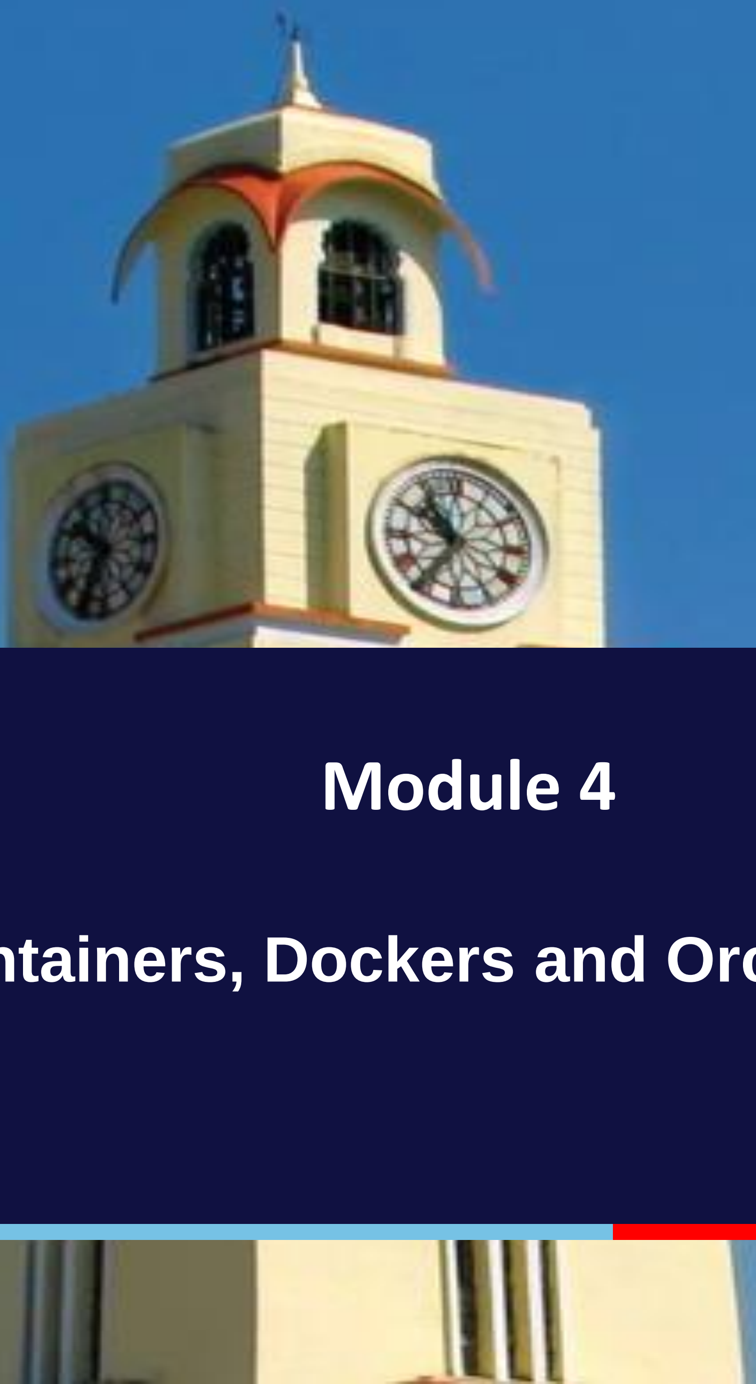
Cloud Computing



BITS Pilani

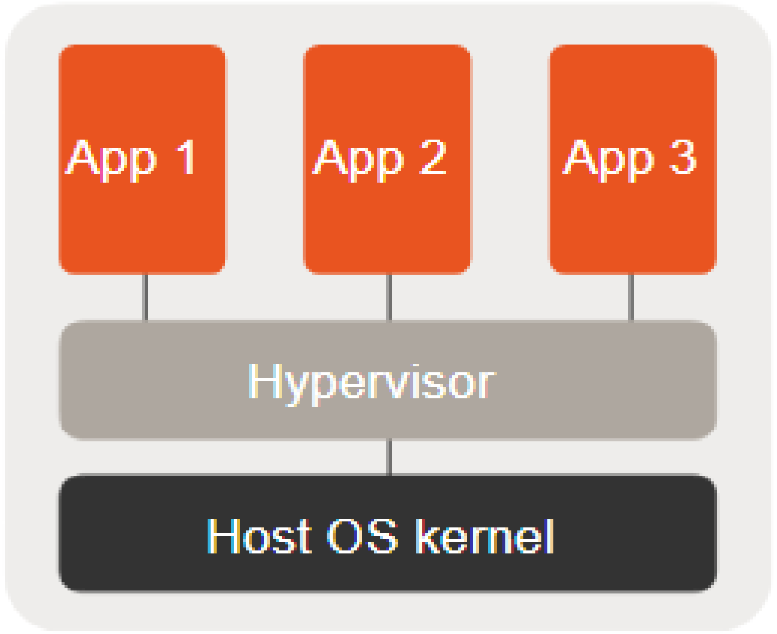
Module 4

(Linux containers, Dockers and Orchestration tools)

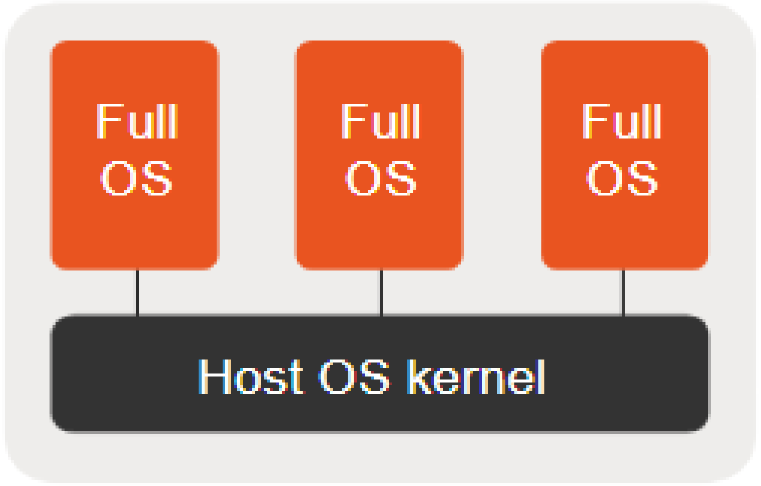
Agenda

- Linux Containers - LXC and LXD.
- Cloud orchestration technologies
- Dockers

Difference Between Application Containers and System Containers

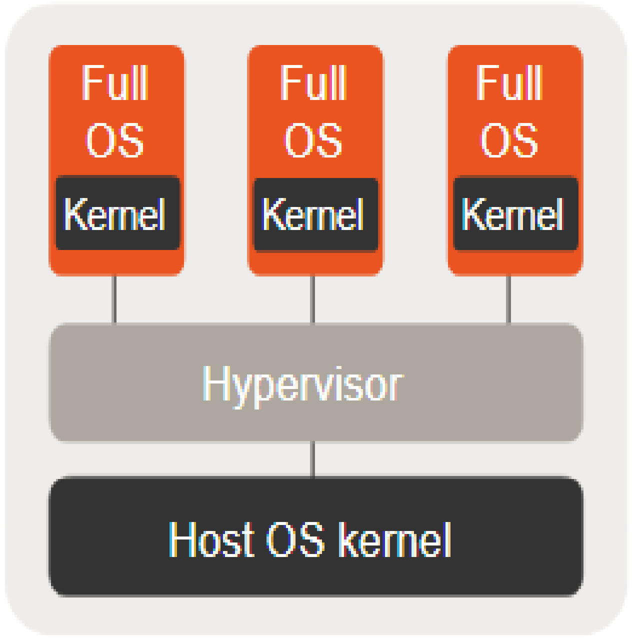


Application containers

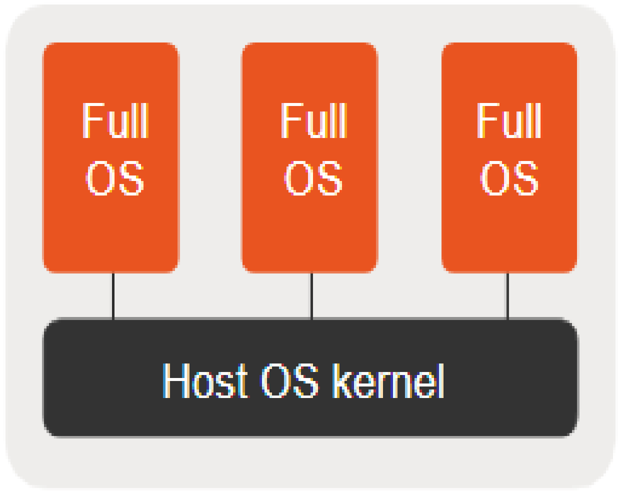


System containers

Difference Between System Containers and Virtual Machines



Virtual machines



System containers

LXC (Linux Containers)

- It is a system container.
- It is a set of 1 or more processes that are isolated from the rest of the system.
- It is used for Linux Containers which is an operating system that is used for running multiple Linux systems virtually on a controlled host via a single Linux kernel.
- It lets Linux users easily create and manage system or application containers.

