Cloud orchestration technologies: Explore your options
Learn the basics, and compare and contrast the various tools available

Suchitra Venugopal
July 11, 2016

Take an in-depth look at the most popular cloud orchestration technologies for developing and deploying applications. This article provides an excellent starting point for understanding essential cloud orchestration concepts, and compares and contrasts the various orchestration mechanisms at a high level. If you're thinking about implementing a cloud orchestration solution, here's your chance to explore the pros and cons of each option.

There's a lot of buzz in the IT industry lately around cloud orchestration, and many of you want to know what all the fuss is about. This article explores what cloud orchestration is and how it fits into the larger evolution of cloud computing. I'll take a look at various orchestration tools and examine the differences between them, so you can better understand what's available to you.

First, it's important to get some background. Before the emergence of virtualization and cloud computing, all processes were performed manually.

As you know, the three most common models of cloud services are Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Figure 1 illustrates how IaaS and virtualization are closely related (in fact, virtualization is a part of IaaS). As you can see, IaaS provides the infrastructure for cloud services to provision servers, storage, networks, OSes, and more; this is similar to hardware virtualization, which provides virtual machines in a data center.
The manual approach to setting up an environment included steps like these:

- Wait for approval
- Buy the hardware
- Install the OS
- Connect to and configure the network
- Get an IP
- Allocate the storage
- Configure the security
- Deploy the database
- Connect to a back-end system
- Deploy the application on the server

Challenges to this approach included management agents like backup and monitoring, networking, and configuring.

About a decade ago, virtualization emerged and did away with many of the manual steps. Hypervisors and VMs were used to deploy applications, which helped reduce hardware costs. However, managing the hypervisors could be challenging, so hypervisor manager programs were introduced to help eliminate many of the manual steps like buying the hardware, installing the OS, and allocating the storage.

Not long after that, cloud computing emerged and helped resolve many of the problems faced in the previous approach. Almost all of the steps in the manual approach became automated, making life much easier for infrastructure folks. Cloud computing allowed organizations to access software from anywhere as services. This helped reduce labor expenses, which are typically higher than hardware costs. Cloud is always available, with zero downtime, and everything can be done quickly. No need to wait for approvals—this truly was a do-it-yourself approach.

**Cloud orchestration**

**Cloud orchestration** is the end-to-end automation of the deployment of services in a cloud environment. More specifically, it is the automated arrangement, coordination, and management of complex computer systems, middleware, and services—all of which helps to accelerate the delivery of IT services while reducing costs. It is used to manage cloud infrastructure, which
supplies and assigns required cloud resources to the customer like the creation of VMs, allocation of storage capacity, management of network resources, and granting access to cloud software. By using the appropriate orchestration mechanisms, users can deploy and start using services on servers or on any cloud platforms.

There are three aspects to cloud orchestration:

- **Resource orchestration**, where resources are allocated
- **Workload orchestration**, where workloads are shared between the resources
- **Service orchestration**, where services are deployed on servers or cloud environments

Figure 2 illustrates how cloud orchestration automates the services in all types of clouds—public, private, and hybrid.

**Figure 2. Cloud orchestration**

Many people think that orchestration and automation are the same thing, but orchestration is actually more complex. Automation usually focuses on a single task, while orchestration deals with the end-to-end process, including management of all related services, taking care of high availability (HA), post deployment, failure recovery, scaling, and more. Automation is usually discussed in the context of specific tasks, whereas orchestration refers to the automation of processes and workflows. Basically, orchestration *automates the automation*—specifically, the order that tasks take place across specific machines, especially where there are diverse dependencies.

**Why should you choose orchestration?**

As you saw in the previous section, the manual process of setting up an environment involves multiple steps. Using an orchestrator tool, it's easy to quickly configure, provision, deploy, and
develop environments, integrate service management, monitoring, backup, and security services—and all of these steps are repeatable.

Another advantage to orchestration is that it enables you to make your products available on a wider variety of cloud environments, which enables users to deploy them more easily. As a result, you can increase your products' exposure to a wider audience, and potentially expand revenue opportunities for your company.

**Orchestration tools**

Various tools and techniques are available for orchestration, each of which are appropriate for specific cases, as you'll see in the following sections.

**Chef and Puppet**

**Chef** is a powerful automation platform that transforms complex infrastructures into code, bringing both servers and services to life. Chef automates configuration, deployment, and management of applications across the network.

Chef uses **cookbooks** to determine how each node should be configured. Cookbooks consist of multiple **recipes**; a recipe is an automation script for a particular service that's written using the Ruby language. The **Chef client** is an agent that runs on a node and performs the actual tasks that configure it. Chef can manage anything that can run the Chef client, like physical machines, virtual machines, containers, or cloud-based instances. The **Chef server** is the central repository for all configuration data. Both the Chef client and Chef server communicate in a secure manner using a combination of public and private keys, which ensure that the Chef server responds only to requests made by the chef-client. There is also an option to install standalone client called **chef-solo**.

