OS user selected to install Installation Manager.
IBM Installation Mgr Install Group	The OS group selected to install Installation Manager.
IBM Installation Mgr Install directory	Install directory of Installation Manager.
IBM Installation Mgr Shared Location	Shared directory of Installation Manager.
IBM Installation Mgr Repository Path	Installation Manager repository path.
Zeron Parameters	
openstack_keystone_url	OpenStack Keystone URL can be found in System → System settings → OpenStack Services and is usually in the following location: <code>https://<openstack_external_hostname>:5000/v3/</code>

- Key pairs need to be stored in the OpenStack runtime. To do that, from the OpenStack dashboard, go to **Project** → **Compute** → **Access & Security** → **Key Pairs**. Here key pairs can be imported or generated. The **Key Pair Name** value is the one to be used for this parameter.
- The virtuser ID is a generic user ID that is created for you with the deployment. You can only set the password.
- An SDK is a set of software development tools that allows the creation of applications for a certain software package, software framework, hardware platform, computer system, video game console, operating system, or similar development platform.

3.2.2 Deploying the DB2 Server OpenStack Heat pattern

Deploy the IBM_DB2_Standalone OpenStack Heat pattern, which stands up a stand-alone DB2 server:

1. From the home page, click **Patterns** → **OpenStack Heat patterns**. You are presented with a listing of all OpenStack Heat patterns.
2. Click **IBM_DB2_Standalone** to see the pattern information, as shown in Figure 3-5.

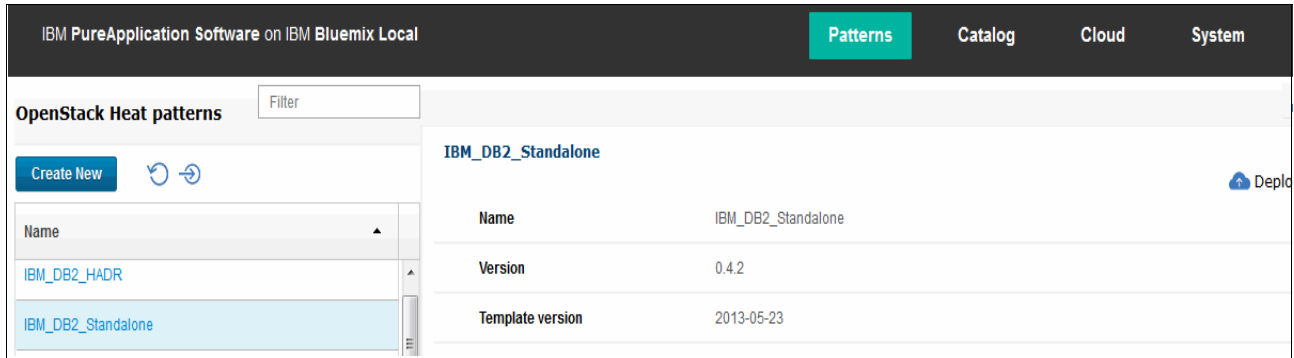


Figure 3-5 Select the IBM_DB2_Standalone OpenStack Heat pattern

3. Click **Deploy** to open the deployment panel, with its parameter options, as displayed in Figure 3-6.

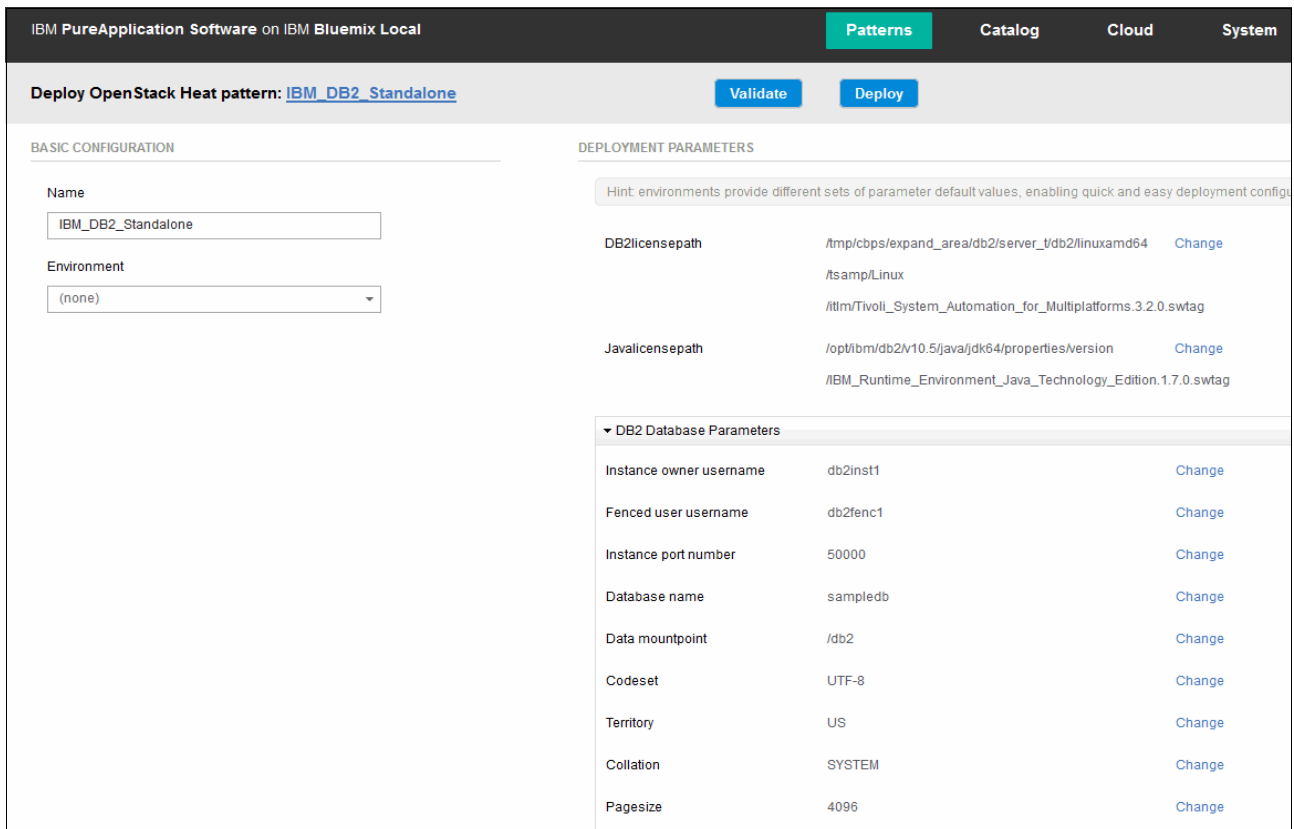


Figure 3-6 Deployment parameters for DB2 OpenStack Heat pattern

A description of all of the values need for this sample pattern is shown in Table 3-2.

Table 3-2 Parameters for the DB2 sample OpenStack Heat pattern

DB2 Database Parameters	Description
Instance owner username	Instance owner username.
Fenced user username	Fenced user username.
Instance port number	Instance port.
Database name	Name of the database to be created.
Data mountpoint	Mountpoint for the database within the instances.
Codeset	Specifies the code set to be used for data entered into this database. After you create the database, you cannot change the specified code set.
Territory	Specifies the territory identifier or locale identifier.
Collation	The type of collating sequence to be used for the data.
Pagesize	Page size of the default buffer pool along with the initial table spaces (SYSCATSPACE, TEMPSPACE1, USERSPACE1) when the database is created.
Host Name Parameters	
Server hostname	DB2 Server hostname.
Image Parameters	
Flavor	OpenStack Nova concept defines the size of a virtual server. The full list can be found at the following web address: http://docs.openstack.org/admin-guide/cli_manage_flavors.html
SSH Key name	An SSH key pair ^a to access the virtual machines' operating system when deployed.
Availability zone	OpenStack Neutron concept, defined at the time Bluemix Local System is setup for OpenStack using Blue Box. Additional availability zones can be defined in the OpenStack dashboard.
OS Image	OS Image to deploy onto.
Password (virtuser)	This is the password for the virtuser user of the system.
Password (root)	This is the password for the root user of the system.
Yum Proxy URL	URL of Proxy to utilize, if no proxy is needed then leave blank.
Yum repository url	URL to point to yum server.
Software Repository URL	Software repository URL.

Network Parameters	
Network	OpenStack Neutron concept, defined at the time Bluemix Local System is set up for OpenStack using Blue Box. Additional networks can be defined in the OpenStack dashboard.
Salt Deployments	
Saltmaster address	Hostname/IP address of the Salt Master. It can be found in System → System Settings → OpenStack services .
Saltmaster API port	Salt Master port. It can be found in System → System Settings → OpenStack services .
Salt API user	The user connecting to Salt API.
Salt API password	Password for connection to Salt API.
Zeron Parameter	
openstack_keystone_url	OpenStack Keystone URL can be found in System → System Settings → OpenStack Services , usually in the following location: <code>https://<openstack_external_hostname>:5000/v3/</code>

- a. Key pairs need to be stored in the OpenStack runtime. from the OpenStack dashboard, select **Project** → **Compute** → **Access & Security** → **Key Pairs**. Here key pairs can be imported or generated. The **Key Pair Name** value is the one to be used for this parameter.

3.3 Managing the OpenStack Heat pattern deployments

In order to manage the OpenStack Heat pattern deployments, complete the following steps:

1. Go to **Patterns** → **OpenStack Heat pattern Instances**, as shown in Figure 3-7.

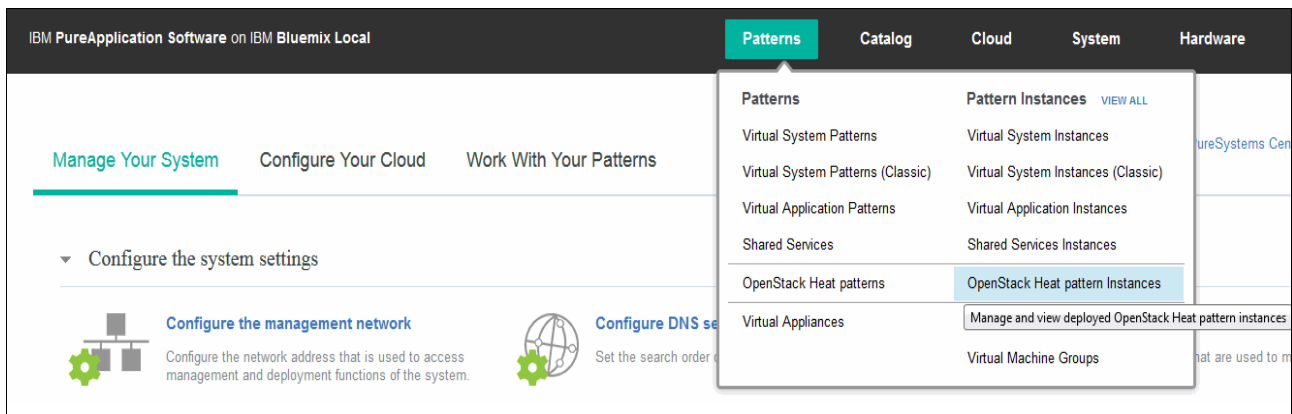
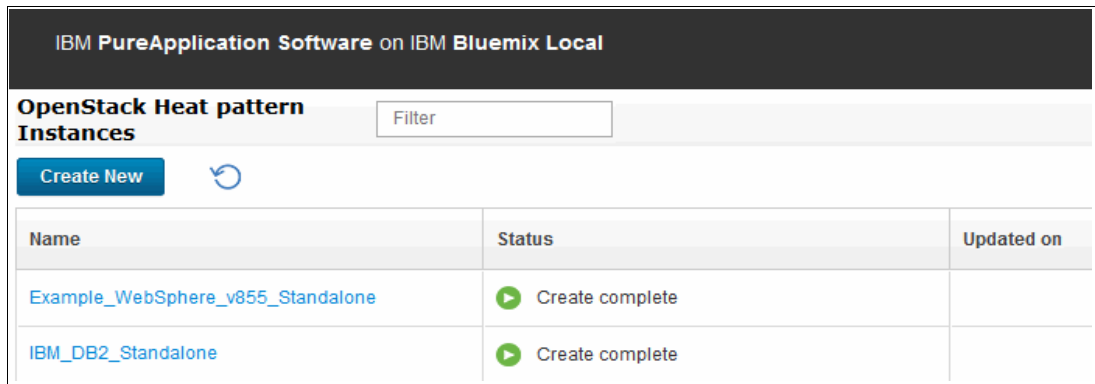


Figure 3-7 Go to OpenStack Heat pattern Instances in Order to Manage

A list of OpenStack Heat pattern deployments will be displayed, as shown in Figure 3-8.



IBM PureApplication Software on IBM Bluemix Local

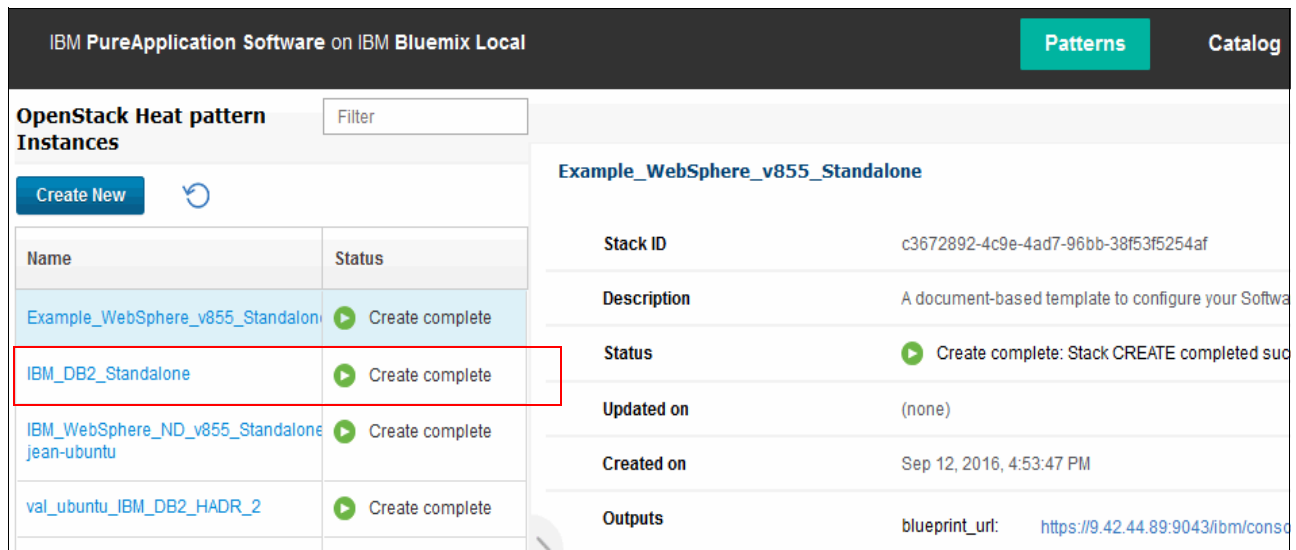
OpenStack Heat pattern Instances

Create New

Name	Status	Updated on
Example_WebSphere_v855_Standalone	Create complete	
IBM_DB2_Standalone	Create complete	

Figure 3-8 Instances that are deployed

2. To select the deployment that you want to manage, click it, as shown in Figure 3-9.



IBM PureApplication Software on IBM Bluemix Local Patterns Catalog

OpenStack Heat pattern Instances

Create New

Name	Status
Example_WebSphere_v855_Standalone	Create complete
IBM_DB2_Standalone	Create complete
IBM_WebSphere_ND_v855_Standalone_jean-ubuntu	Create complete
val_ubuntu_IBM_DB2_HADR_2	Create complete

Example_WebSphere_v855_Standalone

Stack ID c3672892-4c9e-4ad7-96bb-38f53f5254af

Description A document-based template to configure your Software

Status Create complete: Stack CREATE completed suc

Updated on (none)

Created on Sep 12, 2016, 4:53:47 PM

Outputs blueprint_url: <https://9.42.44.89:9043/ibm/cons>

Figure 3-9 Select the standalone WebSphere Application Server deployment

The important information is the Status and the Outputs. The *green* arrow on the **Status** line means a successful deployment. A *red* arrow would mean there was an issue with the deployment.