Features provided by LXC :

- It provides Kernel namespaces such as IPC, mount, PID, network, and user.
- Apparmor and SELinux profiles
- Control groups (Cgroups).
- Seccomp policies
- Chroots (using pivot_root)

Goal: To create an environment as close as possible to a standard Linux installation but without the need for a separate kernel.

LXD (Linux container hypervisor)

- LXD is a next generation system container manager. It offers a user experience similar to virtual machines but using Linux containers instead.
- It is an extension of LXC with more functionality.
- It provides flexibility and scalability for various use cases, with support for different storage backends and network types and the option to install on hardware ranging from an individual laptop or cloud instance to a full server rack.
- It provides support for system containers and virtual machines.

Some Features of LXD are:

- Secure by design (unprivileged containers, resource restrictions and much more)
- Scalable (from containers on your laptop to thousand of compute nodes)
- Intuitive (simple, clear API and crisp command line experience)
- Image based (with a wide variety of Linux distributions published daily)
- Support for Cross-host container and image transfer (including live migration)
- Advanced resource control (cpu, memory, network I/O, block I/O, disk usage and kernel resources)
- Device passthrough (USB, GPU, unix character and block devices, NICs, disks and paths)

Installation and Launching Containers

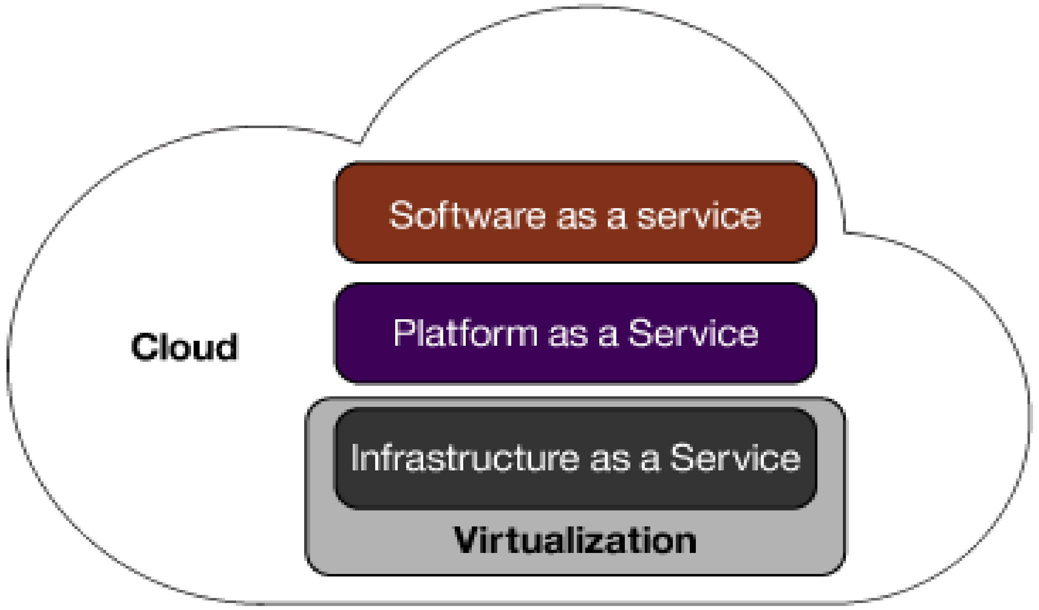
https://www.cyberciti.biz/faq/install-lxd-on-ubuntu-20-04-lts-using-apt/

1. lxc launch images:ubuntu/20.04 first
2. lxc info first
3. lxc stop first
4. lxc delete first
5. Lxc launch images:ubuntu/20.04 firstlimited –c limits.cpu=2 –limits.memory=200MiB
6. lxc execute firstlimited –bash
7. cat /etc/*release

Virtualization and cloud computing

The manual approach to setting up 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



What challenges we face with the manual approach.?
Sol: backup, monitoring, networking, and configuring.

Significance of Configuration Management

The problem Context:
Suppose we have to deploy a software on top of hundreds of systems.
This software can be an operating system or a code or it can be an update of an existing software.
You can do this task manually, but what happens if you have to finish this task overnight because tomorrow might be a **Big Billion Day** sale in the company or some **Mega Sale** etc. in which heavy traffic is expected.
Even if you were able to do this manually there is a high possibility of multiple errors on your big day.

What if the software you updated on hundreds of systems is not working, then how will you revert back to the previous stable version, will you be able to do this task manually.

To solve the above issues:
Configuration Management was introduced. Chef, Puppet, etc. are configuration management tools that can automate the automate above tasks.

How it automates the above tasks?

We need to specify the configurations once on the central server and replicate that on thousands of nodes.

Configuration Management

It helps in performing the below tasks in a very structured and easy way:

- Figuring out which components to change when requirements change.
- Redoing an implementation because the requirements have changed since the last implementation.
- Reverting to a previous version of the component if you have replaced with a new but flawed version.
- Replacing the wrong component because you couldn’t accurately determine which component was supposed to be replaced.

Cloud Orchestration Technologies

Why Orchestration technologies are required?.

- To quickly configure, provision, deploy, and develop environments, integrate service management, monitoring, backup, and security services. All these services are repeatable.

They are used to address key challenges IaaS providers face when building a cloud infrastructure: **managing physical and virtual resources, namely servers, storage, and networks, in a holistic fashion.**

The orchestration of resources must be performed in a way to rapidly and dynamically provision resources to Applications. The software toolkit responsible for this orchestration is called a **virtual infrastructure manager (VIM).**

Cloud Orchestration

What is Cloud orchestration?

- It is the end-to-end automation of the deployment of services in a cloud environment.
  - -Manage cloud infrastructure: 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**.
- Cloud Orchestration automates provisioning of multiple servers, storage, databases and networks to make deployment and management of processes and resources smoother.
- It also ensures the complete maintenance of cloud elements in an integrated and harmonized way.

Objective: To accelerate the delivery of IT services while reducing costs.

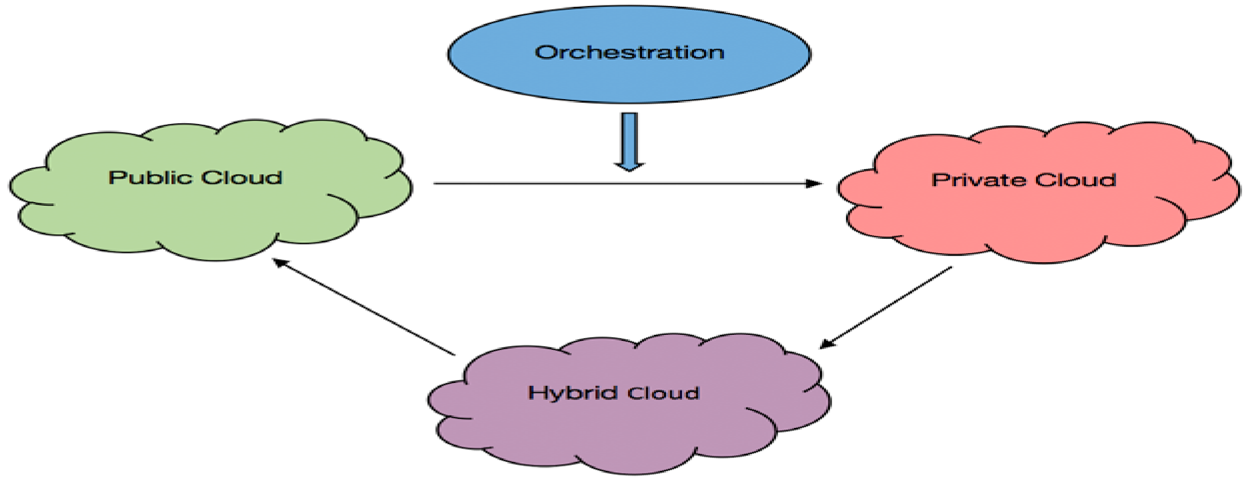
Benefits of Cloud Orchestration

- Effective Visibility and Control through dashboard
- Cost-Effective
- Reduced Errors

Three aspects of 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

The orchestration automates the services in all types of clouds—public, private, and hybrid.



Difference between orchestration and automation

- 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.**
- A single task is involved in cloud automation whereas in cloud orchestration It is concerned with combining multiple such tasks into workflows.
- **Some examples of cloud automation** are Launching a web server, configuring a web server and some cases of cloud orchestration are Combining automated tasks like **launching web server and configuring a web service into a single workflow to meet client requests**

Note:

Note: Orchestrating is a process of automating a series of individual tasks to work together.