**Puppet** is similar to Chef. It requires installation of a master server and client agent in target nodes, and includes an option for a standalone client (equivalent to chef-solo). You can download and install deployment modules using Puppet commands. Like Chef, Puppet comes with a paid Enterprise edition that provides additional features like reporting and orchestration/push deployment.

However, while both Chef and Puppet perform the same basic functions, they differ in their approach. Chef seems to be significantly more integrated and monolithic, whereas Puppet consists of multiple services. This can make Chef somewhat easier to get up and running and manage. Both have their pros and cons, so you will need to evaluate which one makes the most sense for your operations teams and infrastructure development workflow.

Here's how the two platforms stack up against one another:

- Puppet is geared toward system admins who need to specify configurations like dependencies, whereas Chef is for developers who actually write the code for the deployment.
• Puppet depends much more on its own Domain Specific Language (DSL) for defining the rules of configuration, whereas Chef's DSL is just a supplement of Ruby, so most of Chef's recipes are written in standard Ruby code.
• Chef has an omnibus (third-party) installer, which can make installation much easier than that of Puppet.
• Chef is used mostly for OS-level automation, like deployment of servers, patches, and fix issues. Puppet is mostly for mid-level automation, like installing databases and starting Apache.
• Chef seems to cater more toward developer-centric operations teams, while Puppet is geared toward more traditional operations teams with less Ruby programming experience.
• Once you get past its steep initial learning curve, Chef offers a lot more power and flexibility than Puppet.

OpenStack

OpenStack is a free, open source cloud computing software platform that's primarily used as an Infrastructure as a service (IaaS) solution. It consists of a series of interrelated projects that control pools of processing, storage, and networking resources throughout a data center; users manage all of them through a web-based dashboard, command-line tools, or a RESTful API. OpenStack.org released the platform under the terms of the Apache license.

Figure 3. OpenStack components

OpenStack began in 2010 as a joint project of Rackspace Hosting and NASA. Currently, it is managed by the OpenStack Foundation, a non-profit corporate entity that promotes OpenStack software and its community. More than 200 companies have joined the project, including IBM, Canonical, Cisco, Dell, EMC, Ericsson, Hewlett-Packard, Huawei, Intel, Mellanox, Mirantis, Oracle, Red Hat, SUSE Linux, VMware, and Yahoo, to name just a few!

The main components of OpenStack include Nova (compute), Cinder (block storage), Glance (image library), Swift (object storage), Neutron (network), Keystone (identity), and Heat (orchestration tool). Although this article will not describe all of these components in detail, we would like to focus on one particularly noteworthy component: Heat.

Heat

Heat is a pattern-based orchestration mechanism from OpenStack that's known as the Orchestration for OpenStack on OpenStack (OOO) project. Heat provides a template-based
orchestration for describing a cloud application by executing appropriate OpenStack API calls that generate running cloud applications. The software integrates other core components of OpenStack into a one-file template system. The templates allow for the creation of most OpenStack resource types (such as instances, floating IPs, volumes, security groups, and users) as well as more advanced functionality such as instance high availability, instance autoscaling, and nested stacks.

**How Heat works**

You can use Heat instead of writing a script that manages all of the software in OpenStack (like setting up the servers, adding volumes, managing networks, etc.). To do this, you create a Heat template that specifies what infrastructure you will need. If any further changes to the existing service are required later on, you can just modify the Heat template and the Heat engine will make the necessary changes when you rerun the template. When it's finished, you can clean up and release the resources, and they can be used by anyone else who needs them.

**Figure 4. How Heat works**

As you can see in Figure 4, passing a Heat template through the Heat engine creates a stack of resources that are specified in the Heat template.

Heat sits on top of all the other OpenStack services in the orchestration layer and talks to the IPs of all the other components. A Heat template generates a **stack**, which is the fundamental unit of currency in Heat. You write a Heat template with a number of resources in it, and each resource is an object in OpenStack with an object ID. Heat creates those objects and keeps track of their IDs.

You can also use a **nested stack**, which is a resource in a Heat stack that points to another Heat stack. This is like a tree of stacks, where the objects are related and their relationships can be inferred from the Heat template. This nested feature enables independent teams to work on Heat stacks and later merge them together.

The main component of Heat is the Heat engine, which provides the orchestration functionality.
HOT specification

Heat Orchestration Templates (HOT) are native to Heat and are expressed in YAML. These templates consist of:

- **Resources** (mandatory fields) are the OpenStack objects that you need to create, like server, volume, object storage, and network resources. These fields are required in HOT templates.
- **Parameters** (optional) denote the properties of the resources. Declaring the parameters can be more convenient than hard coding the values.
- **Output** (optional) denotes the output created after running the Heat template, such as the IP address of the server.