3. On the **Outputs** line notice the blue link to the WebSphere Application Server console. Click the link to go to the WebSphere Application Server console, shown in Figure 3-10.

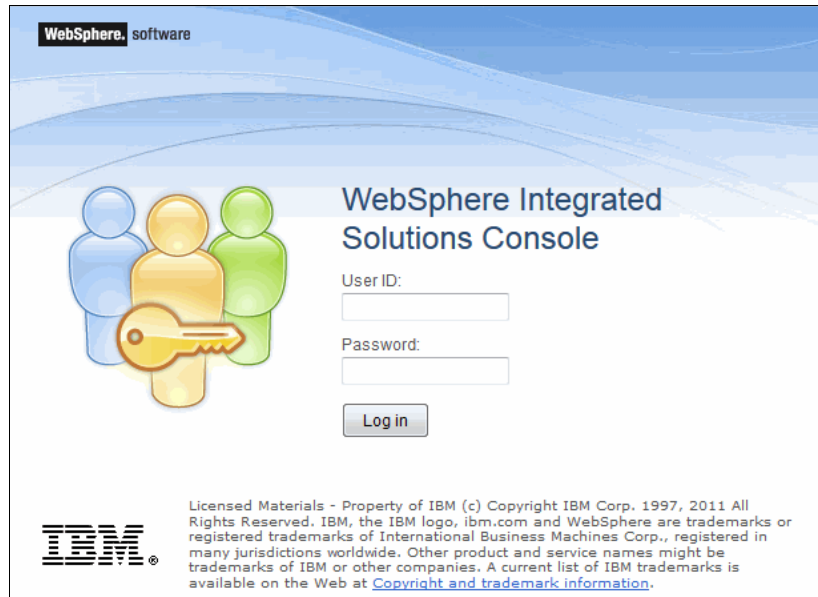


Figure 3-10 WebSphere Application Server console

4. If you click the **Manage** button as shown in Figure 3-7 on page 42, it is also possible to access the Zeron user interface (UI) for this OpenStack Heat pattern instance. More information about Zeron and its UI are provided in Chapter 7, "Zeron for OpenStack Heat patterns" on page 75.

5. An alternative view of the deployed OpenStack Heat pattern instance is also available from the OpenStack dashboard (**Project** → **Orchestration** → **Stacks**), as shown in Figure 3-11.

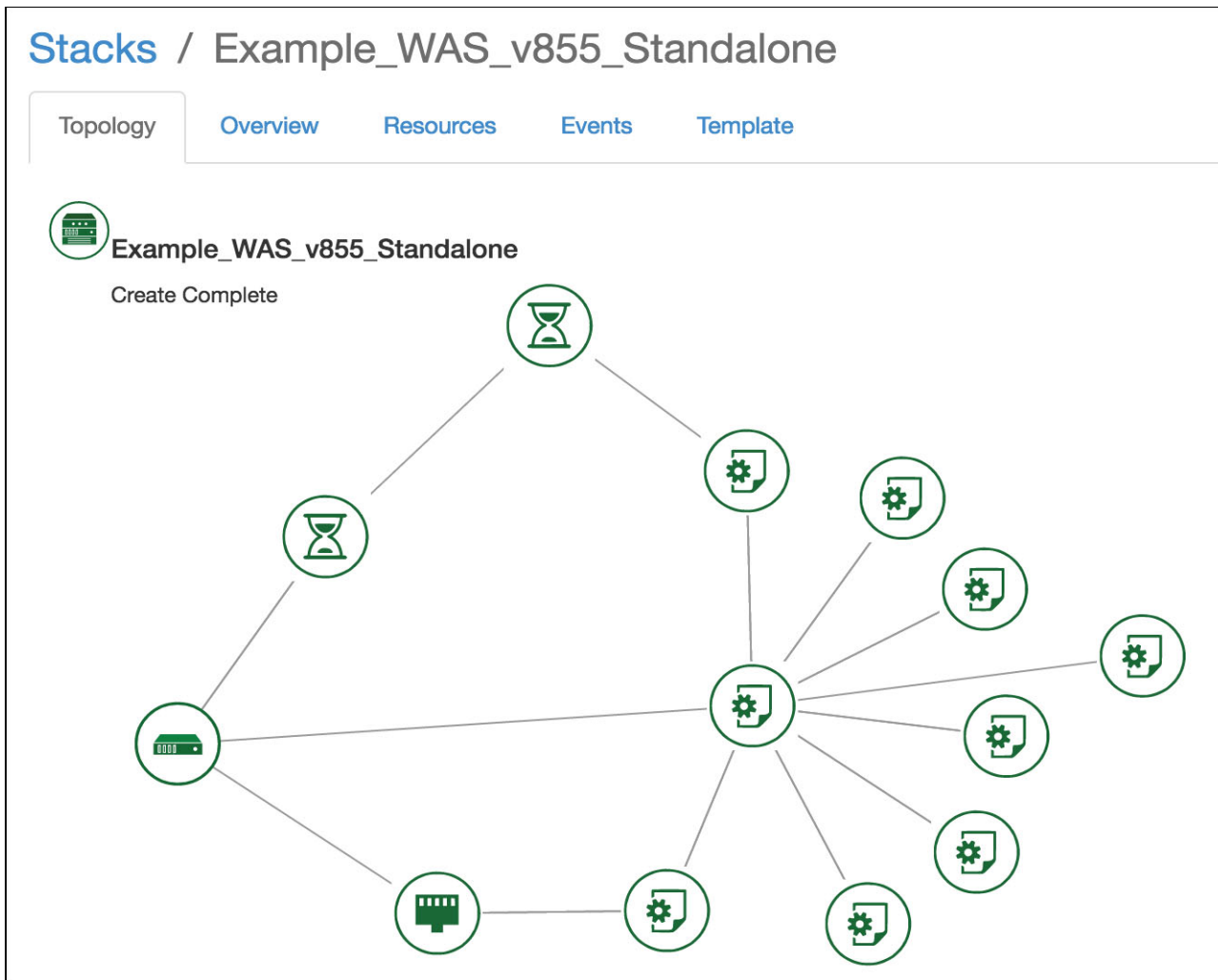


Figure 3-11 WebSphere Application Server Deployment from the OpenStack dashboard



Importing OpenStack Heat patterns

In this chapter, we look at importing OpenStack Heat patterns into IBM PureApplication Software on IBM Bluemix Local System. You can write your own OpenStack Heat patterns or start from an existing Heat Orchestration Template (HOT).

4.1 Importing a simple Heat Orchestration Template

To import a HOT, the YAML file that describes the template must first be obtained. For this example, perform the following steps:

1. Go to the OpenStack Community App Catalog at the following web address:
<http://apps.openstack.org>
2. Download the sample “Hello World” HOT example.
3. From the link (in step 1), select **HEAT TEMPLATES** to display the catalog.
4. Select the Hello World pattern and download it.
5. With the file downloaded, look inside this simple `hello_world.yaml` file, shown in Example 4-1.

Example 4-1 Hello_world.yaml

```
#
# This is a hello world HOT template just defining a single compute
# server.
#
heat_template_version: 2013-05-23
description: >
  Hello world HOT template that just defines a single server.
  Contains just base features to verify base HOT support.
```

The `hello_world.yaml` is a simple text file in YAML format. It starts off with some comments (the ones with a Number sign (#) in front of them) and includes the heat template version. Next is a short description.

For more information about the file format, see the following website:

<http://www.yaml.org/start.html>

Now import this YAML file into IBM PureApplication Software and perform the following steps:

1. Access the OpenStack Heat pattern catalog, as shown in Figure 4-1.

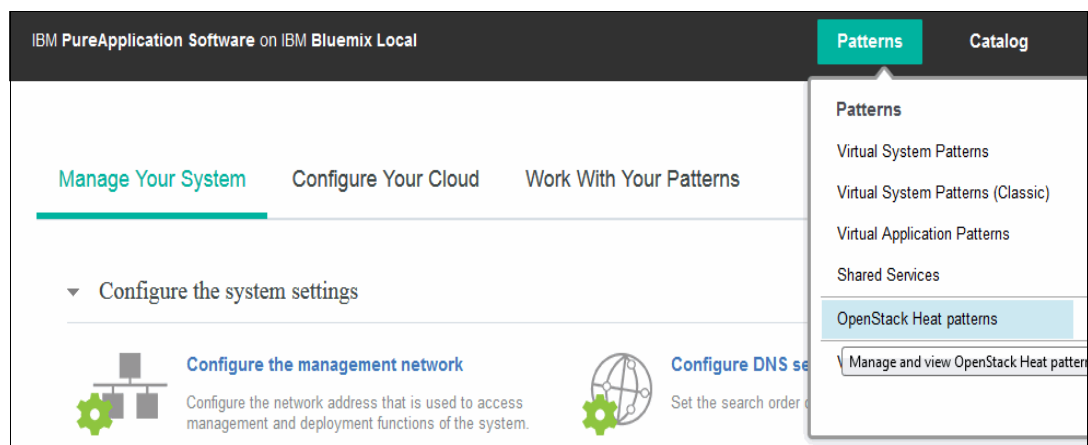


Figure 4-1 Access OpenStack Heat pattern catalog

2. Select the OpenStack Heat patterns, as shown in Figure 4-2.

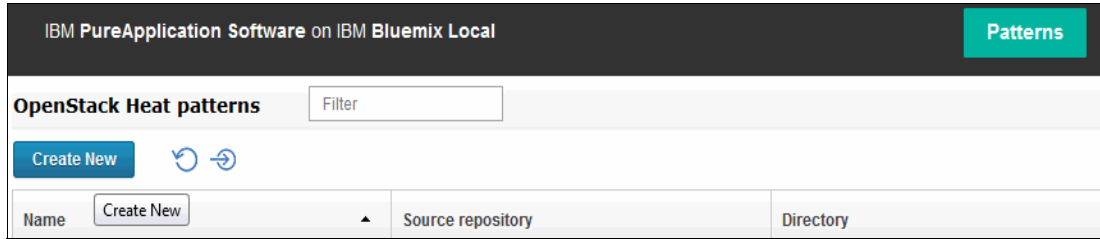


Figure 4-2 OpenStack Heat patterns page with focus on the Import button

3. After you select **Import**, a new dialog box displays, as shown in Figure 4-3. Browse to the file location of the YAML file and import the file. Select **OK**.

Note: IBM PureApplication Software accepts files in YAML format only with a `.yaml` or a `.yml` extension.

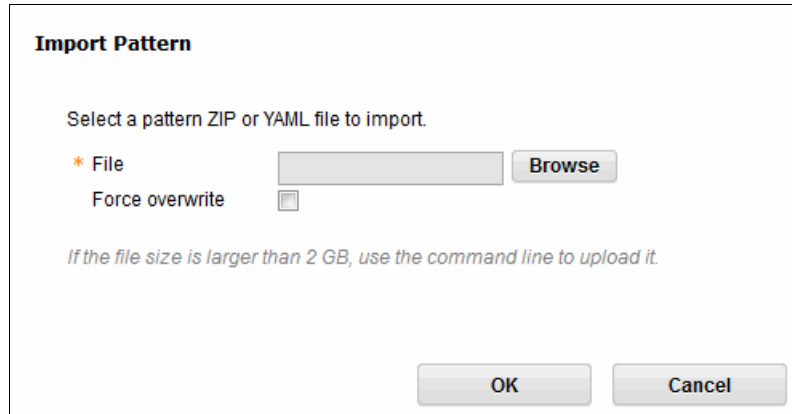


Figure 4-3 Import Pattern Screen

Importing the OpenStack Heat pattern adds and converts the HOT into a basic OpenStack Heat pattern, and then makes the pattern available in the catalog, as shown in Figure 4-4.

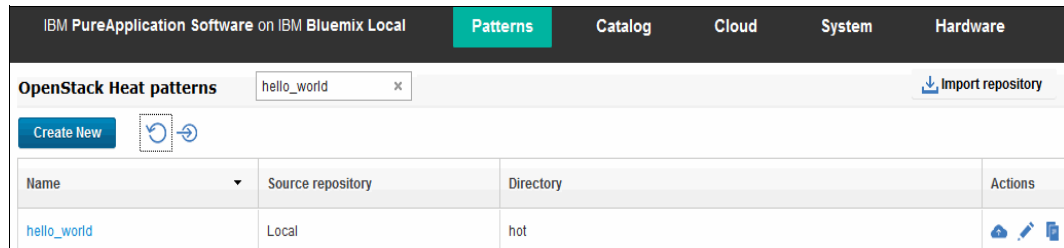


Figure 4-4 Imported OpenStack Heat pattern in system

Now you can use PureApplication Software to edit, enrich, deploy, and manage the newly created OpenStack Heat pattern.

4.2 Importing from a Git repository

Another option for importing OpenStack Heat patterns into PureApplication Software is directly from a Git repository. To do this, you must first set up a link to the repository from PureApplication Software:

1. Go to the OpenStack Heat patterns page, as shown in Figure 4-5.

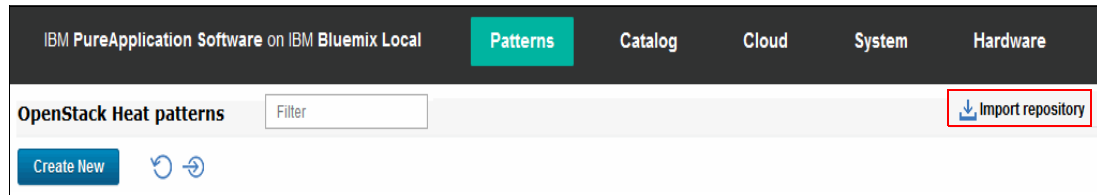


Figure 4-5 OpenStack Heat pattern Page with Import Repository

2. Click **Import Repository** to open the dialog that is shown in Figure 4-6.

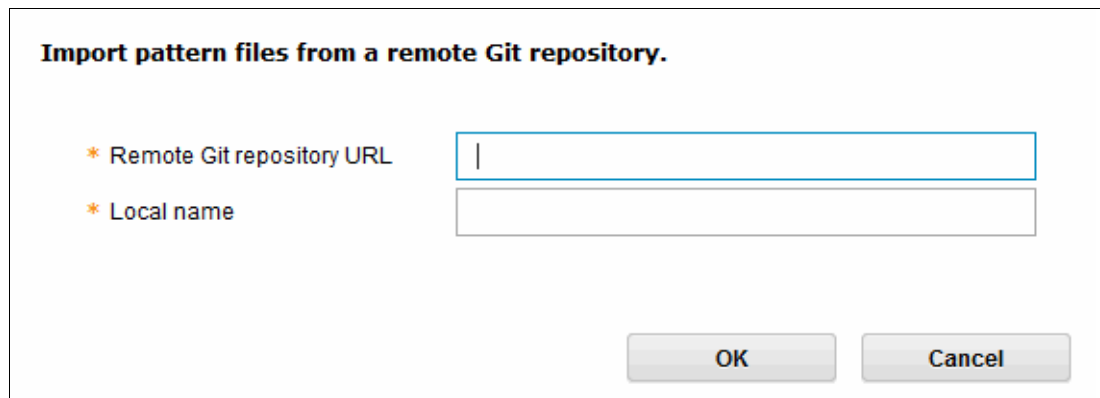


Figure 4-6 Import Repository from Git Repository

3. Complete the **Remote Git repository URL** and a **Local name** that you want to give to the repository. These two names are used to import the OpenStack Heat patterns that are contained in the Git repository into the OpenStack Heat pattern catalog. Click **OK**.