Orchestration tools

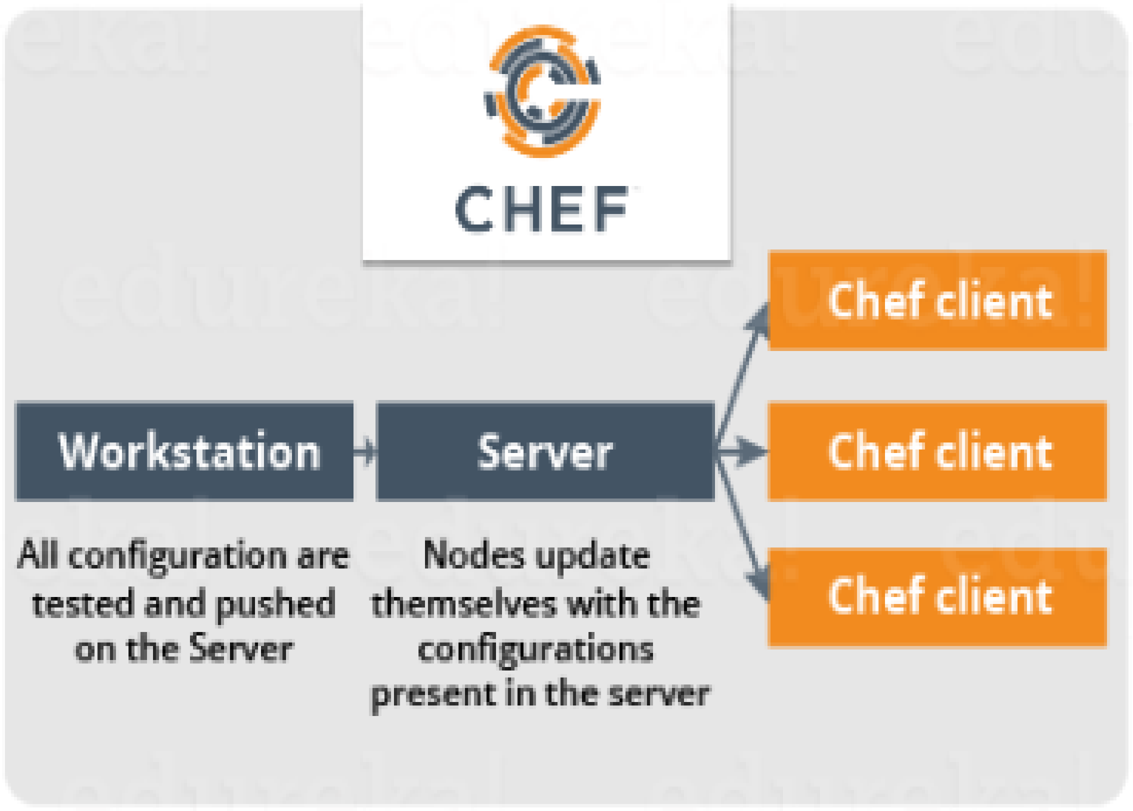
Cloud orchestration has useful tools that can automatically monitor cloud resources and modify VM instances while reducing manual work and work hours and they ensure efficient control on the cloud ecosystem.

1. Chef
2. Puppet
3. OpenStack (Already discussed)
4. Heat
5. Juju
6. Docker (Docker Swarm)

1. Chef

- It is used in infrastructure automation and allow us to automate and control vast numbers of computers in an automated, reliable, and scalable manner.
- Chef translates system administration tasks into reusable definitions, known as cookbooks and recipes.
- Cookbooks are the configuration units that allow us to configure and perform specific tasks within Chef on our remote nodes
- In a recipe, Chef authors define a system's desired state by writing configuration code. It specifies which resources to use and the order in which they are to be used.
- Chef then processes that code along with data about the specific node where the code is running to ensure that the desired state actually matches the state of the system.
- Chef uses a pure-Ruby, domain-specific language (DSL) for writing system configurations. It is developed on the basis of Ruby DSL language. It is used to streamline the task of configuration and managing the company's server.

Chef Architecture


CHEF
Workstation
Server
Chef client
Chef client
Chef client
All configuration are tested and pushed on the Server
Nodes update themselves with the configurations present in the server

Ref: https://www.edureka.co/blog/chef-tutorial/

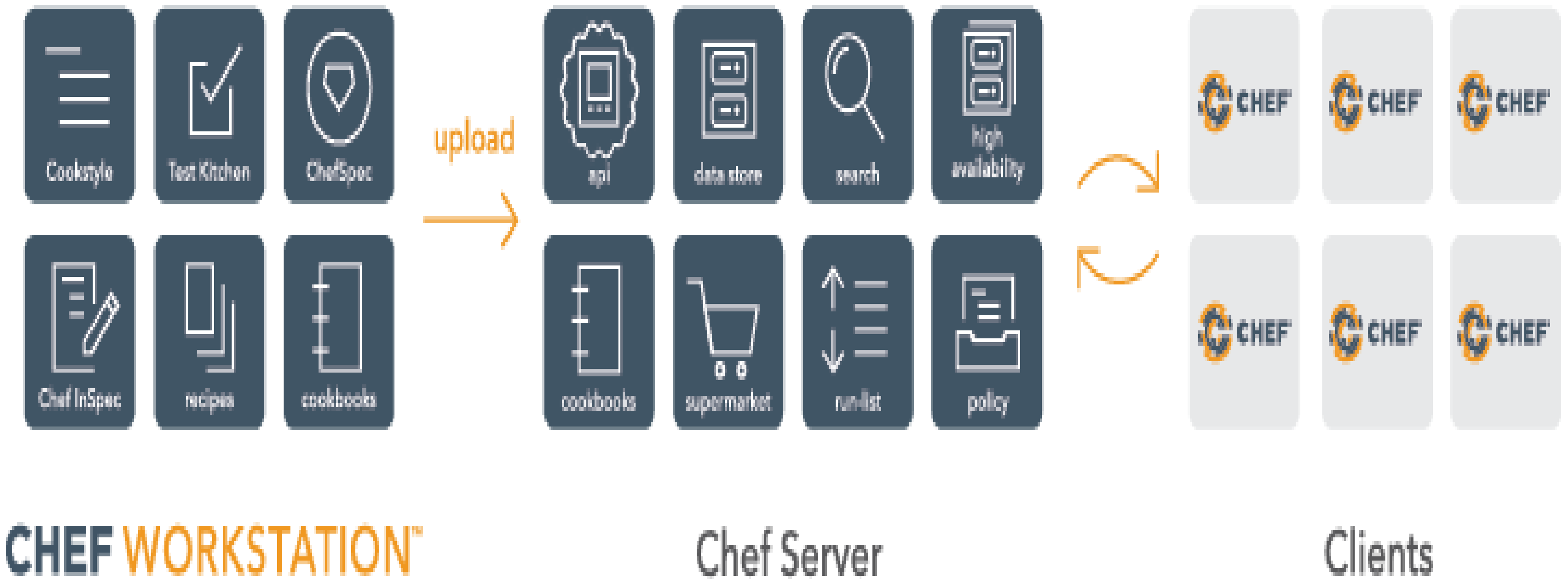
Functionality:
Chef allows us to dynamically provision and de-provision your infrastructure on demand to keep up with peaks in usage and traffic.

- It enables new services and features to be deployed and updated more frequently, with little risk of downtime.
- With Chef, we can take advantage of all the flexibility and cost savings that cloud offers.

Note: 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.

Chef Architecture Cont...



Chef Architecture Cont…

- **Pull Configuration:** Nodes will automatically update themselves with the config urations present in the

Server.

- Chef supports multiple platforms like AIX, RHEL/CentOS, FreeBSD, OS X, Solaris, Microsoft Windows and Ubuntu.
- Chef can be integrated with cloud-based platforms such as Amazon EC2, Google Cloud Platform, OpenStack, SoftLayer, Microsoft Azure and Rackspace to automatically provision and configure new machines.

Example demonstration of creating a chef book:
https://www.digitalocean.com/community/tutorials/how-to-create-simple-chef-cookbooks-to-manage-infrastructure-on-ubuntu

Chef cookbooks

- Chef uses **cookbooks** to determine how each node should be configured.
- Cookbooks are usually used to handle one specific service, application, or functionality. For instance, **a cook book can be created to set and sync the node's time with a specific server. It may install and configure a database application. Cookbooks are basically packages for infrastructure choices.**
- Cookbooks consist of multiple **recipes**; a recipe is an automation script for a particular service that's written using the Ruby language.

- Cookbooks are created on the workstation and then uploaded to a Chef server.
- From there, recipes and policies described within the cookbook can be assigned to nodes as part of the node's "run-list".
- A run-list is a sequential list of recipes and roles that are run on a node by chef-client in order to bring the node into compliance with the policy you set for it.