Each resource, in turn, consists of:

- References—used to create nested stacks
- Properties—input values for the resource
- Attributes—output values for the resource

Juju

Juju is an open source automatic service orchestration management tool developed by Canonical, the developers of the Ubuntu OS. It enables you to deploy, manage, and scale software and services on a wide variety of cloud services and servers. Juju can significantly reduce the workload for deploying and configuring a product's services.

Figure 5. Services deployed using Juju

![Diagram of services deployed using Juju](image-url)
**Benefits**

Juju is the fastest way to model and deploy applications or solutions on all major public clouds and containers. It helps to reduce deployment time from days to minutes. Juju works with existing configuration management tools, and can scale workloads up or down very easily. No prior knowledge of the application stack is needed to deploy a Juju charm for the product. Juju includes providers for all major public clouds, such as Amazon Web Services, Azure, and HP as well as OpenStack, MAAS, and LXC containers. Juju can also be deployed on IBM SoftLayer using the manual provider available in Juju, so anyone can use Juju with SoftLayer by provisioning the machines manually and then telling Juju where those machines are. Local providers on LXC containers allow you to recreate the production deployment-like environment on your own laptop. It also offers a quick and easy environment for testing deployments on a local machine. Users can deploy entire cloud environments in seconds using bundles, which can save a lot of time and effort.

**Charms**

Juju utilizes charms, which are open source tools that simplify specific deployment and management tasks. A charm is a set of scripts that can be written in any language that fire off hooks based on certain things. After a service is deployed, Juju can define relationships between services and expose some services to the outside world. Charms give Juju its power. They encapsulate application configurations, define how services are deployed, how they connect to other services, and how they are scaled. Charms define how services integrate, and how their service units react to events in the distributed environment, as orchestrated by Juju.

A Juju charm usually includes all of the intelligence needed to scale a service horizontally by adding machines to the cluster, preserving relationships with all of the services that depend on that service. This enables you to build out (and scale up and down) the service you want, especially on the cloud. Juju provides both a command-line interface and an intuitive web application for designing, building, configuring, deploying, and managing your infrastructure. Juju automates the mundane tasks, allowing you to focus on creating amazing applications.

Charms are easy to share, and there are hundreds of charms already rated and reviewed in the Juju charm store.

**Relationships and other features**

Juju allows services to be instantly integrated via relationships. Relationships allow the complexity of integrating services to be abstracted from the user. Juju relationships are loosely typed definitions of how services should interact with one another. These definitions are handled through an interface. Juju decides which services can be related based solely on the interface names.

Some of the advanced features in Juju include:

- **Juju Compose** builds new charms from existing ones using a layering approach, so that common tasks require much less rework. Features in the lower layers are inherited by the new charms.
- **Subordinate charms** are related charms that can be grouped together as subordinate or principal charms. A principal charm is the main charm and a subordinate charm cannot stand alone, so it is deployed along with the principal charm.
- **Leadership hooks** are automated mechanisms provided by Juju that select a leader/master in a clustered environment.

**Charm store**
Juju includes a collection of charms that let you deploy whatever services you like in Juju. A **bundle** is a collection of charms that are designed to work together. Since charms and bundles are open and worked on by the community, they represent a set of best practices for deploying these services. Both charms and bundles are included in what we collectively call the **charm store**.

To have your charm listed as a charmer team recommended charm, you have to undergo a rigorous review process where the team evaluates the charm, tests it, and deploys and runs tests against the provided service with different configuration patterns.

**Docker**
**Docker** is an open platform for developing, shipping, running, and delivering applications quickly. With Docker, the user can separate your applications from your infrastructure and treat your infrastructure like a managed application. Docker helps you ship code faster, test faster, deploy faster, and reduce the time between writing code and running code.

Docker does this by combining a lightweight container virtualization platform with workflows and tooling that help manage and deploy applications. At its core, Docker provides a way to run almost any application securely isolated in a container. This allows you to run many containers simultaneously on your host. The lightweight nature of containers, which run without the extra load of a hypervisor, means you can get more out of your hardware.

Docker containers wrap a piece of software in a complete file system that contains everything it needs to run: code, runtime, system tools, system libraries—anything you can install on a server. This guarantees that it will always run the same, regardless of the environment where it is running.

Docker is:

- **Lightweight**—containers running on a single machine all share the same operating system kernel so they start instantly and make more efficient use of RAM. Images are constructed from layered file systems so they can share common files, which makes disk usage and image downloads much more efficient.
- **Open**—Docker containers are based on open standards. This allows them to run on all major Linux distributions and Microsoft operating systems with support for every infrastructure.
- **Secure**—containers isolate applications from each other and the underlying infrastructure while providing an added layer of protection for the application.