Note: All patterns that are found in the Git repository are brought into PureApplication Software at the same time.

4. Using the **Manage repository** button shown in Figure 4-7, you can at any point reload the repository, so that the most up-to-date patterns are made available in the OpenStack Heat pattern catalog.

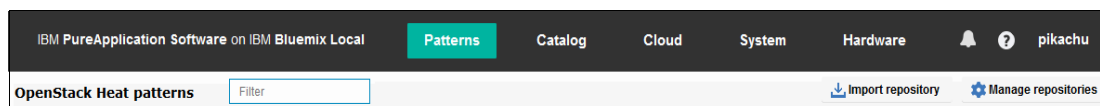


Figure 4-7 Manage Git Repositories

5. When a repository is no longer needed, you can delete it. These two actions (update and delete) are available for each linked Git repository, as shown in Figure 4-8.

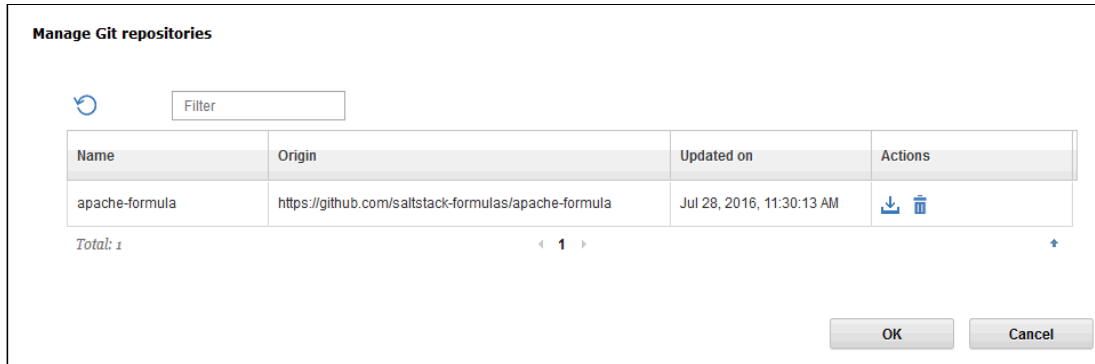


Figure 4-8 Manage Git repositories

Note: When a new OpenStack Heat pattern is added to the linked repository, it does not get directly imported in the OpenStack Heat pattern catalog. The **reload** action must be performed for a new OpenStack Heat pattern to get imported into the OpenStack Heat pattern catalog.



Creating an OpenStack Heat pattern

In Chapter 4, “Importing OpenStack Heat patterns” on page 47 we described the following topics:

- ▶ How to work with the OpenStack Heat pattern samples that are provided in IBM PureApplication Software on IBM Bluemix Local System
- ▶ How to import an OpenStack Heat orchestration template (HOT), and use it as the base of an OpenStack Heat pattern

In this chapter, we describe how to create a new OpenStack Heat pattern.

The sample we use is a simple Ubuntu image with an Apache web server installed on it.

There are several fundamental steps that developers need to follow to create an OpenStack Heat pattern on PureApplication Software:

1. Create a *blueprint* in the IBM UrbanCode Deploy (UCD) Blueprint Designer.
2. Define the OpenStack Heat pattern in UCD Blueprint Designer.
3. Add SaltStack formulas to the OpenStack Heat pattern (optional).
4. Commit and push the OpenStack Heat pattern to Git.

After the pattern is created, it can be deployed, accessed, managed, and monitored by completing steps similar to those described in the previous chapters of this IBM Redbooks publication.

This chapter describes the previous for a sample OpenStack Heat pattern, as well as taking the reader through the deployment and verification of the sample pattern.

5.1 Creating a blueprint in UCD Blueprint Designer

Complete the following steps to create a blueprint in UCD Blueprint Designer:

1. Go to **Patterns** → **OpenStack Heat patterns**, as shown in Figure 5-1.

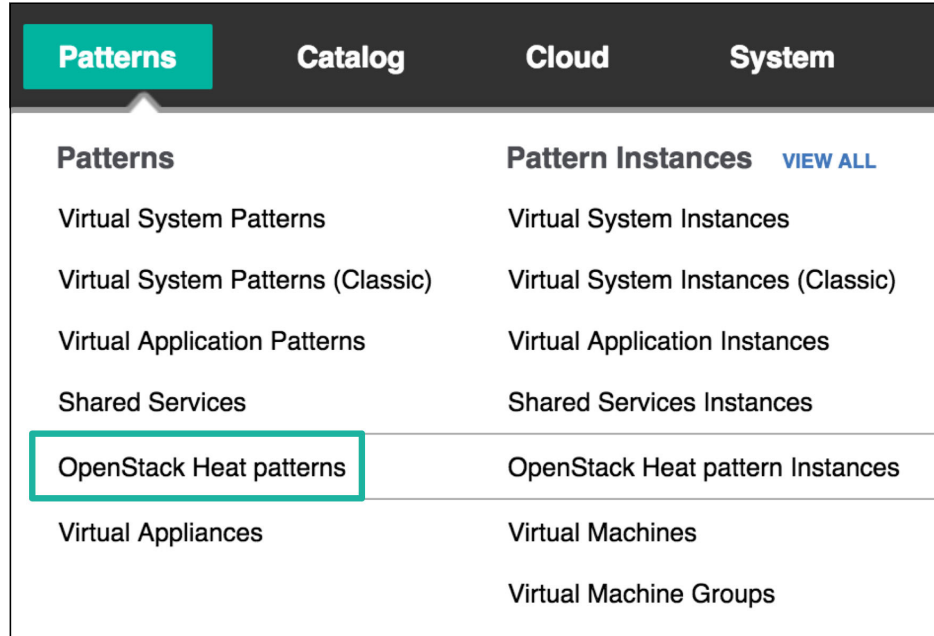


Figure 5-1 Access OpenStack Heat patterns

This page shows the OpenStack Heat patterns that are contained in the Git repository part of the PureApplication Software *OpenStack Heat pattern engine*. An example is shown in Figure 5-2.

The screenshot shows the 'OpenStack Heat patterns' catalog. It includes a 'Filter' input field, a 'Create New' button, and a table of patterns.

Name	Source repository	Directory	Actions
IBM_DB2_HADR	Local	db2_hadr	
IBM_DB2_Standalone	Local	ibm_db2	
IBM_WebSphere_ND_v855_MultipleNode	Local	ibm_was	
IBM_WebSphere_ND_v855_MultipleNode_Scaling	Local	ibm_was	

Figure 5-2 OpenStack Heat pattern catalog

2. To create a new OpenStack Heat pattern, click **Create New** on the page shown in Figure 5-2 on page 54.

This opens the UrbanCode Deploy (UCD) Blueprint Designer, as shown in Figure 5-3, in a separate browser tab. The *OpenStack Heat pattern engine* implementation in PureApplication Software uses UCD Blueprint Designer for pattern creation and editing. In UCD Blueprint Designer, patterns are called *blueprints*.

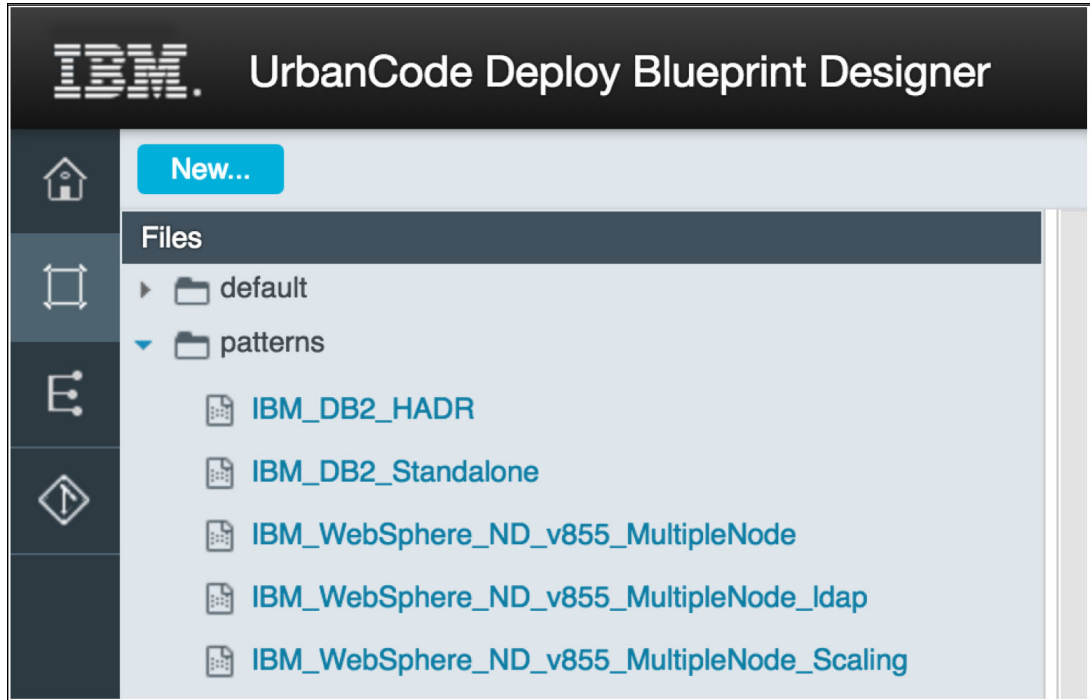
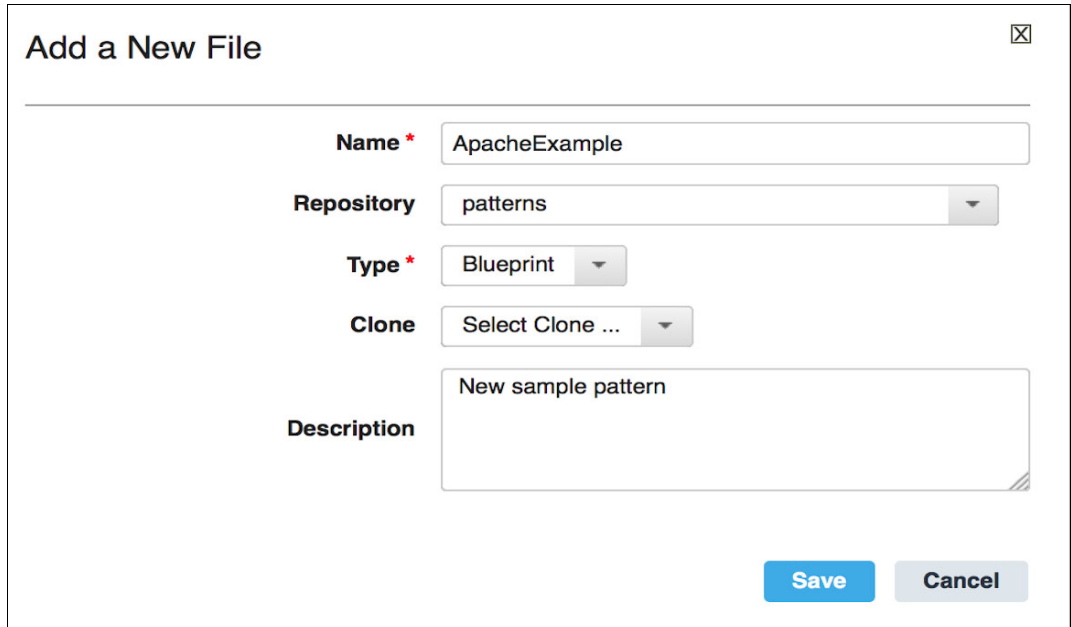


Figure 5-3 Blueprints in UrbanCode Deploy Blueprint Designer

Note: It is important to note that UCD Blueprint Designer is an *offline* authoring environment. Therefore, all changes that are made to patterns are only saved to a local Git repository. Changes need to be manually pushed to the PureApplication Software pattern repository, still based on Git, in order for the new patterns, or new versions of the patterns, to be available in the OpenStack Heat pattern catalog. For more information, see 5.4, “Committing and pushing the OpenStack Heat pattern to Git” on page 60 for details.

3. From the UCD Blueprint Designer, click **New** to create a new *blueprint*. This opens the wizard that is shown in Figure 5-4.



The screenshot shows a dialog box titled "Add a New File" with a close button in the top right corner. The form contains the following fields:

- Name ***: ApacheExample
- Repository**: patterns
- Type ***: Blueprint
- Clone**: Select Clone ...
- Description**: New sample pattern

At the bottom right, there are two buttons: "Save" (blue) and "Cancel" (grey).

Figure 5-4 UCD Blueprint Designer: New blueprint wizard

4. Complete the parameters, as shown in Figure 5-4. Remember to select the correct **Repository** where the OpenStack Heat pattern template will be saved. This is called *patterns* for the internal UCD Blueprint Designer Git repository. **Type** should be *Blueprint*, and no **Clone** needs to be selected, because we are starting from an empty canvas.
5. Clicking **Save** opens the pattern editor page. This page has two tabs:
 - **Diagram**, where a pattern can be created through a graphical user interface (GUI)
 - **Source**, where the Heat template source statements are automatically generated based on the Diagram, and can be manually edited

In this example, we use the Diagram view to create a pattern by using the available artifacts in the palette on the right side of the Blueprint Designer, which is shown in Figure 5-5. The editing experience is based on dragging components to the canvas, and wiring them together, which is similar to the editing experience for PureApplication *Patterns of Expertise*.

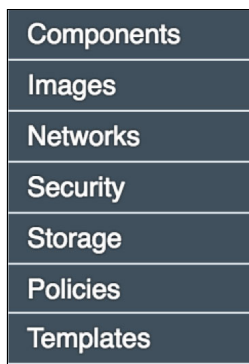


Figure 5-5 UCD Blueprint Designer: Artifacts palette

The palette in Figure 5-5 lists seven categories, which are explained in the following list:

- ▶ *Components* are the SaltStack formulas that can be used within an OpenStack Heat pattern.
- ▶ *Images* are the available operating system (OS) images, such as Red Hat 7.2 and Ubuntu 14.04. OpenStack Glance underpins this category.
- ▶ *Networks* define network segments within the pattern. They are useful when assigning IPs to the OpenStack Heat pattern components. OpenStack Neutron underpins this category.
- ▶ *Security* enables you to define and associate security groups with OpenStack Heat patterns and their components. OpenStack Keystone underpins this category.
- ▶ *Storage* contains storage volumes, which can be associated with patterns. OpenStack Cinder¹ underpins this category.
- ▶ *Policies* enable you to create and apply policies, such as auto scaling and load balancing. Policies are underpinned by different OpenStack projects, such as Heat, Ceilometer, and Neutron.
- ▶ *Templates* are template instances for Heat resource types, if already available. These templates are typically not used when designing OpenStack Heat patterns. OpenStack Heat underpins this category.