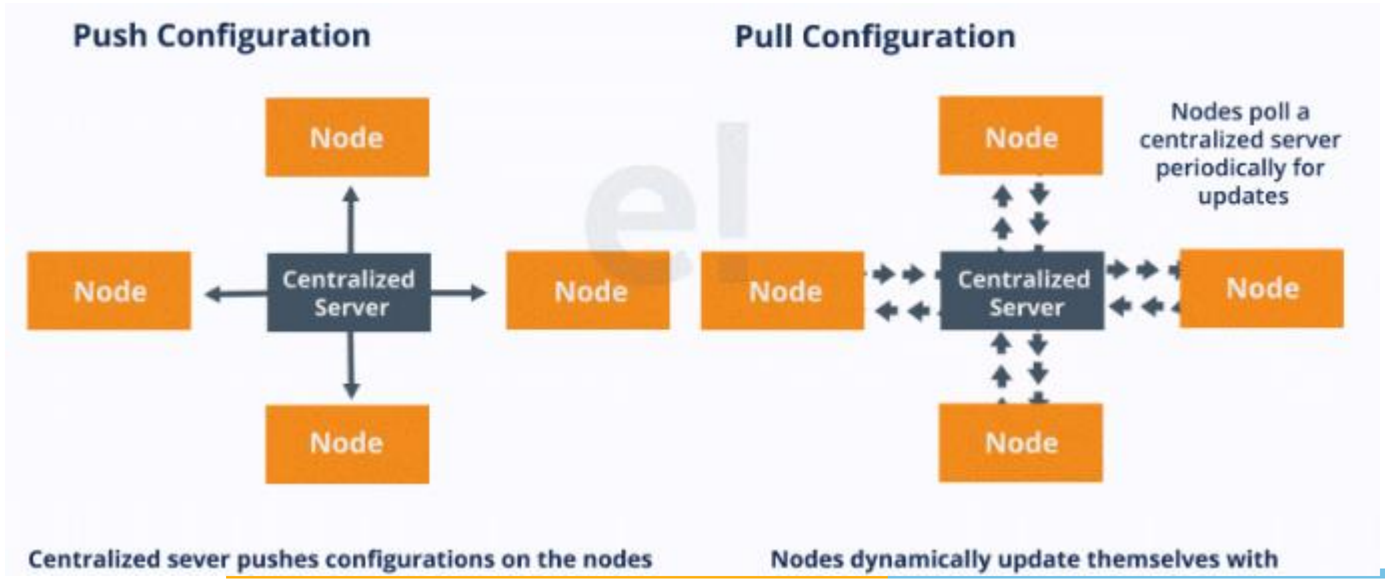
Note1: https://www.linode.com/docs/guides/beginners-guide-chef/ and https://docs.chef.io/chef_overview/

Note2: https://www.digitalocean.com/community/tutorials/how-to-create-simple-chef-cookbooks-to-manage-infrastructure-on-ubuntu

There are broadly two ways to manage your configurations namely Push and Pull configurations.

Pull Configuration: In this type of Configuration Management, the nodes poll a centralized server periodically for updates. These nodes are dynamically configured so basically they are pulling configurations from the centralized server.
Pull configuration is used by tools like Chef, Puppet etc.

Push Configuration: In this type of Configuration Management, the centralized Server pushes the configurations to the nodes. Unlike Pull Configuration, there are **certain commands that have to be executed in the cen tralized server** in order to configure the nodes.
Push configuration is used by tools like Ansible and Salt Stack


Push Configuration
Pull Configuration
Node
Node
Node
Node
Centralized Server
Node
Node
Node
Node
Centralized Server
Nodes poll a centralized server periodically for updates
Centralized sever pushes configurations on the nodes
Nodes dynamically update themselves with

Ref: https://www.edureka.co/blog/what-is-chef/

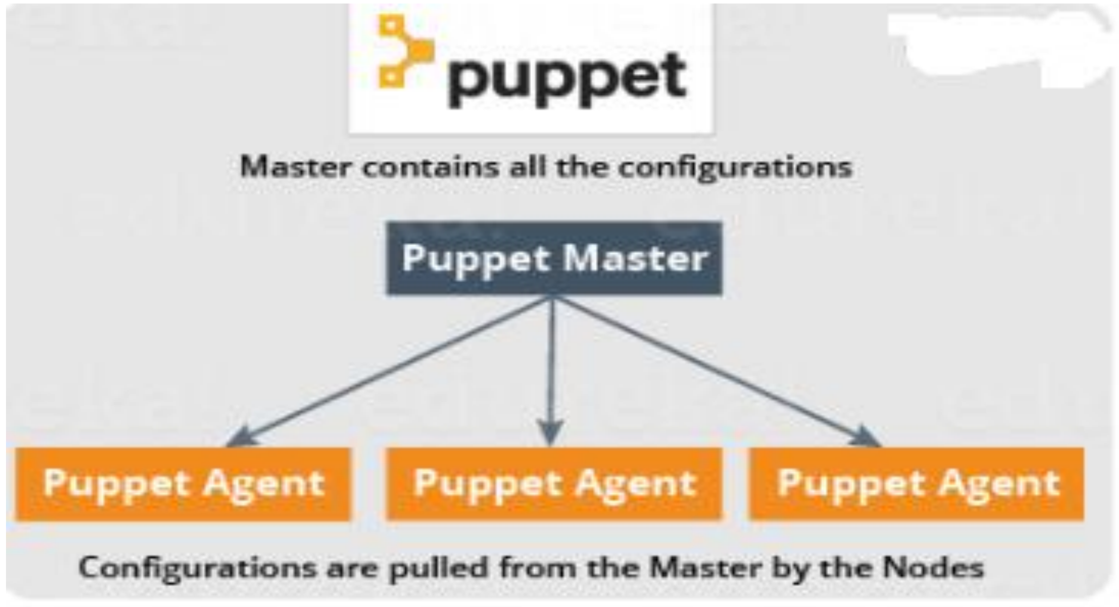
2. Puppet

What is Puppet?

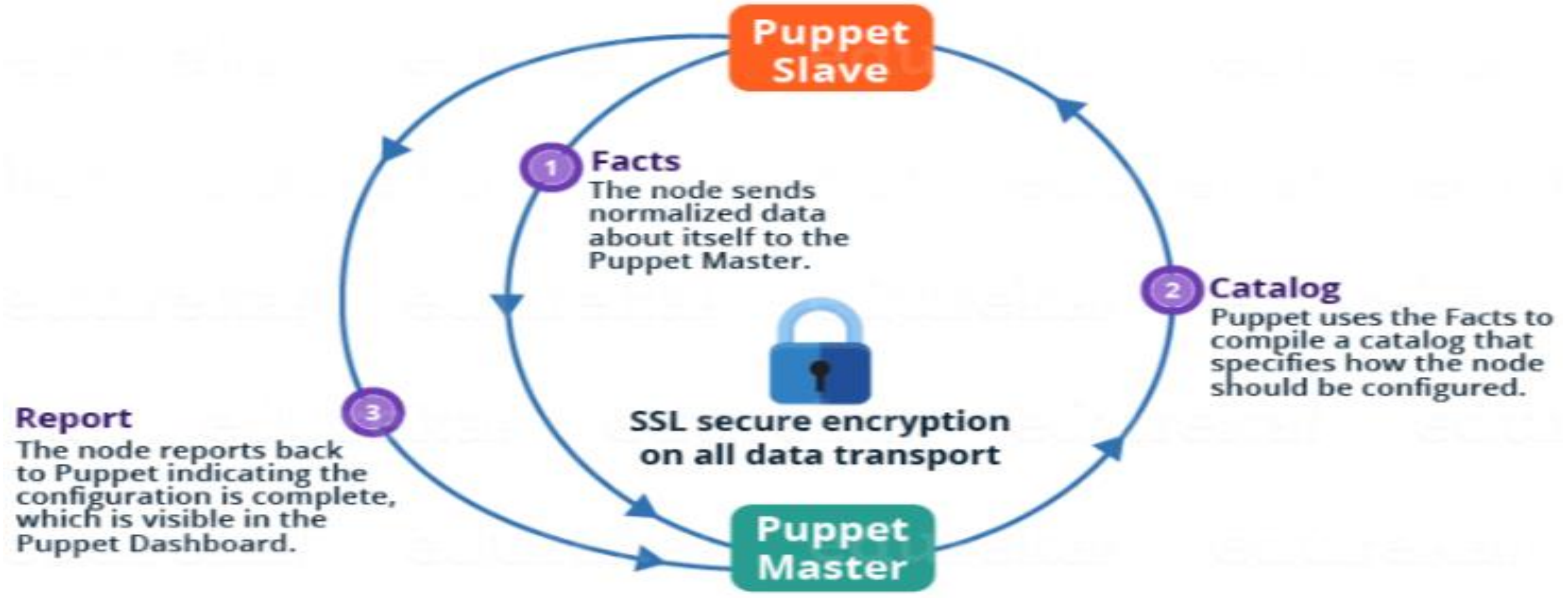
- Puppet is a Configuration Management tool and can be used to install and manage software on existing server instances (e.g., installation of packages, starting of services, installing scripts or config files on the instance).
- Puppet is also used as a software deployment tool. It is an open-source configuration management software widely used for server configuration, management, deployment, and orchestration of various applications and services across the whole infrastructure of an organization.
- Puppet is specially designed to manage the configuration of Linux and Windows systems.
- It is written in Ruby and uses its unique **D**omain **S**pecific **L**anguage (DSL) to describe system configuration.
- They do the heavy lifting of making one or many instances perform their roles without the user needing to specify the exact commands. No more manual configuration or ad-hoc scripts are needed.