**Containers vs. virtual machines**
Each virtual machine includes the application, the necessary binaries and libraries, and an entire guest operating system—all of which may be tens of gigabytes in size. Containers include the
application and all of its dependencies, but share the kernel with other containers. They run as an isolated process in userspace on the host operating system. They're also not tied to any specific infrastructure: Docker containers run on any computer, on any infrastructure, and in any cloud. The key difference between containers and VMs is that while the hypervisor abstracts an entire device, containers just abstract the operating system kernel. This means that one thing hypervisors can do that containers can’t is use different operating systems or kernels.

**Figure 6. Containers vs. virtual machines**

Benefits

Docker containers provide a range of benefits, including:

- **Faster delivery of applications**—Docker is perfect for helping with the development lifecycle. It also allows you to develop on local containers that contain the apps and services.
- **Deploy and scale more easily**—Docker's container-based platform allows for highly portable workloads. They can run on a developer's local host, on physical or virtual machines in a data center, or in the cloud. You can use Docker to quickly scale apps and services up or down.
- **Achieve higher density and run more workloads**—Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based VMs. This is especially useful in high-density environments, such as building your own cloud or Platform as a Service. However, it is also useful for small- and medium-sized deployments where you want to get more out of the resources you have.
- **Eliminate environmental inconsistencies**—By packaging the application with its configs and dependencies together and shipping it as a container, the app will always work as designed locally, on another machine.
- **Empower developer creativity**—The isolation capabilities of Docker containers liberate developers from having to use approved language stacks and tooling. Developers can use the best language and tools for their application service without having to worry about causing conflicts.
• **Accelerate developer onboarding**—Stop wasting hours trying to set up developer environments, spin up new instances, and make copies of production code to run locally. With Docker, you can easily take copies of your live environment and run them on any new endpoint that’s running Docker.

**Dockerfiles**

A Dockerfile is a text document that contains all the commands a user can call on the command line to assemble an image. Using `docker build`, you can create an automated build that executes several command-line instructions in succession. Docker can build images automatically by reading the instructions from a Dockerfile.

**Docker Hub**

Docker Hub is a cloud-hosted service from Docker that provides registry capabilities for public and private content. It makes it easier for you to collaborate with the broader Docker community or with your own team on key content, or automate your application by building workflows.

**Comparing cloud orchestration tools**

Table 1 gives a side-by-side comparison of the various tools covered in this article:

**Table 1. Comparison of cloud orchestration tools**

<table>
<thead>
<tr>
<th>Chef</th>
<th>Puppet</th>
<th>Heat</th>
<th>Juju</th>
<th>Docker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mainly used for the automation of deployments. At first, this was used more on OS level for things like deployment of servers, patches, and fixes. Later on, it was used for things like installing middleware.</td>
<td>Originally used at the middleware level for things like installing databases and starting Apache. It explored everything with APIs. In time, its use expanded to installing at the OS level, as well.</td>
<td>Orchestration mechanism from OpenStack.</td>
<td>Pattern-based service layer (automation only) for Ubuntu.</td>
<td>Used as both a virtualization technology and an orchestration tool.</td>
</tr>
<tr>
<td>Caters more toward developer-centric operations teams.</td>
<td>Caters to more traditional operations teams with less Ruby programming experience.</td>
<td>Orchestrates everything on OpenStack. Mostly for infrastructure. Heat uses Chef/Puppet for installation.</td>
<td>Works on all popular cloud platforms—Ubuntu local machines, bare metal, etc.</td>
<td>Open platform that enables developers and system administrators to build, ship, and run distributed applications.</td>
</tr>
<tr>
<td>Procedural where recipes are written in Ruby code, which is quite natural for developers. Chef's steep learning curve is often seen to be risky in large organizations, where it can be difficult to build and maintain skills in large teams.</td>
<td>Learning curve is less imposing because Puppet is primarily model driven.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Once you get through the initially steep learning curve, Chef can provide a lot more power and flexibility than other tools.</td>
<td>Puppet is a more mature product with a larger user base than that of Chef.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

This article has provided a high-level overview of the most popular cloud orchestration mechanisms to help you compare contrast the various options, and decide which one is best suited for your needs. I have drawn on my own experiences in learning these technologies, and encourage you to explore them in more depth as appropriate. The Related topics links below can help you with those explorations.
Related topics

- Chef
- Puppet
- OpenStack.org
- OpenStack for beginners
- Heat
- Heat for OpenStack orchestration
- Heat template guide
- Docker
- Juju
- Juju Charms store

© Copyright IBM Corporation 2016

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)