5.2 Defining a new OpenStack Heat pattern

To define the sample OpenStack Heat pattern, complete the following steps:

1. Go to the **Diagram** tab for the pattern that was created through the wizard shown in Figure 5-4 on page 56.

Tip: At any point during this process, you can switch to the **Source** tab to see the HOT getting populated with the components that get added to the diagram.

2. Drag an Ubuntu image on to the canvas from the Images category, as shown in Figure 5-6.

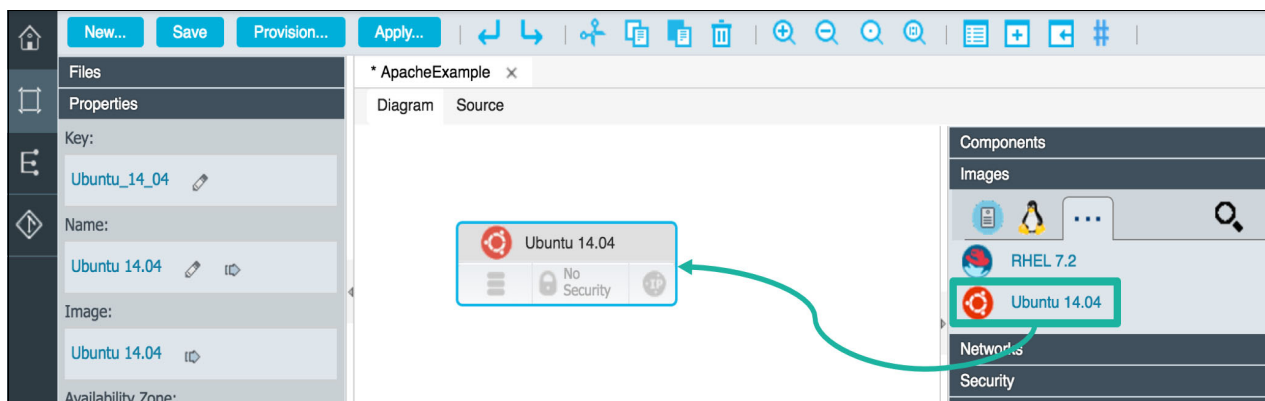


Figure 5-6 Add Ubuntu image to the sample OpenStack Heat pattern

¹ At the time of writing of this paper, OpenStack Cinder is not supported on Bluemix Local System. Therefore, adding storage volumes to OpenStack Heat patterns is not a viable operation.

3. Add a network to the canvas from the Networks category. In this example, a network called *internal* is already defined and available in the **Networks** tab.

Note: Networks and availability zones are defined at the time that Bluemix Local System is set up for OpenStack by using IBM Blue Box. Additional networks can be defined through the OpenStack dashboard, which you can reach from the Console by selecting **System** → **System Settings** → **OpenStack Services** → **OpenStack management dashboard**.

4. Connect the network to the image, as shown in Figure 5-7.

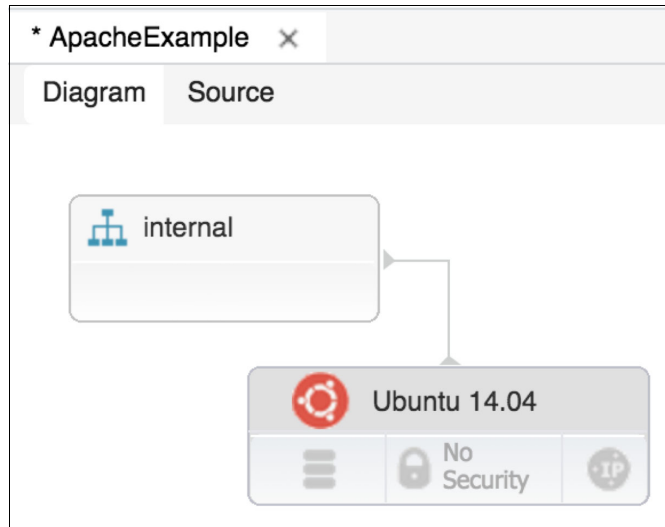


Figure 5-7 Connect network to image

5. Save the OpenStack Heat pattern by selecting **Save**.

5.3 Adding SaltStack formulas to the OpenStack Heat pattern

So far, we created a simple OpenStack Heat pattern with a single virtual machine (VM), containing an Ubuntu OS, connected to a network. As described in Chapter 1, “Automated application delivery with OpenStack Heat patterns” on page 1, to add a software component to the image, OpenStack Heat patterns leverage SaltStack scripts, or *formulas*.

In a similar fashion to *Patterns of Expertise* scripts, SaltStack formulas are not created directly on the UCD Blueprint Editor, but need to be manually uploaded to the OpenStack Heat pattern repositories to appear in the palette. This procedure is explained in Chapter 6, “Creating automation for OpenStack Heat patterns with SaltStack” on page 65 of this paper.

In the same way that scripts are added to *Patterns of Expertise*, it is possible to drag SaltStack formulas from the palette onto images. For the sample OpenStack Heat pattern described in this chapter, we use an existing formula available in the palette: One that installs an Apache web server.

To add this formula, complete the following steps:

1. Add the SaltStack formula to the OS component. The formula that is used in this paper is called *apache*. This prompts a request to select the desired SaltStack state: In essence, which Apache component should be installed. To learn more about SaltStack states and formulas, see Chapter 6, “Creating automation for OpenStack Heat patterns with SaltStack” on page 65. In this case, we choose the full *apache* installation, as depicted in Figure 5-8.

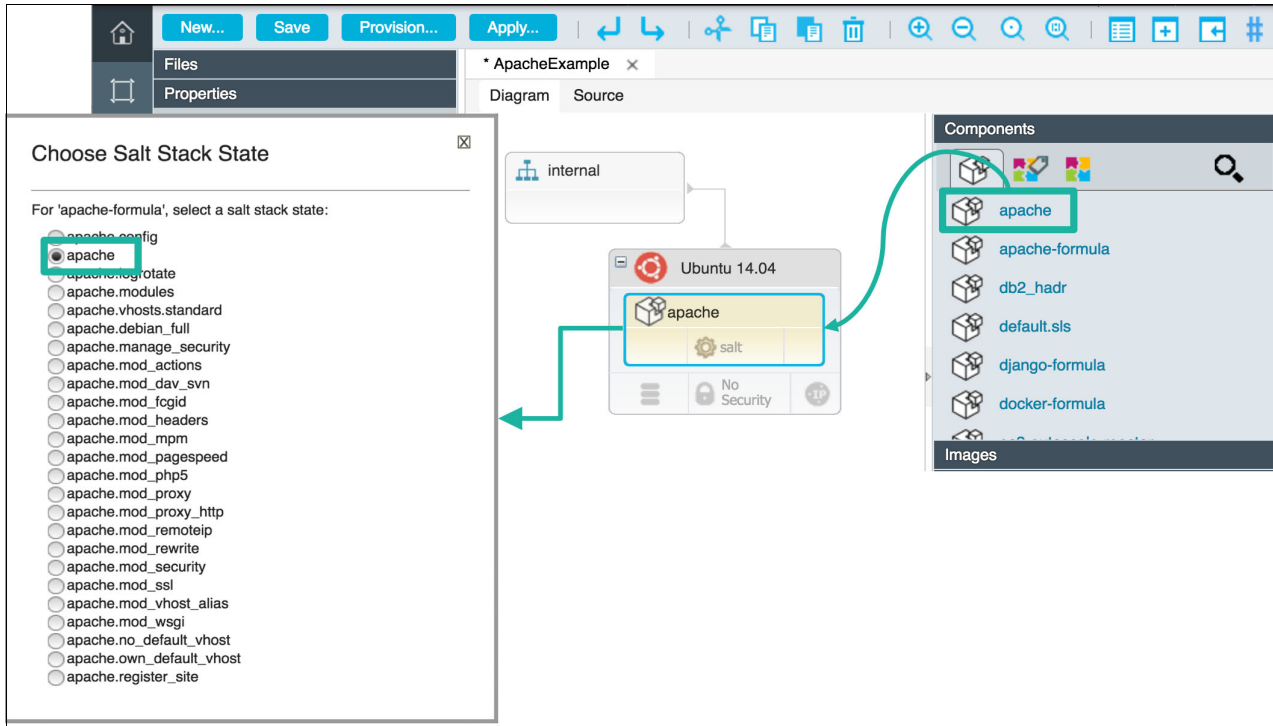


Figure 5-8 Adding the SaltStack formula, which installs Apache, to the sample OpenStack Heat pattern

Note: At the time of writing of this paper, dragging a SaltStack formula onto the blueprint does not automatically generate the full set of resources that are needed to install the required component. To add the missing resources and lines of code, you must manually edit the generated Heat Orchestration Template in the Source tab shown in Figure 5-8.

2. Save the OpenStack Heat pattern by selecting **Save**.

The creation of the pattern is complete, even if the pattern is still only saved to the local Git repository. Now the automatically generated Heat Orchestration Template can be reviewed, and edited if necessary, by clicking the **Source** tab.

Note: The OpenStack Heat pattern engine implementation in PureApplication Software uses SaltStack in the background to install Zeron and its agents. Although this is not done explicitly, it is sufficient to add any SaltStack formula to the OpenStack Heat pattern, for Zeron to be deployed together with the OpenStack Heat pattern instance.

5.4 Committing and pushing the OpenStack Heat pattern to Git

To make the newly created OpenStack Heat pattern available in the pattern catalog in PureApplication Software, the pattern needs to be committed to the local repository present in UCD Blueprint Designer, and then pushed to the PureApplication pattern repository. Both repositories are based on Git. To commit then push the pattern, complete the following steps:

1. Access the repositories view in UCD Blueprint Designer, and specifically the **patterns** repository, as shown in Figure 5-9.

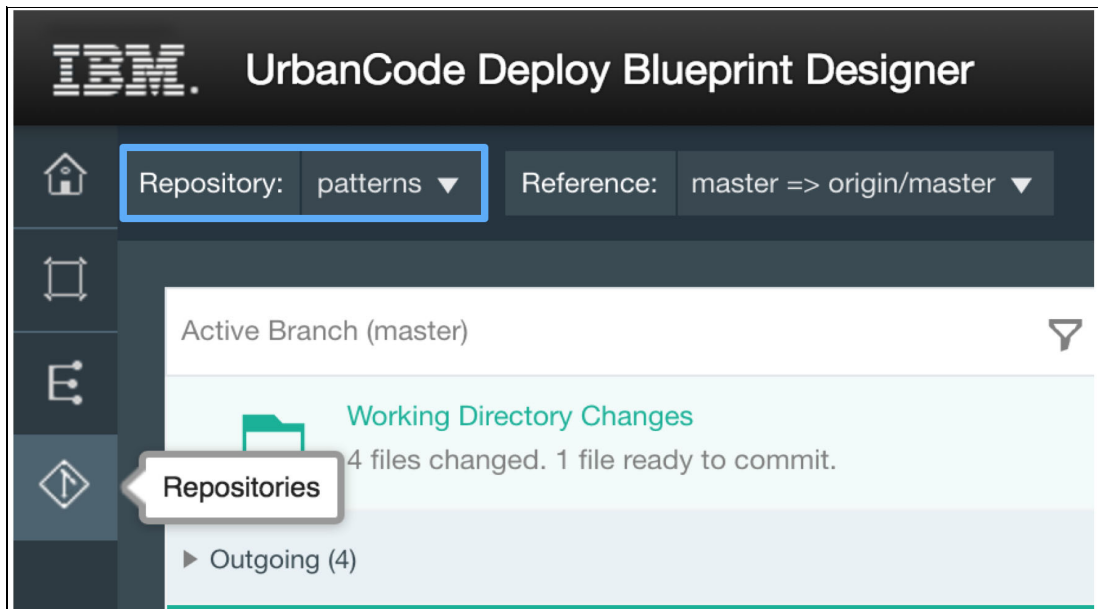


Figure 5-9 Repositories view in UCD Blueprint Designer

2. In the right section of the window, select the YAML file that describes the sample that we created, and commit the changes to the repository, as shown in Figure 5-10.

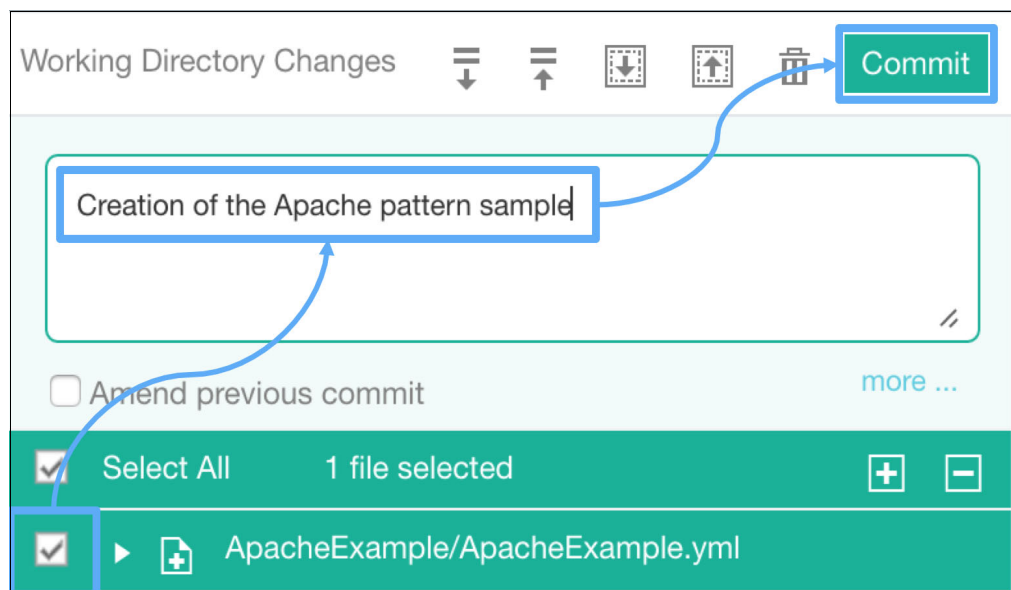


Figure 5-10 Commit the sample OpenStack Heat pattern

After changes are committed to the local repository, they can be pushed to the PureApplication repository.

3. In the left section of the window, select the commit that is related to the sample, and push the changes to the repository by selecting **Push**, as shown in Figure 5-11.