Puppet (Cont…)

- It requires installation of a master server and client agent in target nodes, **and includes an option for a standalone client.**


puppet
Master contains all the configurations
Puppet Master
Puppet Agent
Puppet Agent
Puppet Agent
Configurations are pulled from the Master by the Nodes

Puppet


Puppet Slave
1 Facts
The node sends normalized data about itself to the Puppet Master.
2 Catalog
Puppet uses the Facts to compile a catalog that specifies how the node should be configured.
3 Report
The node reports back to Puppet indicating the configuration is complete, which is visible in the Puppet Dashboard.
SSL secure encryption on all data transport
Puppet Master

https://www.edureka.co/blog/puppet-tutorial/

Puppet Versions

Puppet comes in two versions:

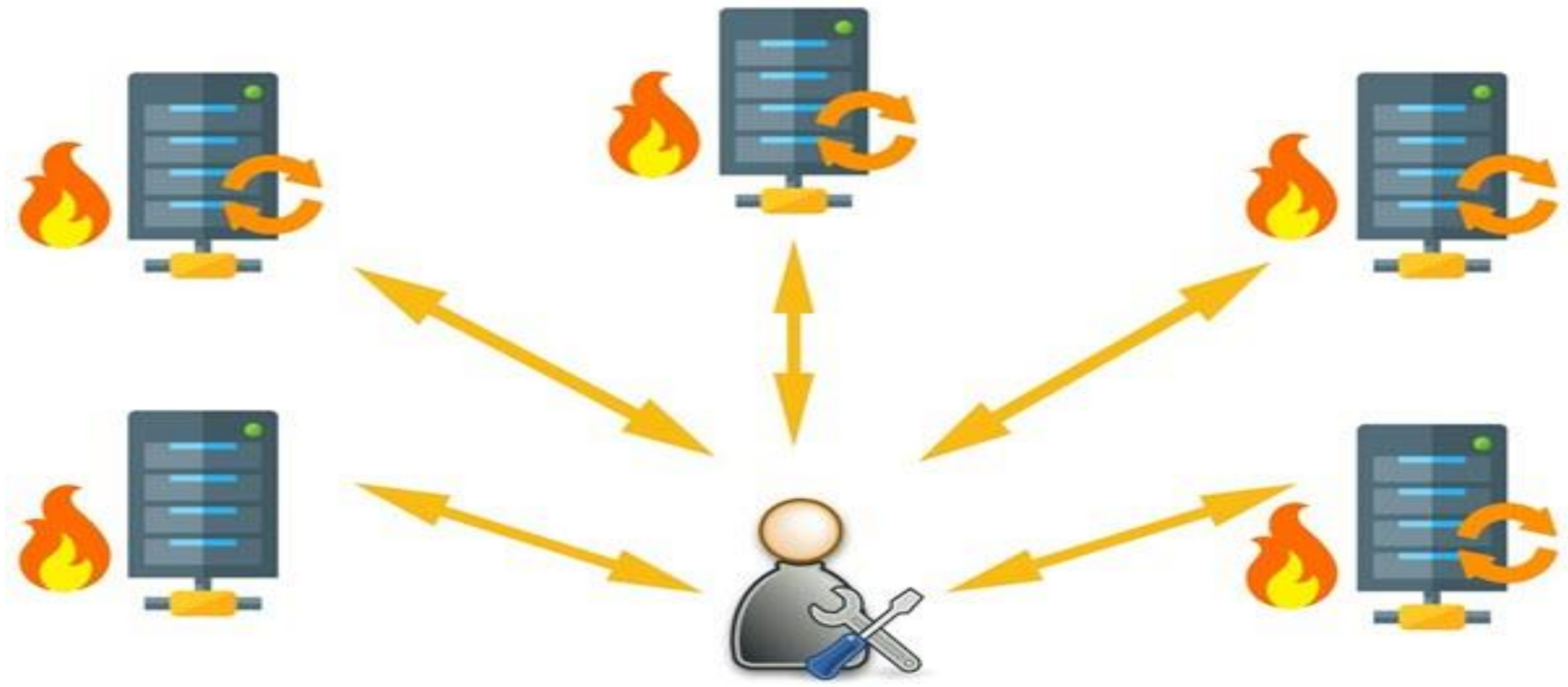
Open Source Puppet: It is a basic version of Puppet configuration management tool, which is also known as Open Source Puppet. It is available directly from Puppet's website and is licensed under the Apache 2.0 system.

Puppet Enterprise: Commercial version that offers features like compliance reporting, orchestration, role-based access control, GUI,API and command line tools for effective management of nodes.

Puppet Functionalities

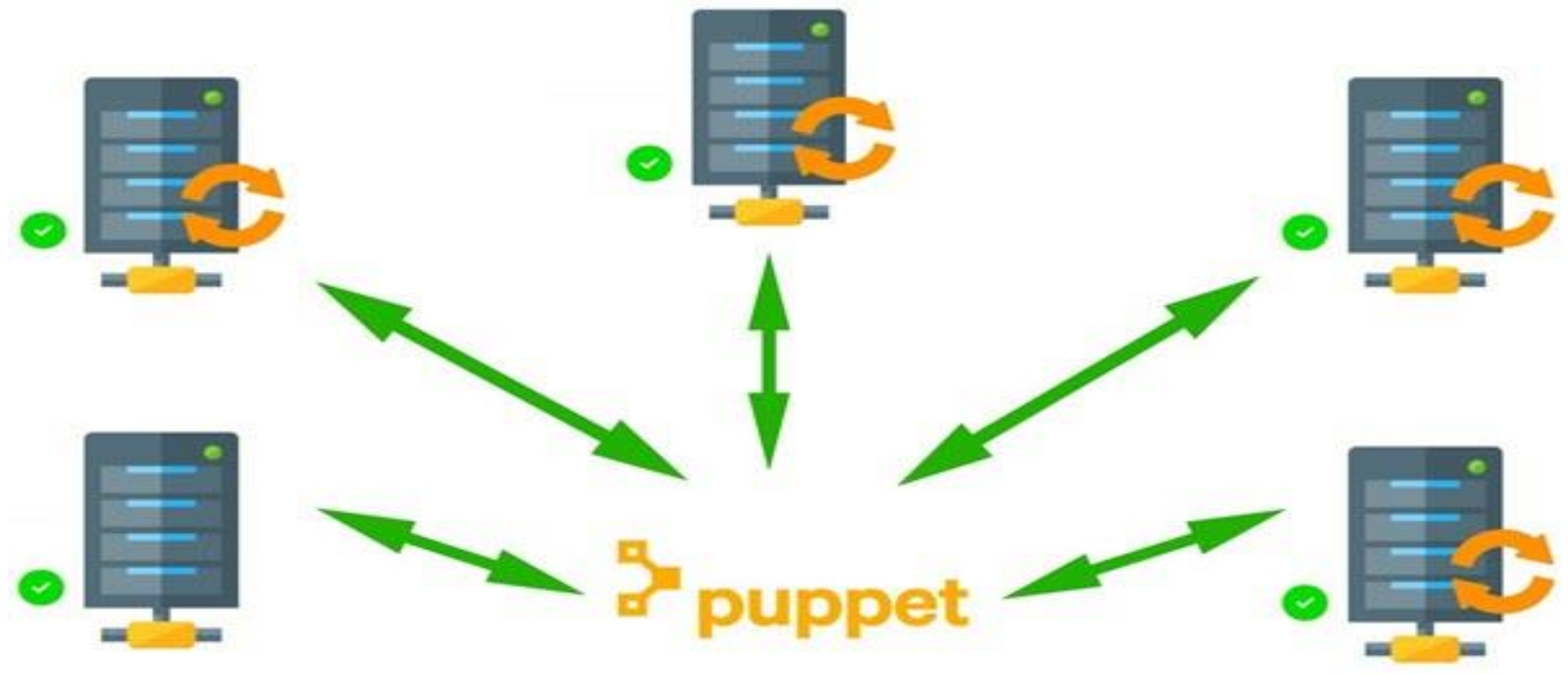
- **Defining distinct configurations** for each and every host, and continuously checking and confir ming whether the required configuration is in place or not.
- Dynamic **scaling-up and scaling-down** of machines.
- Providing **control over all your configured machines**, so a centralized (master-server) change gets propagated to all, automatically.

What Puppet Can do?.



Note: The role of **system admin** is to ensure that all these servers are always up to date and running with full functionality.

What Puppet Can do?. (Cont…)



Note: Puppet here allows you to write a simple code which can be deployed automatically on these servers. This reduces the human effort and makes the development process fast and effective.

How Puppet Works?

- Puppet is based on a **Pull deployment model,** where the agent nodes check in regularly after ever y **1800** seconds with the master node to see if anything needs to be updated in the agent.
- If anything needs to be updated the agent pulls the necessary puppet codes from the master and per forms required actions.

Example: Master - Agent Setup:

The Master:

A Linux based machine with Puppet master software installed on it. It is responsible for maintaining configurations in the form of puppet codes. The master node can only be Linux.

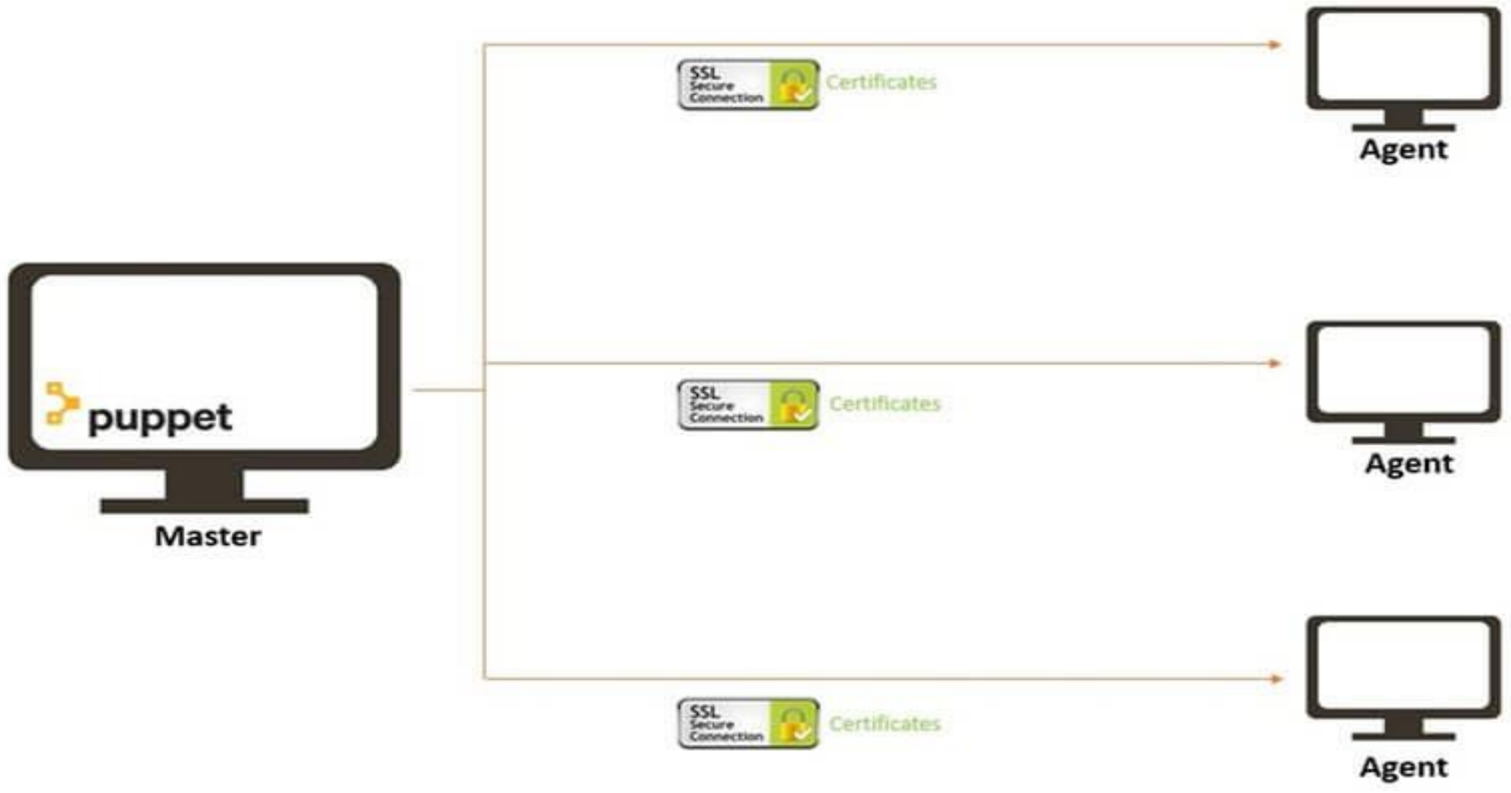
The Agents:

The target machines managed by a puppet with the puppet agent software installed on them.

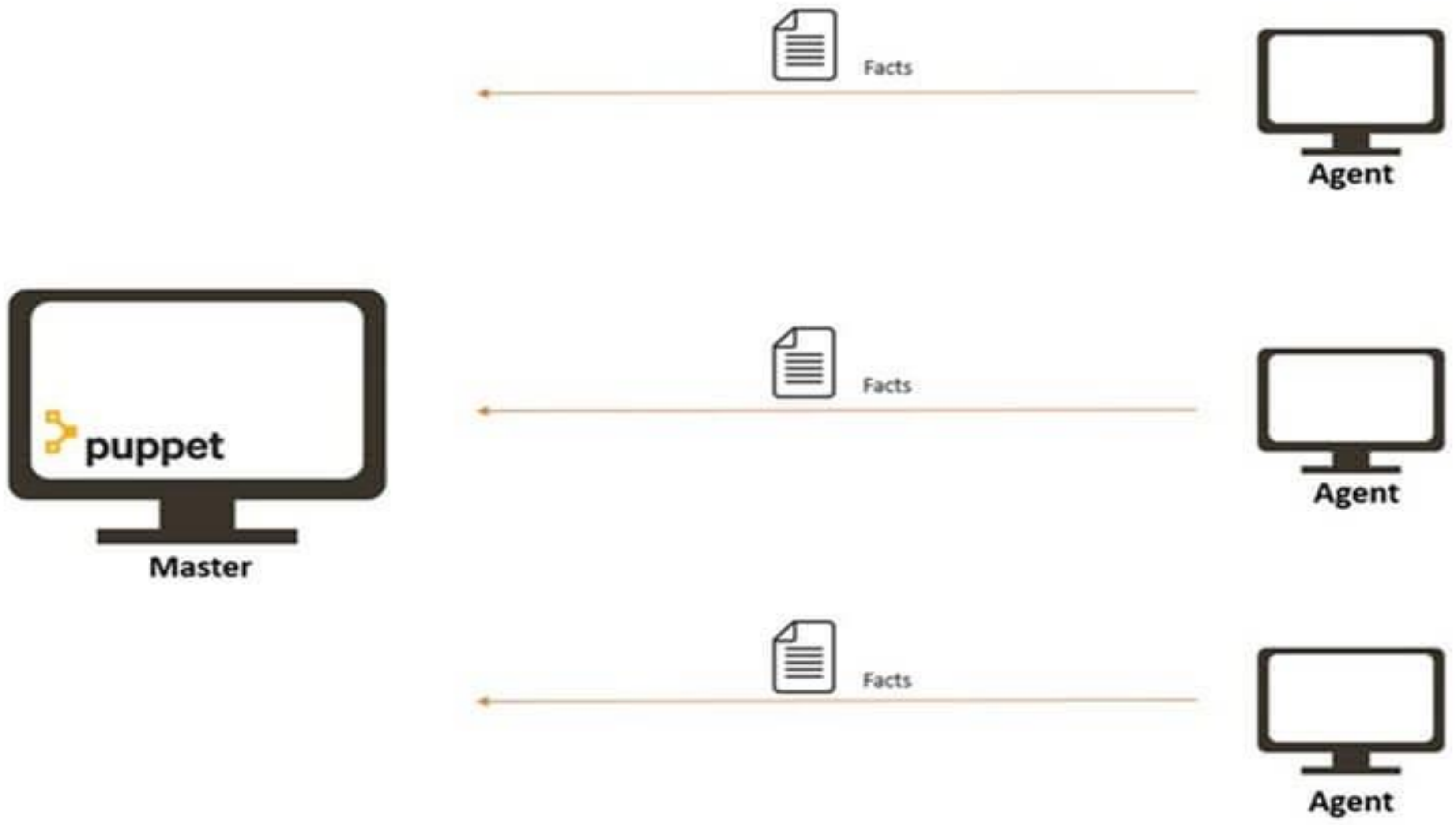
How Puppet Works? (Cont…)

- The agent can be configured on any supported operating system such as Linux or Windows or Solaris or Mac OS.
- The communication between master and agent is established through secure certificates.

1. Connection Establishment



How Puppet Works? (Cont…)

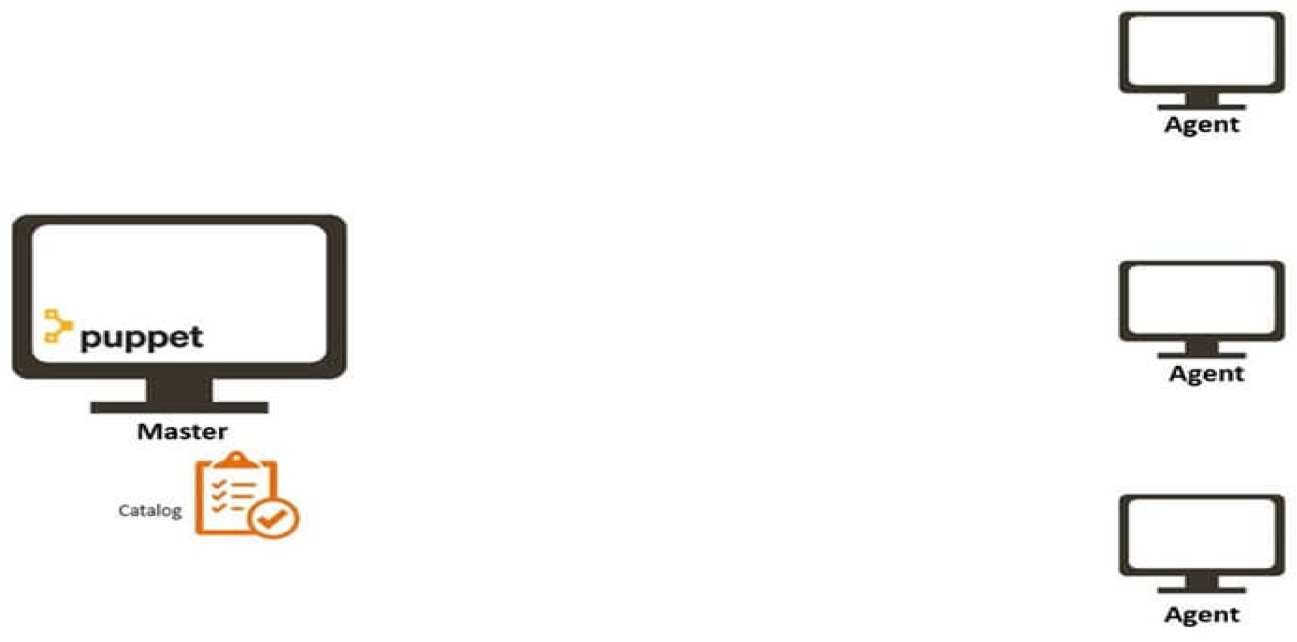


2. Agents send Facts to the Master

Facts contain information that includes the hostname, kernel details, IP address, file name details, etc.