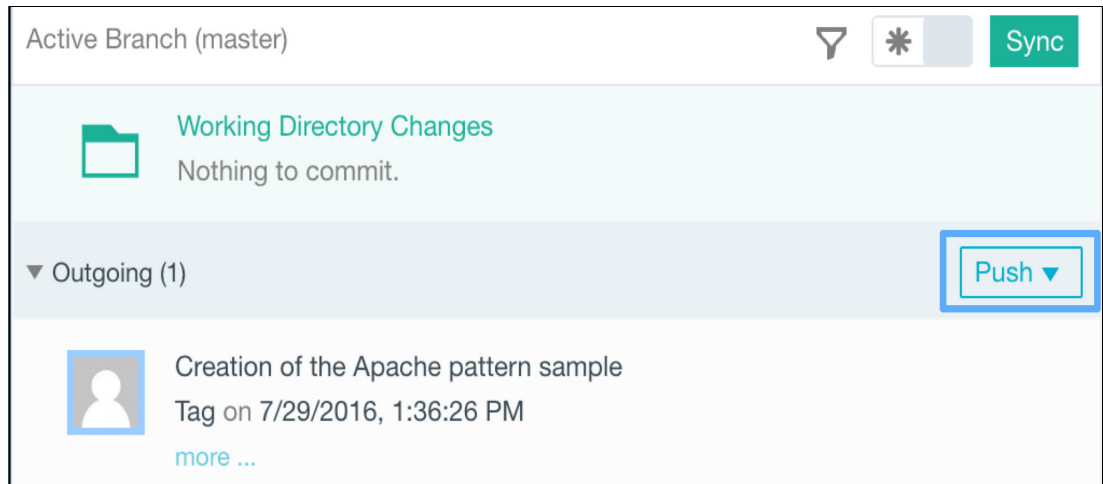


Figure 5-11 Push the sample OpenStack Heat pattern

This makes the OpenStack Heat pattern sample available in the OpenStack Heat pattern catalog that is shown in Figure 5-2 on page 54.

5.5 Deploying the OpenStack Heat pattern

When the OpenStack Heat pattern is present in the OpenStack Heat pattern catalog, it can be deployed to Bluemix Local System. The pattern is deployed in a similar fashion to the deployment of *Patterns of Expertise*, and as described in Chapter 3, “Deploying sample OpenStack Heat patterns” on page 35, for the WebSphere Application Server and DB2 example.

To deploy the sample pattern that we created in this chapter, complete the following steps:

1. Select it and click **Deploy**. This opens the deployment page shown in Figure 5-12.

Deploy OpenStack Heat pattern: [ApacheExample](#) Validate Deploy

BASIC CONFIGURATION

Name:

Environment:

DEPLOYMENT PARAMETERS

Hint: environments provide different sets of parameter default values, enabling quick and easy deployment configuration.

Software Repository URL: [Revert to default](#)

Image Parameters

flavor: [Revert to default](#)

key_name: [Revert to default](#)

availability_zone: [Revert to default](#)

Salt Deployment

Saltmaster address: [Revert to default](#)

Saltmaster api port: 8000 [Change](#)

Salt api user: [Revert to default](#)

Salt api password: ***** [Change](#)

Zeron Parameters

openstack_keystone_url: [Revert to default](#)

zeron_administrators: [Revert to default](#)

Network Parameters

network-id__for__internal: c2de9f58-8bbc-4484-8b98-14793afd8cec [Change](#)

subnet_id_f2f848d5-705a-41aa-ac4e-b9c353d1bcd8: f2f848d5-705a-41aa-ac4e-b9c353d1bcd8 [Change](#)

Figure 5-12 Deployment page for the sample pattern

The parameters in Figure 5-12 are the same as those that are present in the automatically generated HOT described in 5.3, “Adding SaltStack formulas to the OpenStack Heat pattern” on page 58.

Table 5-1, Table 5-2, Table 5-3, and Table 5-4 show the parameters, their description, and how to populate them, for the SaltStack formula, the image, the network, and the SaltStack components (in order) of the sample OpenStack Heat pattern.

Table 5-1 Software repository and Image parameters for the sample OpenStack Heat pattern

Parameter	Description	Value
Software Repository URL	Software repository URL.	Automatically populated
flavor	OpenStack Nova concept to define the size of a virtual server. A full list can be found on the following website: http://docs.openstack.org/admin-guide/cli_manage_flavors.html	For example: m1.medium
key-name	A Secure Shell (SSH) key ^a to access the virtual machines' OS when deployed.	Key that is generated in the OpenStack dashboard
availability_zone	OpenStack Neutron concept, which is defined at the time that Bluemix Local System is set up for OpenStack by using Blue Box. Additional availability zones can be defined in the OpenStack dashboard.	Corresponding value that is defined in OpenStack

a. Key pairs need to be stored in the OpenStack runtime. To do that, from the OpenStack dashboard, go to **Project** → **Compute** → **Access & Security** → **Key Pairs**. Here key pairs can be imported or generated. The **Key Pair Name** value is the one to be used for this parameter.

Table 5-2 SaltStack master parameters for the sample OpenStack Heat pattern

Parameter	Description	Value
Saltmaster address	SaltStack master URL	Can be found in System → System Settings → OpenStack services
Saltmaster API port	SaltStack master port	Can be found in System → System Settings → OpenStack services
Salt API user	User credential for the SaltStack master	Automatically retrieved from the Lightweight Directory Access Protocol (LDAP)
Salt API password	Password credential for the SaltStack server	Automatically retrieved from the LDAP

Table 5-3 Zeron parameters for the sample OpenStack Heat pattern

Parameter	Description	Value
openstack_keyston_url	OpenStack Keystone URL	Can be found in System → System Settings → OpenStack services .
zeron-administrators	List of users who can access Zeron and its Instance Console after pattern deployment.	To be added manually. Typically includes the user who deploys the pattern.

Table 5-4 Network parameters for the sample OpenStack Heat pattern

Parameter	Description	Value
network-id__for__internal	ID of the network	Automatically generated
subnet_id	ID of the subnet	Automatically generated

- When the parameters are completed, click **Validate** to confirm that the OpenStack Heat pattern sample is ready to be deployed.

Note: These parameters can be saved as an *Environment*, which is specific to a single OpenStack Heat pattern. *Environments* can be used to populate the required parameters for successive deployments of the same OpenStack Heat pattern. Multiple environment objects can be defined for the same pattern to represent different deployment configurations, for example, test and development configurations.

- Next, click **Deploy**.

Note: There are two other ways to deploy OpenStack Heat patterns:

- ▶ From the UCD Blueprint Developer
- ▶ From the OpenStack dashboard

Those additional methods are not described in this paper.

- The OpenStack Heat pattern instance view shown in Figure 5-13 (accessed by selecting **Patterns** → **OpenStack Heat pattern Instances**) shows the successful deployment of the OpenStack Heat pattern sample.

The screenshot shows the 'OpenStack Heat pattern Instances' view. On the left, there is a table of instances, all with a status of 'Create complete'. The 'ApacheExample' instance is selected. The main view shows details for 'ApacheExample':

- Stack ID:** 5b0fd695-67ff-4788-ba38-2dea06697ec4
- Description:** Created 10/5/16 by pikachu. For Cloud None
- Status:** Create complete: Stack CREATE completed successfully
- Updated on:** (none)
- Created on:** Oct 10, 2016, 3:11:05 PM
- Outputs:** apache_server: <http://9.42.44.95>

Below the details is a 'Resources' table:

Name	Type	Status	Updated on
apache_wait_condition	AWS::CloudFormation::WaitCondition	Create complete: state changed	Oct 10, 2016, 3:11:06 PM
run_orchestration	OS::Heat::SoftwareConfig	Create complete: state changed	Oct 10, 2016, 3:11:06 PM
Ubuntu_14_04	OS::Nova::Server	Create complete: state changed	Oct 10, 2016, 3:11:06 PM
apache_init	OS::Heat::MultipartMime	Create complete: state changed	Oct 10, 2016, 3:11:06 PM
Ubuntu_14_04_to__intern	OS::Neutron::Port	Create complete: state changed	Oct 10, 2016, 3:11:06 PM
apache_wait_handle	AWS::CloudFormation::WaitConditionHandle	Create complete: state changed	Oct 10, 2016, 3:11:06 PM
create_pillar_file	OS::Heat::CloudConfig	Create complete: state changed	Oct 10, 2016, 3:11:06 PM
assign_roles	OS::Heat::SoftwareConfig	Create complete: state changed	Oct 10, 2016, 3:11:06 PM
bootstrap_salt	OS::Heat::SoftwareConfig	Create complete: state changed	Oct 10, 2016, 3:11:06 PM

Figure 5-13 OpenStack Heat pattern Instances view

- By clicking **Manage** after the sample pattern deployment, it is possible to access the Zeron user interface (UI) for this OpenStack Heat pattern instance. For more information about Zeron and its UI, see Chapter 7, “Zeron for OpenStack Heat patterns” on page 75.

The sample pattern that is described in this chapter also creates a link to the Apache web server landing page, which can be selected to test the success of the deployment.



Creating automation for OpenStack Heat patterns with SaltStack

SaltStack is a suite of tools that provides the following benefits:

- ▶ Automation for enterprise information technology (IT) operations
- ▶ Event-driven data center orchestration
- ▶ A flexible configuration management for DevOps at the scale needed

This chapter provides some usage examples for SaltStack, independent from its use within OpenStack Heat patterns, to successively focus on the role that is played by SaltStack orchestration within OpenStack Heat patterns, specifically in IBM PureApplication Software on IBM Bluemix Local System.

6.1 SaltStack concepts

In Chapter 1, “Automated application delivery with OpenStack Heat patterns” on page 1, a few key SaltStack concepts were introduced. They are listed in Table 6-1.

Table 6-1 SaltStack concepts

Concept	Description
Master	Main server that controls the infrastructure and SaltStack application engine, working as a repository for data and a control point for operations between the minions.
Minion	System process, which enables communication with the master and the ability to run instructions received from the master. It runs in the virtual machine or the container to be configured at the time of deployment.
State	Set of rules and configurations that describe a desired state for a particular minion.
Formula	A collection of states, for example, the steps that need to be followed to install a program. Each step is represented by a state.
Grain	Interface to derive information about the underlying system where a minion is hosted. This information is configured on the minion and sent back to the master. Most grains are auto configured: For example, the host name and the default gateway of a system. It is also possible to create custom grains, such as the roles that a container or virtual machine (VM) fulfills.
Pillar	Essentially the reverse of a grain: Information that is passed by a master down to a minion. A minion cannot get pillar data intended for other minions, only for itself. There is no auto-configured pillar data, only custom.
Execution Functions	Commands running specific functions.
Execution Modules	Groups of execution functions that are grouped by affinity.
Reactor	A function that can be used to extend an existing execution function or module, and which can be exposed as a REST API on the master. For example, a reactor to pass a key to a minion and make it accept it automatically.
Mine	A database that is running on the master to keep information directly in the master. It is used for the kind of information that is not yet available at the time of the minion deployment, which would otherwise be part of the pillar data. For example, the dynamically assigned IP addresses for all minions.

In the following sections, we further describe some of these concepts.

6.1.1 Master and minion communication

A SaltStack master and SaltStack minions communicate by using Transmission Control Protocol (TCP) through ports 4505 and 4506.

The configuration files in SaltStack define the names for master and minions by using an Internet Protocol (IP) address, or a Fully Qualified Domain Name (FQDN).

Key exchange

When the configuration files described previously are defined, and the SaltStack services (specifically master and minions) are started, SaltStack initiates a process of key exchange, in which the minions send their keys to the master to be accepted.

A trust relationship is established between master and a minion when the master accepts an Advanced Encryption Standard (AES) key that is sent from that minion. This secures the channel, ensuring that no form of tampering with the data can be performed.

Example 6-1 shows the list of keys that are received by the master and waiting to be accepted. Note that the command-line argument `-L` is used to list them all.

Example 6-1 Key management, List keys

```
[root@master ~]# SaltStack-key -L
Unaccepted Keys: alpha bravo charlie delta
Accepted Keys:
```

A SaltStack command can be triggered either automatically or manually, for the master to accept the keys that were sent to it.

Example 6-2 shows the command to accept all keys (command line argument `-A`) and the list of keys after they are accepted.

Example 6-2 Key management - List keys

```
[root@master ~]# SaltStack-key -A
[root@master ~]# SaltStack-key -L
Unaccepted Keys:
Accepted Keys: alpha bravo charlie delta
```

After the keys are accepted, communication between master and minions is established. This action enables the master to perform remote execution of modules, functions, orchestration files, and configuration files onto the minions.

6.2 General SaltStack examples

This section provides usage examples for SaltStack, independently from its use within OpenStack Heat patterns.

The examples included in this introduction are not extensive. SaltStack provides modules and functions to perform various operations, such as checking service status, running custom commands and programs, and querying information from the minions. For more information about these and other examples, see the following website:

<https://docs.saltstack.com/en/latest/>

6.2.1 Remote execution

The most basic form of command execution in a SaltStack installation is given in the command (template) in the Basic operation template shown in Example 6-3.

Example 6-3 Basic operation template

```
sudo <Salt> <options> '<target>' <function> <arguments>
```

The elements are described in the following list:

Salt	The name of the SaltStack execution program.
Options	Includes specifics for debugging or verbose.
Target	Identifies the target minion where the command is going to be run. This field supports regular expressions to match minions' names.
Function	SaltStack execution function or module, as defined in Figure 6-1 on page 66.
Arguments	Data that is passed to the function or module as parameter.

An example of this command is given in Example 6-4.

Example 6-4 Using grain information to filter commands

```
sudo Salt -grain 'os_family:RedHat' test.ping  
myminion: True
```

Details about these elements are described in the following list:

- ▶ `-grain 'os_family:RedHat'` indicates that we are filtering by using grain information: Specifically, all minions that are running a Red Hat operating system (OS).
- ▶ `test.ping` runs a communication test with the target minions.
- ▶ The output `myminion: True` indicates a successful communication with the selected minions.

6.2.2 Package installation

Following the template that is given in Example 6-3 on page 68, the command given in Example 6-5 installs a nginx web server package into all minions. For more information, see the nginx website:

<https://nginx.org/en/>

Example 6-5 Package installation command

```
sudo Salt '*' pkg.install nginx
```

The elements are as follows:

- ▶ `Salt` is the name of the SaltStack execution program.
- ▶ `'*'` is a wildcard that selects all minions.
- ▶ The `pkg.install` module is the install function for Linux packages.
- ▶ `nginx` is passed as an argument to the `pkg.install` function, which is the package to install.

6.2.3 SaltStack states

The examples given so far are single commands that implement a particular aspect of the infrastructure or application automation. However, as mentioned in Table 6-1 on page 66, SaltStack goes beyond single instructions and can provide a framework to create *states* that are applied to minions. The minions use the state execution module that is defined within Salt state files (files) to bring their host system into the desired state. Example 6-6 shows a command run by a minion to reach the desired state.

Example 6-6 Applying a state to a minion

```
sudo SaltStack '*' state.sls apache
```

The following list describes the elements in Example 6-6:

- ▶ `state.sls` identifies the Salt state file that contains the execution module.
- ▶ `apache` is the name of the state that is defined in the SLS file.

The `state.sls` file is a YAML file that is stored in the master, which could look like Example 6-7.

Example 6-7 The state.sls file in the SaltStack master

```
install_apache:
  pkg.installed:
    - name: apache
```

6.2.4 Formula files

Formulas are a pre-written collection of states. Their capabilities are the same as the ones of a state file, so they can, for example, install a package, configure, or control a service or an OS task, and provide similar functions.

Formulas are defined in a *top file*, which has the following structure:

- ▶ Environment: A tree directory that contains a set of *state* files to configure systems
- ▶ Target: A group of hosts that are running minions, to which *state* files will be applied
- ▶ State files to apply to a target

An environment contains targets, and a target contains states.

The default name for *top files* is `top.sls`. Their name derives from the fact that they are typically stored at the top of the directory hierarchy that contains state files. This directory hierarchy is known as a *state tree*. Example 6-8 shows a formula that defines an environment that is called “base”, with its target. The target is made up by the hosts that are running a minion with an ID beginning with the word “web”. The *state file* called “apache” is applied to the hosts in the target.

Example 6-8 SaltStack formula example

```
base:
  'web*':
    -apache
```

A repository of publicly available formulas is available within the SaltStack community at the following Salt Stack Formulas website:

<https://github.com/saltstack-formulas>

6.2.5 Git repositories and SaltStack files

Source control and repository operations are usually performed by using version control tools, so that artifacts, such as formulas and states, can be reused. SaltStack provides interfaces to connect to such tools, the most popular being Git.

Example 6-9 shows how a GitFS file server can be configured within the SaltStack configuration files, to define endpoints from which SaltStack services retrieve states and formulas. See the SaltStack website for details:

<https://docs.saltstack.com/en/latest/topics/tutorials/gitfs.html#tutorial-gitfs>

Example 6-9 Git repositories definition

```
Master config:
fileserver_backend: - git
gitfs_remotes:

- git://github.com/saltstack/salt.git
- git://github.com/saltstack/salt-states.git
- git://github.com/saltstack/salt-ci.git
```

6.3 Using SaltStack within OpenStack Heat patterns

This section explains in detail the interaction between SaltStack and OpenStack Heat patterns in PureApplication Software. The core of the interaction happens when you provision an OpenStack Heat pattern. The interaction is from the creation of the minions' installation and configuration, retrieving information from the relevant repositories, to the final report when the orchestrated deployment completes.

Different from using SaltStack in isolation, within OpenStack Heat patterns some operations, such as minion installation and key management, are performed automatically by the *OpenStack Heat pattern engine*.

As mentioned in Chapter 2, "Setting up IBM Bluemix Local System for OpenStack Heat patterns" on page 25, the *OpenStack Heat pattern engine* implementation in PureApplication Software provides four Git repositories. These four repositories serve as version control tools for OpenStack Heat patterns and SaltStack modules, pillars, and formulas:

- ▶ Pattern repository, where the OpenStack Heat patterns are stored
- ▶ Module repository, where the SaltStack execution modules are stored
- ▶ Pillars repository, where the SaltStack pillars are stored
- ▶ Formulas repository, where the SaltStack formulas are stored

The remainder of this section describes the workflow that is followed when deploying an OpenStack Heat pattern instance. Particular attention is given to the role that SaltStack plays within the orchestration, and in relation to OpenStack Heat.

The deployment workflow is depicted in Figure 6-1.

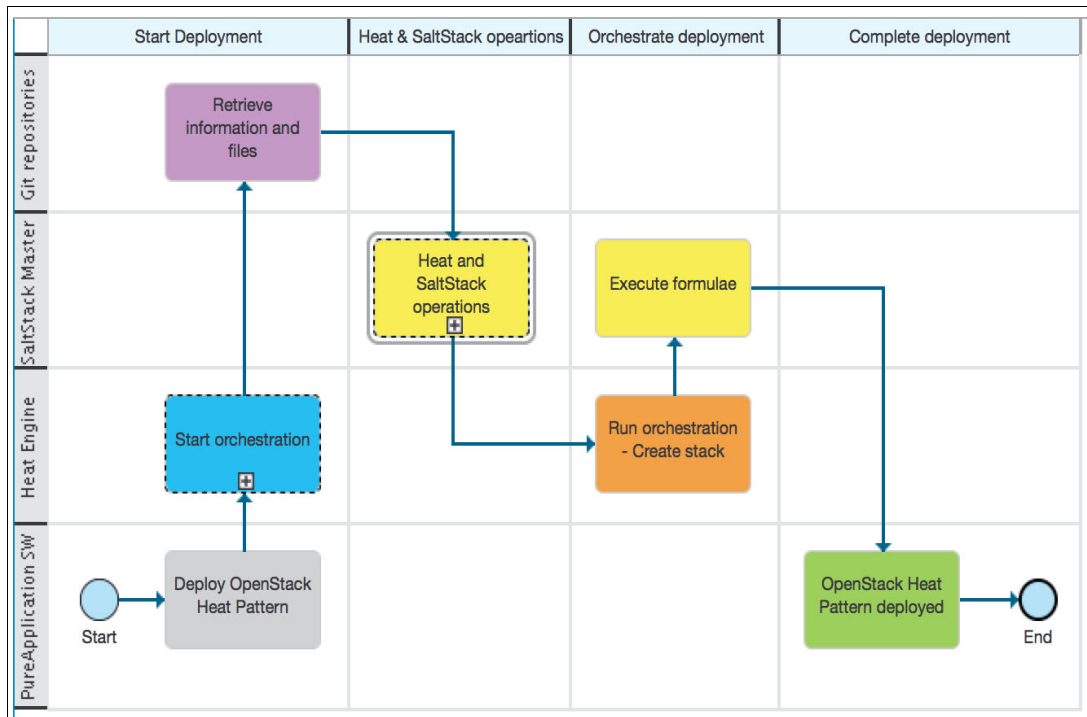


Figure 6-1 End-to-end process for deployments that are supported by SaltStack in OpenStack Heat patterns

The workflow of Figure 6-1 can be further expanded, as shown in Figure 6-2.

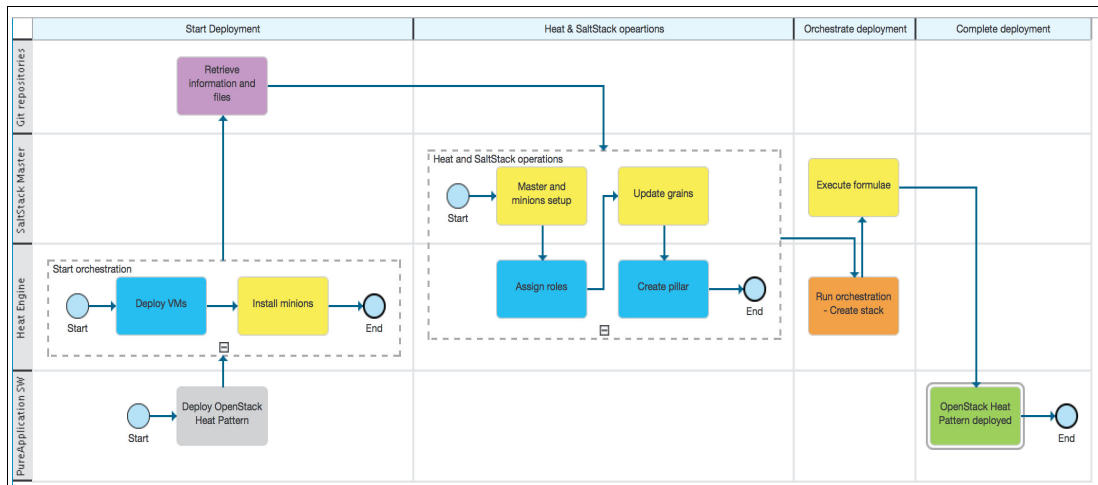


Figure 6-2 Expanded process for deployments that are supported by SaltStack in OpenStack Heat patterns

Each step in the workflow that is represented in Figure 6-2 is explained in detail in the following sections.

6.3.1 Deploying an OpenStack Heat pattern

An OpenStack Heat pattern is typically deployed by following the method that is described in Chapter 3, “Deploying sample OpenStack Heat patterns” on page 35 and Chapter 5, “Creating an OpenStack Heat pattern” on page 53. Figure 6-3 shows the deployment from the deployment page.

The screenshot shows a web interface for deploying an OpenStack Heat pattern. The title is "Deploy OpenStack Heat pattern: IBM WebSphere ND v855 Standalone". There are two buttons: "Validate" and "Deploy". The interface is divided into two main sections: "BASIC CONFIGURATION" and "DEPLOYMENT PARAMETERS".

BASIC CONFIGURATION

- Name:
- Environment:

DEPLOYMENT PARAMETERS

Hint: environments provide different sets of parameter default values, enabling quick and easy deployment configuration.

Host Name Parameters

was_vm_name	ibm_was_v855_01	Change
-------------	-----------------	------------------------

Image Parameters

Flavor	<input type="text" value="m1.medium"/>	Revert to default
SSH Key name	<input type="text" value="Default"/>	Revert to default
Availability zone	<input type="text" value="nova"/>	Revert to default
OS Image	<input type="text" value="RHEL 7.2"/>	Revert to default

Figure 6-3 OpenStack Heat pattern deployment page

6.3.2 Starting Heat orchestration

When the Heat engine receives the template that is contained in the OpenStack Heat pattern, it initiates the process to collect the infrastructure components by interacting with the rest of the OpenStack services. For example, it retrieves the image file for the OS from OpenStack Glance, or gathers the ID of the selected network from the OpenStack Neutron service.

6.3.3 Deploying virtual machines and install minions

The Heat engine also performs a critical step in the provisioning process: Loading and installing the virtual machines to be deployed with the SaltStack minions.

6.3.4 Retrieving information and files from Git repositories

As mentioned before, SaltStack interacts with the Git repositories in the *OpenStack Heat pattern engine* to store information about formulas, states, execution modules, and pillars. After the virtual machines are deployed, as described in 6.3.2, “Starting Heat orchestration”, the SaltStack orchestration retrieves information from these repositories that is later used to configure the minions, and to apply states and formulas to their hosts.

6.3.5 Heat and SaltStack operations

This section describes the SaltStack master and minions activities.

Master and minions setup

The *OpenStack Heat pattern engine* implementations that are described in this IBM Redbooks publication use only one SaltStack master. Moreover, they enforce a one-to-one relationship between a minion and its host.

After the initial handshake between the master and the minions occurs, the master performs the initial operations in the minions, such as registering their Secure Sockets Layer (SSL) keys and setting up their names.

Assigning Roles

The next step in the workflow is the Heat engine creating grains to assign roles and configuration values to the minions and their hosts.

Two main custom grains are assigned:

Stack ID	Used to identify all minions that are hosted on virtual machines that are part of the same OpenStack Heat pattern instance, or stack.
Role ID	Used to define the purpose of the virtual machine that hosts the minion within the OpenStack Heat pattern (for example, a web server or a deployment manager).

Note: These two custom grains enable identification of the correct virtual machines on which to perform typical pattern operations, such as monitoring individual pattern components.

Updating grains

The minions are updated with the custom grains that are created by the Heat engine.

Creating a pillar

In this step, the Heat engine creates a SaltStack pillar file that contains data specific to the OpenStack Heat pattern to be deployed. The pillar file is then stored in the master, and the data it contains is used by the minions during and after the deployment process.

6.3.6 Running orchestration and running a formula

At this point, the SaltStack formula is run in all minions, bringing them into their desired state. In this phase, elements (such as middleware installation and configuration) are applied, depending on the pattern definition. For example, the required software components are installed in each host.

Every time a state is reached, the minions reply to the master with their status. The master, in turn, updates the Heat engine with the results of the operations.

6.3.7 OpenStack Heat pattern deployed

When the SaltStack and Heat operations are completed, the OpenStack Heat pattern instance is updated with all details regarding its different components. This process also updates the OpenStack Heat pattern Instances page, where the successful deployment of the pattern is displayed, along with the information relative to the deployed resources. An example is shown in Figure 6-4.

The screenshot shows the details of an OpenStack Heat pattern instance named 'Example_WAS_v855_Standalone'. The instance is in a 'Create complete' state, indicating a successful deployment. The status bar at the top right includes options for Refresh, Manage, Start, Stop, and Delete. The instance ID is 84678e95-1e57-43d1-806b-fa6a9f4ad1e6. The description is 'A document-based template to configure your Software Defined Environment.' The instance was created on Jul 19, 2016, at 8:37:08 PM. The blueprint URI is https://9.3.169.204:9043/libm/console. The Resources section is expanded, showing a table of 13 resources, all of which are in a 'Create complete: state changed' state.

Name	Type	Status	Updated on
wait_condition	AWS::CloudFormation::WaitCondition	Create complete: state changed	Jul 19, 2016, 8:37:08 PM
wait_handle	AWS::CloudFormation::WaitConditionHandle	Create complete: state changed	Jul 19, 2016, 8:37:09 PM
setup_yum_repo	OS::Heat::CloudConfig	Create complete: state changed	Jul 19, 2016, 8:37:09 PM
was_standalone_init	OS::Heat::MultipartMime	Create complete: state changed	Jul 19, 2016, 8:37:08 PM
prepare_yum	OS::Heat::SoftwareConfig	Create complete: state changed	Jul 19, 2016, 8:37:09 PM
run_orchestration	OS::Heat::SoftwareConfig	Create complete: state changed	Jul 19, 2016, 8:37:09 PM
create_pillar_file	OS::Heat::SoftwareConfig	Create complete: state changed	Jul 19, 2016, 8:37:09 PM
assign_roles	OS::Heat::SoftwareConfig	Create complete: state changed	Jul 19, 2016, 8:37:09 PM
setup_ssh	OS::Heat::SoftwareConfig	Create complete: state changed	Jul 19, 2016, 8:37:09 PM
bootstrap_salt	OS::Heat::SoftwareConfig	Create complete: state changed	Jul 19, 2016, 8:37:09 PM

Figure 6-4 OpenStack Heat pattern instance successfully deployed



Zeron for OpenStack Heat patterns

Students of chemistry know that Zeron is a highly alloyed stainless steel. In the world of cloud software patterns, Zeron is appropriately named because it tries to bridge the gap between IBM PureApplication System patterns and OpenStack Heat patterns, from a lifecycle management and operations standpoint.

As stated in Chapter 1, “Automated application delivery with OpenStack Heat patterns” on page 1, IBM created a framework to manage post-deployment actions through the automation scripting technologies. This framework is named Zeron and is written in Python.

The goal of Zeron is to provide a runtime for the complete lifecycle management of all patterns. This lifecycle includes monitoring, shared services, operation, logging, patching, and license management. Another tenet of Zeron is for it to be a lightweight pluggable runtime that runs in virtual machines (VMs) and in software containers.

Tip: The maestro framework was used to manage PureApplication System patterns. The Zeron framework is used to do the same thing with OpenStack Heat patterns.

7.1 Zeron architecture

The code and the infrastructure to run Zeron is less than 5 megabytes (MB), therefore requiring a small footprint. This footprint means that it does not take up many resources: processing (CPU), memory, or disk space. Zeron is designed to be a pluggable software component. As mentioned, it provides monitoring, logging, patch management, and license management by way of pluggable handlers.

The handler interface is simple and file-based. The default handlers provide basic functionality, and even provide interfaces for middleware that is running in VMs or containers.

Figure 7-1 shows the Zeron topology with Zeron instances that are running in virtual machines (VMs).