How Puppet Works? (Cont…)

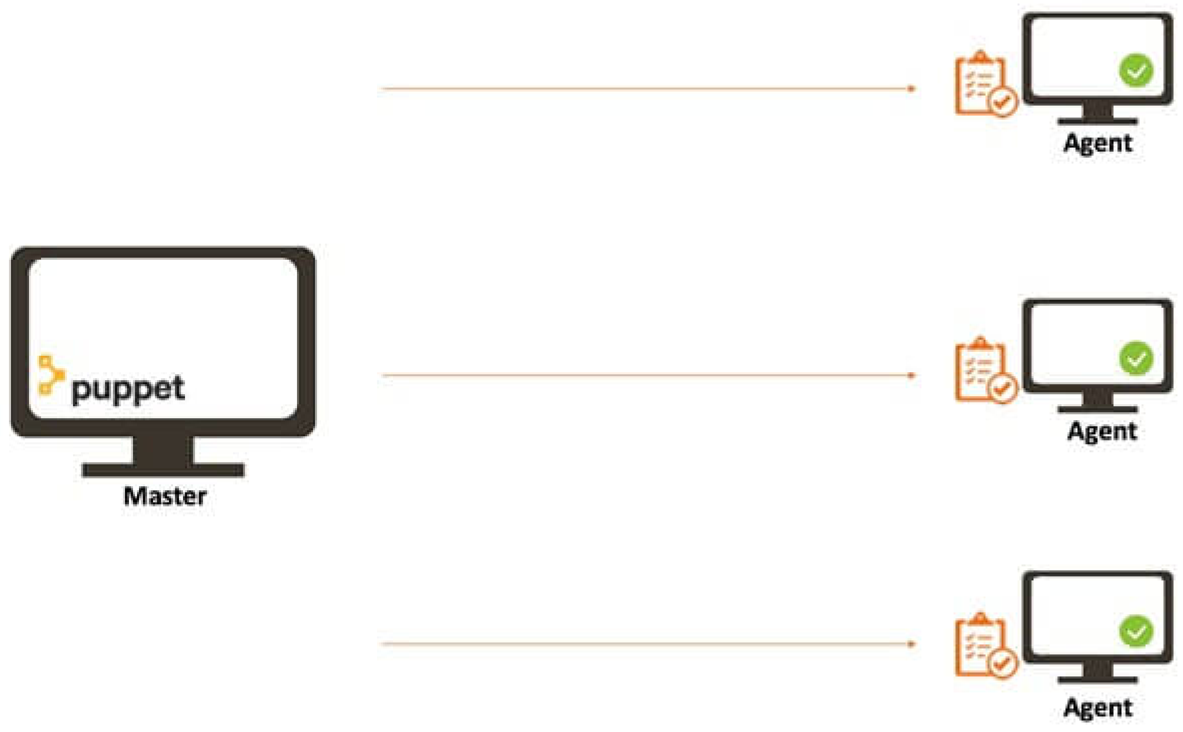
3. Master sends catalog to Agent



Master sends catalog to Agent

Puppet Master uses Facts' data and compiles a list with the configuration to be applied to the agent. This list of configuration to be performed on an agent is known as a **catalog (**package in stallation, upgrades or removals, File System creation, user creation or deletion, server reboot, IP configuration changes, etc.**).**

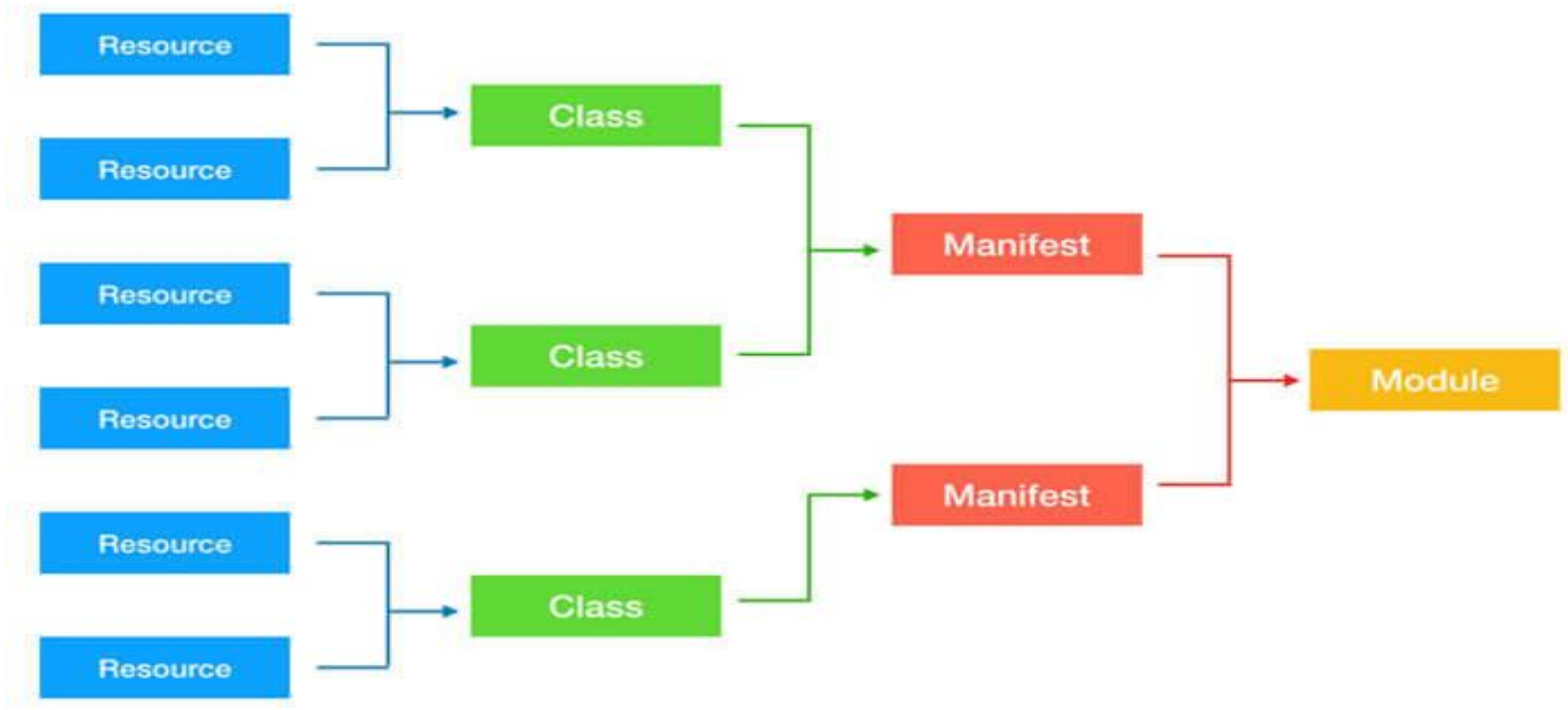
How Puppet Works? (Cont…)



4. Each Agents applies configuration

Note: Once the node apply configuration, then the node reports back to puppet master indicating that the configuration has been applied and completed.

Puppet Blocks



Puppet Blocks (Cont…)

Puppet Resources:
Puppet Resources are the building blocks of Puppet.
Resources are the **inbuilt functions** that run at the back end to perform the required operations in puppet.

Puppet Classes:
A combination of different resources can be grouped together into a single unit called class.

Puppet Manifest:
Manifest is a directory containing puppet DSL files. Those files have a .pp extension. The .pp extension stands for puppet program. The puppet code consists of definitions or declarations of Puppet Classes.

Puppet Modules:
Modules are a collection of files and directories such as Manifests, Class definitions. They are the re-usable and sharable units in Puppet.

Ref:https://www.guru99.com/puppet-tutorial.html and https://www.guru99.com/devops-tutorial.html

Differences between Puppet and Chef

How the two platforms (Puppet and Chef) 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.

Differences between Puppet and Chef (Cont…)

While both tools have similar goals, the means used to achieve them differ. Some of them are provided below:

2. How resources are described:

Puppet uses a **declarative language** that is similar to JSON or XML.
You describe the resource's state but cannot intervene in how this state is achieved.

Chef uses an **imperative language**. This means you more or less have full-featured Ruby at your disposal.