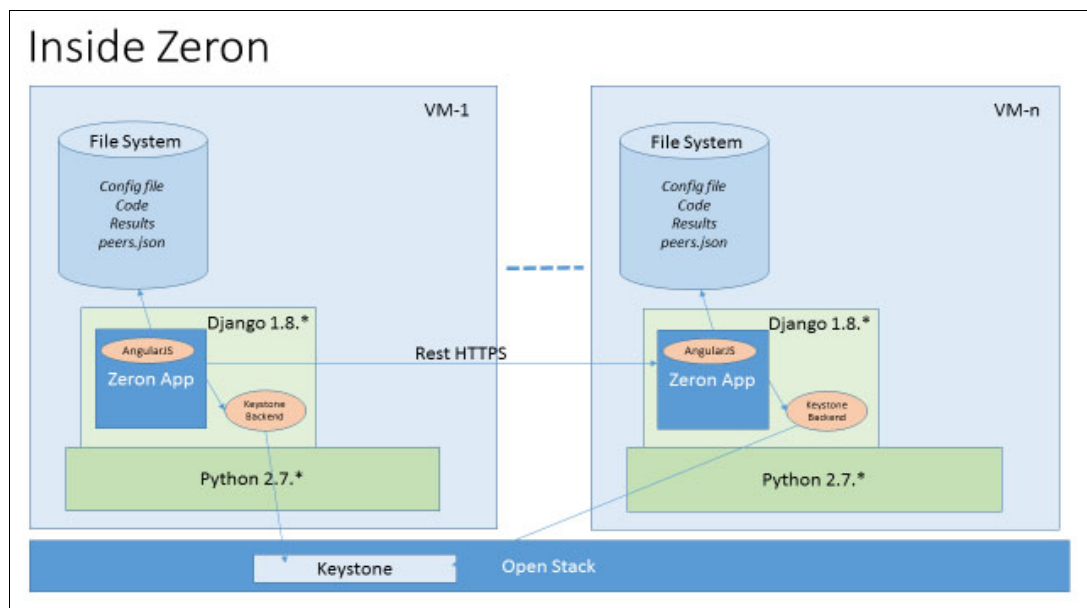


Figure 7-1 Zeron topology

Zeron can run both in IBM Bluemix Local System and in IBM Bluemix Cloud. It is worth noting that there are handlers available for IBM Bluemix that provide logging and monitoring services.

The following section defines in more detail Zeron's terminology, security, and components.

7.1.1 Zeron terminology

The following terminology is used in Zeron:

- ▶ ServiceType

A set of services that are supported by Zeron. Each service has some configuration data that needs to be provided by the service users. Each service can have multiple implementations, but each has a default implementation that is provided by Zeron.

- ▶ Handler

A handler is piece of software that provides the implementation to the ServiceType. It must implement a Python class with the name `service.py` in the top directory, along with its code packaged as a `.zip` or `.tgz` file.

The handler configuration file is in the following directory:

```
$ZERON_ROOT/services/handlers/<service_type>/<handler_name>/handler.json
```

For example: `$ZERON_ROOT/services/handlers/logging/bluemix/handler.json`

► **SoftwareComponent**

A piece of middleware component that needs to be installed in a VM or in a container. The pattern developer has to provide the software component that gets deployed in each VM. Each software component needs to provide information for each of the ServiceTypes that it wants to use. The middleware provides the following JSON files: `logging.json`, `monitoring.json`, `operations.json`, `patches.json`, `license.json`, and `tracking.json`.

These files get stored in the following directory structure:

```
$ZERON_CFG_ROOT/services/config/<service_type>/<software_type>/<software_name>/
```

For example, `logging.json` for WebSphere Application Server would be in the following location:

```
$ZERON_CFG_ROOT/services/config/logging/was/dmgr/logging.json
```

7.1.2 Security in Zeron

From a security perspective, all of the user interfaces (UIs) and REST application programming interface (API) endpoints are protected by authentication and role-based access control (RBAC), and all communication is over secure Hypertext Transfer Protocol (HTTP), specifically HTTPS. Zeron supports pluggable authentication backends, such as the OpenStack Keystone when running within OpenStack, and a local Django database backend for testing.

Users are grouped into three roles: Admin, Read Only, and Read-Write. Table 7-1 shows the three user roles and their privileges.

Table 7-1 Zeron user roles

Role	Privileges
Admin	Can perform only administrative work but no execution of operations
Read Only	Can only view but cannot call operations or perform any changes
Read-Write	Can perform Read Only tasks and can run operations (write)

7.1.3 Inside Zeron

The Zeron software package contains the following items:

- *Django*, which is a Python-based web server.
- A set of files that defines interfaces.
- The `peers.json` file defines the set of Zeron nodes (VM or container) with their Internet Protocol (IP) and port information.
- The `handler.json` file defines the details of a handler to perform one or more service type implementations. For example, there are two handlers for license management: Default and IBM License Metric Tool (ILMT).
- Each service implementation defines what it needs to configure.
- The `inventory.json` file enables each software component to announce its information to Zeron.

- ▶ The `<serviceType>.json` file defines the service needed and provided by service software defined in the handler.
- ▶ The `logging.json` file defines the set of directories and file patterns that middleware wants the logging service type to show.
- ▶ The `monitoring.json`, `operation.json`, `fixes.json`, and `license.json` files are similar to the logging handler.
- ▶ `ZERON_ROOT`, where the Zeron runtime is located.
- ▶ `ZERON_CFG_ROOT`, where software components declare their information for Zeron.

Section 7.3, “Zeron handlers” briefly describes the default handlers, with a sample listing in 7.3.4, “Operation handler type”.

7.2 Zeron user interface

The Zeron user interface has the same user functionality no matter where it runs. However, it can be re-skinned to create a custom look. It can be accessed on port 8000:

`https://<Server_IP>:8000/zeron/console`

7.2.1 Zeron UI for OpenStack Heat patterns on PureApplication Software

After an OpenStack Heat pattern is deployed, as explained in Chapter 3, “Deploying sample OpenStack Heat patterns” on page 35, you can go to the instance details and click **Manage**, as shown in Figure 7-2.

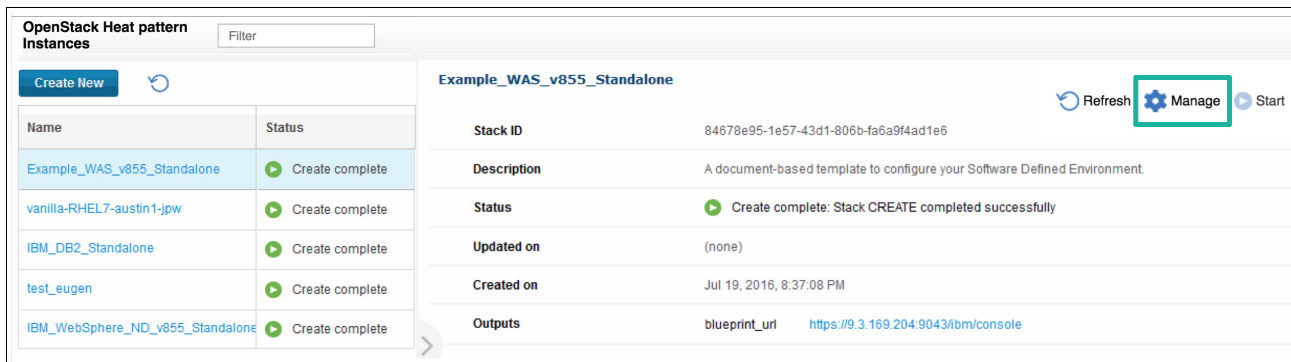


Figure 7-2 Manage menu in a pattern instance

That action opens the Zeron UI in a separate web browser window, as shown in Figure 7-3 on page 79. You will notice all the default handlers (Logging, Monitoring, Operations, License Tracking, Fixes, and Links) as menu options, plus the Administration menu.

Figure 7-3 on page 79 also shows the Administration menu that has two sub-menus: Handlers Configuration and Software Inventory. Note that the Logging handler is highlighted.

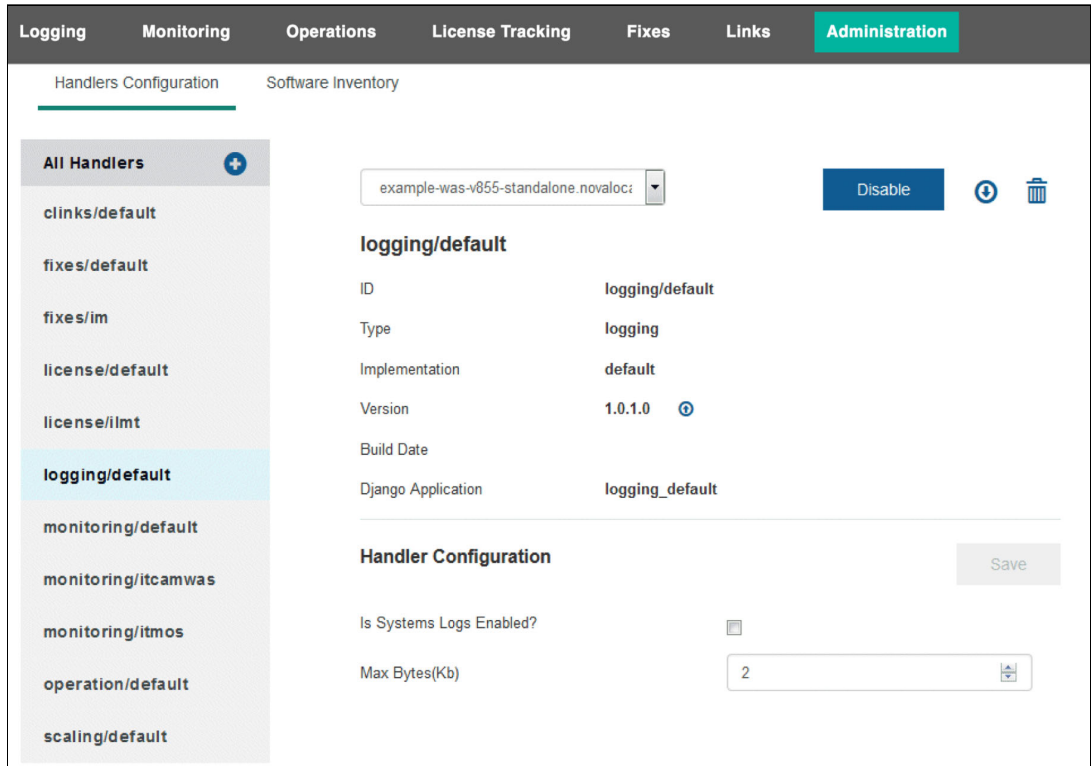


Figure 7-3 Administration menu after logging in

There are four operations that can be performed on the Administration window:

- ▶ **Disable:** To disable the handler, click **Disable**.
- ▶ **Download:** To download the handler from that node, click the downward facing arrow icon. As an example, the handler shown in Figure 7-3 is saved as logging_default.tgz.
- ▶ **Uninstall:** Click the **trash can** icon to uninstall the displayed handler.
- ▶ **Install:** Click the Plus sign (+) to install a new handler. Figure 7-4 shows the pop-up window that you get when you choose to install a new handler. After you specify the handler package file, click the **Upload** button to actually install the handler.

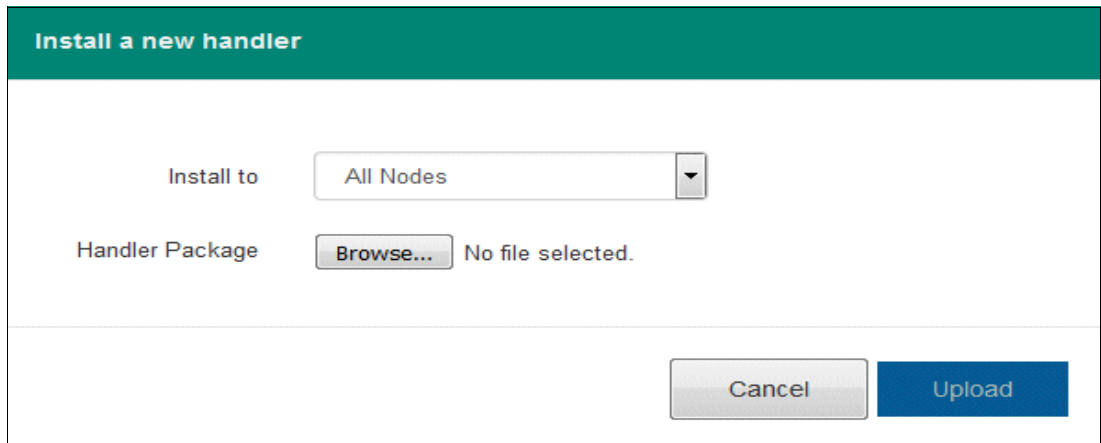


Figure 7-4 Pop-up window when installing a new handler in Zeron

Logging Menu

When you view the Logging option, you see the Default handler information as seen in Figure 7-5. To download the log file, click the down arrow or expand the instance and view the log in that window.

Figure 7-5 shows the server status log of the WebSphere Application Server pattern that was deployed in Chapter 3, “Deploying sample OpenStack Heat patterns” on page 35.

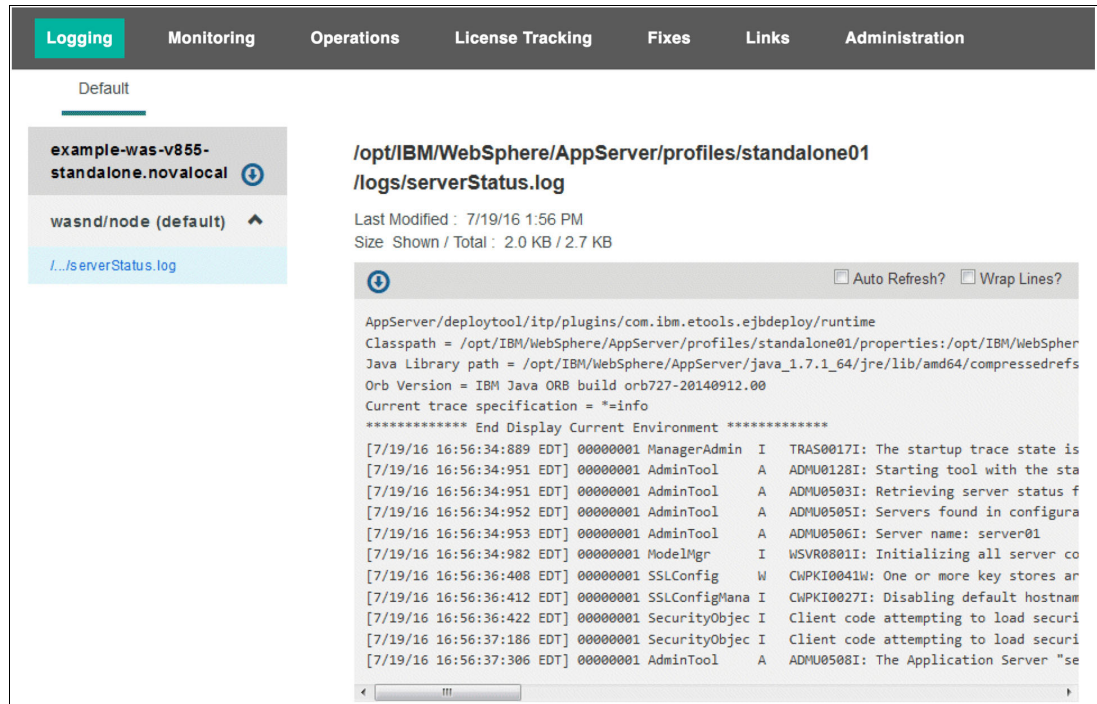


Figure 7-5 Logging menu in Zeron

Monitoring Menu

Similarly, the Monitoring option displays the Default handler information, as shown in Figure 7-6 on page 81. You can highlight the component on the left and view various resource usage graphs in real time.

The resources that are displayed in digital or graphical form include the following information:

- ▶ Virtual CPUs
- ▶ Virtual Memory
- ▶ Storage Size
- ▶ System Health
- ▶ CPU Usage
- ▶ Memory Usage
- ▶ Root Disk Usage
- ▶ Network Received Byte
- ▶ Network Transmitted Byte

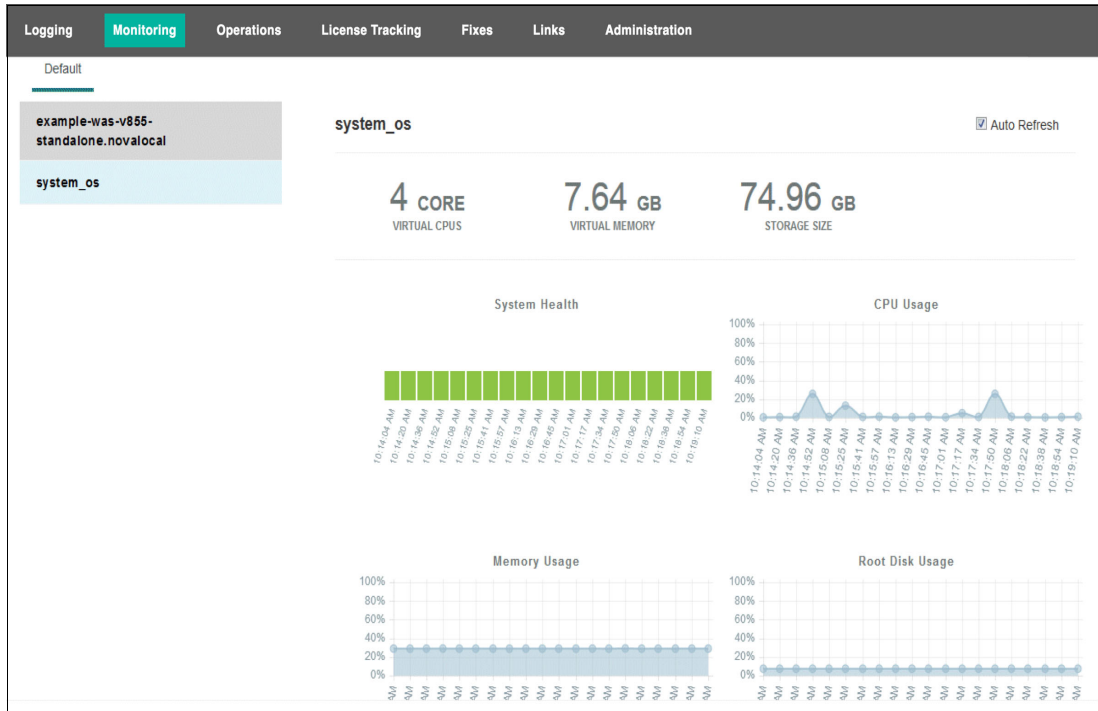


Figure 7-6 Monitoring menu in Zeron

Operations menu

The Operations menu shows the contents of its Default handler. Highlighting a component on the left displays information about it. Figure 7-7 shows details about the IBM Installation Manager contained in the IBM WebSphere Application Server OpenStack Heat pattern.

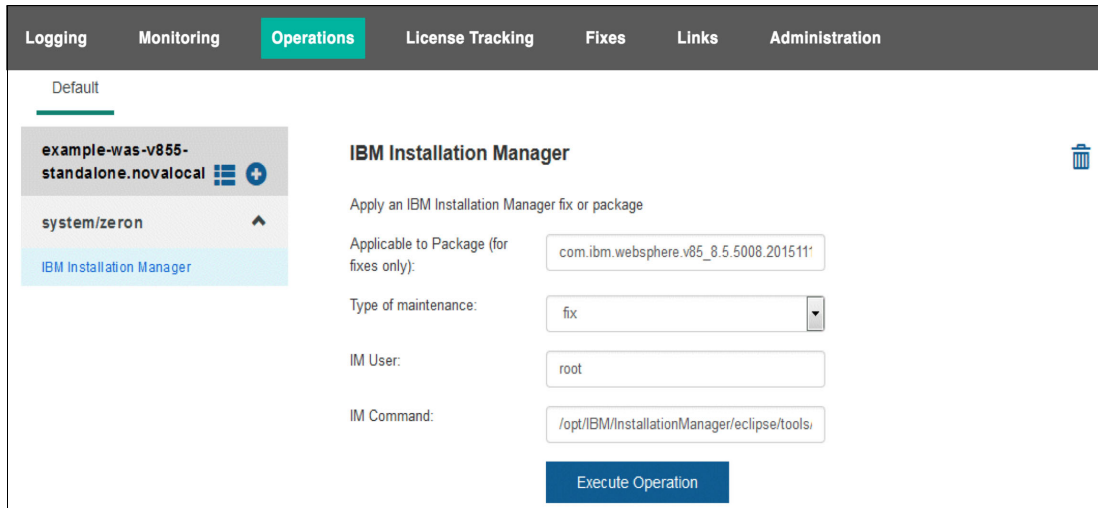


Figure 7-7 Operations menu in Zeron

Operations depend on the software component. IBM Installation Manager can be used to apply a maintenance fix or a package. Such operations can be run by choosing the type of maintenance and clicking **Execute Operation**. You also have the option to install a new operation by clicking the Plus sign (+) on the left.

If you install a new operation, the appropriate Operation Package must be supplied, as shown in Figure 7-8.

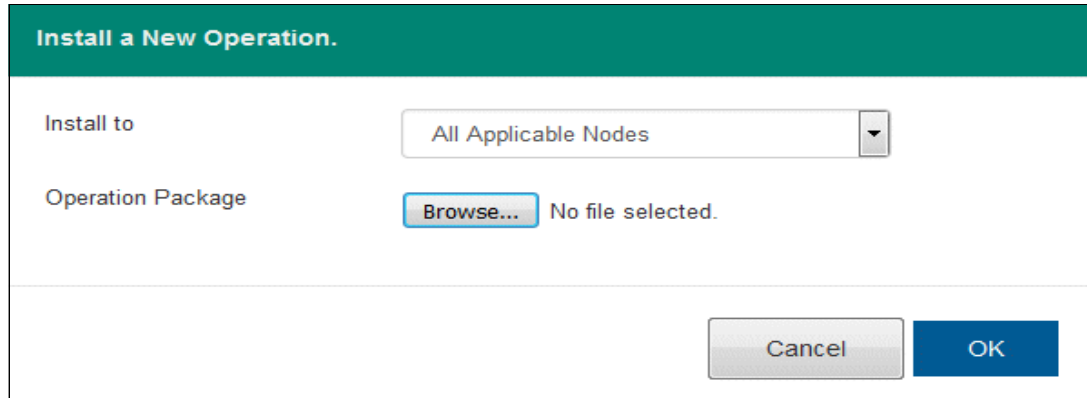


Figure 7-8 Installing a new operation in Zeron

The remaining menu options, License Tracking, Fixes, and Links, work in a similar manner.

7.3 Zeron handlers

This section describes the main types of handlers used in Zeron.

7.3.1 Inventory handler type

This service lists all software that is installed on each VM or container in a pattern. Each software component that expects to utilize Zeron creates an `inventory.json` file and stores it in the following directory:

```
${ZERON_CFG_ROOT}/services/inventory/<sw_type>/<sw_name>/inventory.json
```

Each `inventory.json` file provides the following information:

- ▶ All the middleware software types and running instances
- ▶ Version of all software
- ▶ Any environment variables and their values

There is an `inventory.json` file for the Zeron node itself, which contains the Zeron node name, and the location information for `peers.json`. It can be found in the following directory:

```
${ZERON_CFG_ROOT}/services/inventory/__node__/__node__/
```

7.3.2 Logging handler type

Zeron currently provides two logging handlers: Default and Bluemix. Each middleware product provides a `logging.json` file. For example, WebSphere Application Server provides the following file:

```
${ZERON_CFG_ROOT}/services/logging/wasnd/dmgr/logging.json
```

Like other handler JavaScript Object Notation (JSON) files, the `logging.json` file contains the name, directory, and pattern of files that logs for its software instance.

The Default handler shows the logs in the text area of fixed size. Users who have access also have the ability to download the log file. The fixed format size is configurable by the Zeron administrator.

The Bluemix handler pushes those logs to IBM Bluemix ELK¹ logging service, Kibana. For those unfamiliar with ELK, the ELK stack contains three components: Elasticsearch, Logstash, and Kibana. For more information, see the following web address:

<http://www.ibm.com/developerworks/library/ba-bluemix-elklog/index.html>

Similar to what was shown in Figure 7-3 on page 79, the Zeron UI provides the Uniform Resource Locator (URL) to log on to IBM Bluemix and view the Bluemix logs.

7.3.3 Monitoring handler type

Similar to logging, Zeron provides two monitoring handlers: Default and Bluemix. Each middleware product provides a `monitoring.json` file with the file path where it stores monitoring data. For example the operating system (OS) monitoring can be found at the following website:

```
${ZERON_CFG_ROOT}/services/monitoring/system/os/monitoring.json
```

The monitoring handler supports multiple collector types, which are various sources, including file, script, and REST API.

The default handler is read at periodic intervals, and it defines where to find the monitoring data file. The Zeron UI shows data in graphical format, in addition to static information, as we saw in Figure 7-6 on page 81.

The Bluemix handler installs a Grafana client to push data from the VM to the server. The following login URL is available on IBM Bluemix to see the graphs and analysis:

<https://logmet.ng.bluemix.net>

For more information, see the following article:

<https://developer.ibm.com/bluemix/2016/01/22/bluemix-monitoring-dashboard-quick-reference/>

7.3.4 Operation handler type

Unlike logging and monitoring, Zeron provides only one handler for Operation, which is Default. Each middleware product provides an `operation.json` file and the supported files, such as a `.tgz` file. For example, WebSphere Application Server provides a file from the following directory:

```
${ZERON_CFG_ROOT}/services/operation/wasnd/dmgr/operation.json
```

This file contains the list of operations, each with a set of parameters, scripts, and other configuration information. Figure 7-7 on page 81 showed the window that displays when installing a new Operation.

¹ For more information about the Elastic Stack, see the following website:

<https://www.elastic.co/products?camp=Branded-GGL-Exact&src=adwords&mdm=cpc&trm=elk%20stack&gclid=CPa bx82jxs4CFcZbhgodLOYN8w>

Example 7-1 is an example of the handler.json file for the default Operation handler. Notice the three key/value pairs toward the end of the file: service_type, service_impl, and django_app.

Example 7-1 The handler.json file for the Default Operation handler

```
{
  "enabled" : true,
  "configuration": {
    "MAX_RESULTS_DISK_SIZE_MB": {
      "default_value": "10",
      "description": "max_results_disksize_desc",
      "max_value": "512",
      "value": "",
      "min_value": "0",
      "is_required": false,
      "regex": "",
      "type": "int",
      "name": "max_results_disksize_mb"
    },
    "MAX_DAYS_TO_KEEP_RESULTS": {
      "default_value": "365",
      "description": "days_keep_results_desc",
      "max_value": "3650",
      "value": "",
      "min_value": "0",
      "is_required": false,
      "regex": "",
      "type": "int",
      "name": "days_keep_results"
    }
  },
  "version" : "1.0.0.0",
  "buid_date_time" : "",
  "service_type" : "operation",
  "service_impl" : "default",
  "django_app": "operation_default"
}
```

The default handler runs the pre-script, script, post-script, and undo script. The Zeron UI shows the scripts to gather parameters needed with each operation.

7.4 Developer Roles in Zeron

Development teams involved in the creation of solutions that need to run on multiple cloud environments, such as OpenStack, PureApplication System products, Amazon Web Services (AWS), Google Cloud, and others, have to focus on different areas. That is best done by having developers in specialized roles, as described in the following list:

- ▶ **Middleware Content Developer**

A software developer who can create Python-based scripts for installation and configuration; to run and apply fixes, patches, and upgrades; and for operations. Knows how best to install, configure, and deploy middleware in an enterprise-ready manner.

- ▶ **Application Developer**

A software developer who builds applications that run on middleware. When the content is developed, the developer passes it to a Pattern Builder Developer to package it for cloud deployments. Knows about middleware and applications that run on it, but does not necessarily need to know about Zeron or OpenStack.

- ▶ **Pattern Builder Developer**

A software developer who takes content scripts and creates custom Heat resources, such as SoftwareConfig or SoftwareDeployment based on OpenStack. Builds content for Zeron handlers, which this pattern needs to use. Uses content developed by the middleware content developer and application developer.

- ▶ **Pattern Deployer-Maintainer**

Deploys the Heat YAML on OpenStack. Uses the Zeron UI to manage the pattern instance. Maintains the running patterns by using the Zeron Administration Console to maintain the pattern.

- ▶ **Handler Developer**

A software developer who creates a custom handler for one or more services, such as Logging, Monitoring, Patches/iFixes, Upgrades, License Tracking, and Operations (Service Types). This optional role is for teams who want to create new implementations of the handlers.

Handler Developers, for example, can build and package the handler as a `.zip` or `.tgz` file, which then gets installed on each Zeron node. The `handler.json` contains, at a minimum, the name, version, `django-app`, build info, configuration, and enabled flag.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed description of the topics covered in this paper.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *Establishing a Secure Hybrid Cloud with the IBM PureApplication Family*, SG24-8284
- ▶ *Integrating IBM PureApplication System into an Existing Data Center*, SG24-8285
- ▶ *IBM Bluemix The Cloud Platform for Creating and Delivering Applications*, REDP-5242

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- ▶ Ansible:
<https://www.ansible.com>
- ▶ Chef:
<https://www.chef.io>
- ▶ *Discover PureApplication System Patterns of Expertise*:
<http://ibm.biz/PatternsOfExpertise>
- ▶ Docker is the dominant container technology in the market:
<https://www.docker.com>
- ▶ Git:
<https://git-scm.com/>
- ▶ Hypervisor:
<https://en.wikipedia.org/wiki/Hypervisor>
- ▶ IBM Blue Box:
<https://www.blueboxcloud.com/>
- ▶ IBM SoftLayer datacenter:
<http://www.softlayer.com>
- ▶ KVM:
<http://ibm.biz/Kernel-basedVM>

- ▶ nginx web server:
<https://nginx.org/en/>
- ▶ openSUSE Linux:
<https://www.opensuse.org>
- ▶ Open Container Initiative:
<https://www.opencontainers.org>
- ▶ OpenStack infrastructure:
<https://wiki.openstack.org/wiki/Heat>
- ▶ OpenStack Keystone:
<http://keystone.openstack.org>
- ▶ Puppet:
<https://puppet.com>
- ▶ Red Hat Linux:
<https://www.redhat.com>
- ▶ Rkt:
<https://coreos.com/rkt/>
<https://golang.org/>
- ▶ SaltStack:
<https://saltstack.com>
- ▶ Vmware ESXi:
<http://www.vmware.com/>
- ▶ Ubuntu Linux:
<http://www.ubuntu.com>
- ▶ YAML format:
<http://www.yaml.org/start.html>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



REDP-5352-00

ISBN 0738455725

Printed in U.S.A.

Get connected