For example, **Puppet is like writing configuration files** whereas using Chef is like **programming the control of your nodes.**
If you or your team have more experience with system administration, you may prefer Puppet. On the other hand, if most of you are developers, Chef might be a better fit.

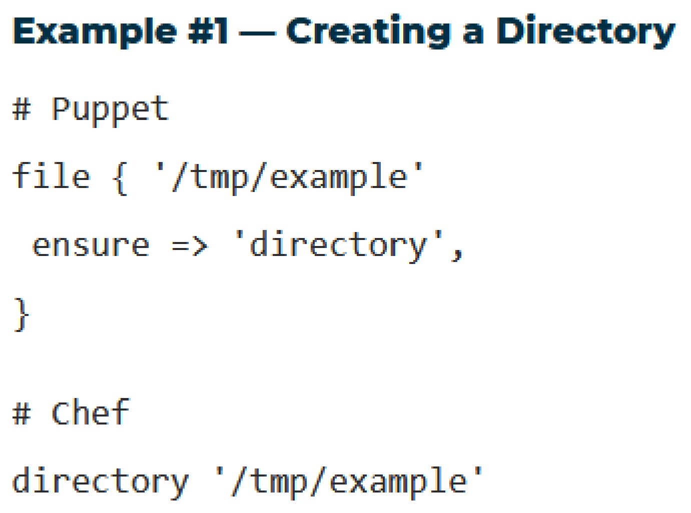
Differences between Puppet and Chef (Cont…)

2. **Configuration files:**
In Puppet, you create **manifests and modules**, while in Chef you deal with **recipes and cookbooks**.
Manifests and recipes usually describe **single resources** while **modules and cookbooks** describe the more general concepts (a LAMP server running your application, for instance).

Examples of resources with which one can deal with are files, directories, network interfaces, and applications. Commands like mkdir, cat, and apt-get or yum are replaced with desired states (“present,” “absent,” “updated,” and so on).

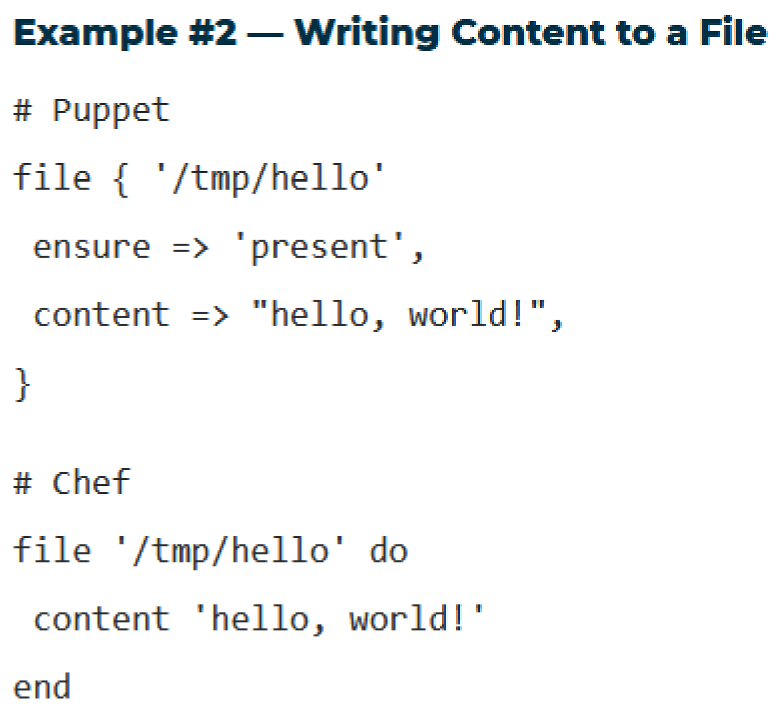
Differences between Puppet and Chef (Cont…)

Examples of resource definitions for a **directory and a file**



Example #1 — Creating a Directory

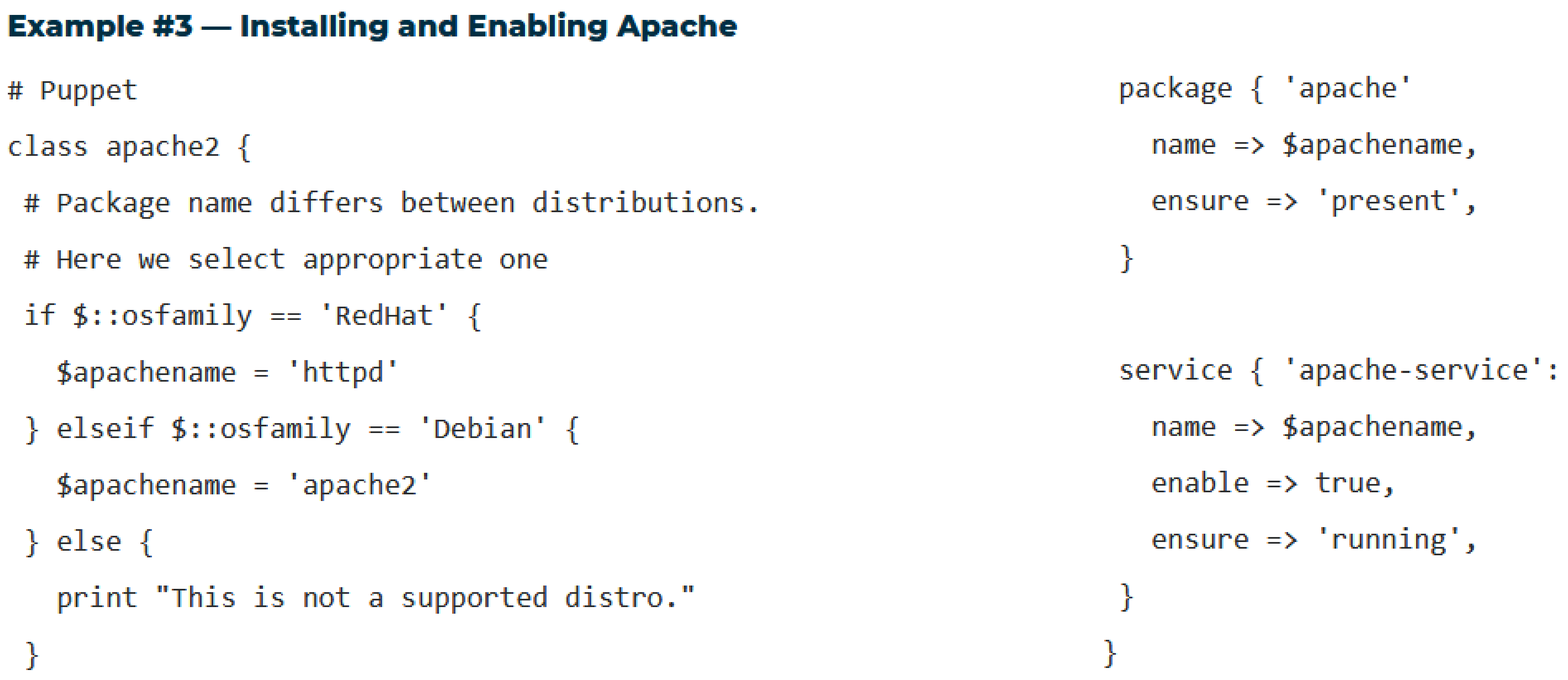
```
# Puppet
file { '/tmp/example'
 ensure => 'directory',
}

# Chef
directory '/tmp/example'
```



Example #2 — Writing Content to a File

```
# Puppet
file { '/tmp/hello'
 ensure => 'present',
 content => "hello, world!",
}

# Chef
file '/tmp/hello' do
 content 'hello, world!'
end
```

Differences between Puppet and Chef (Cont…)



Example #3 — Installing and Enabling Apache

```
# Puppet
class apache2 {
 # Package name differs between distributions.
 # Here we select appropriate one
 if $::osfamily == 'RedHat' {
   $apachename = 'httpd'
 } elseif $::osfamily == 'Debian' {
   $apachename = 'apache2'
 } else {
   print "This is not a supported distro."
 }
```

```
 package { 'apache'
   name => $apachename,
   ensure => 'present',
 }

 service { 'apache-service':
   name => $apachename,
   enable => true,
   ensure => 'running',
 }
}
```

Differences between Puppet and Chef (Cont…)

Examples of resource definitions for a **installing and Enabling Apache using Chef**

```
# Chef
case node[:platform]
 when 'ubuntu', 'debian'
   # Update apt cache every 86400 seconds (that is once a day)
   apt_update 'Update the apt cache daily' do
     frequency 86_400
     action :periodic
   end

   apachename = 'apache2'
 when 'redhat', 'centos'
   apachename = 'httpd'
end
```

```
# Install 'apache2' package
package 'Install apache2' do
 package_name apachename
end

# Enable and start the service
service apachename do
 supports :status => true
 action [:enable, :start]
end
```

3. OpenStack

Note: It is already discussed and please refer to module 3 slides that was shared with you on CMS

4. Heat

- **Heat** is the main project in the **OpenStack Orchestration program**.
- **Heat** allows developers to store the requirements of a cloud application in a file that defines what resources are necessary for that application.
- In many ways, Heat does for applications what OpenStack infrastructure components (Nova, Cinder, and the like) do for vendor hardware and software—it simplifies the integration.

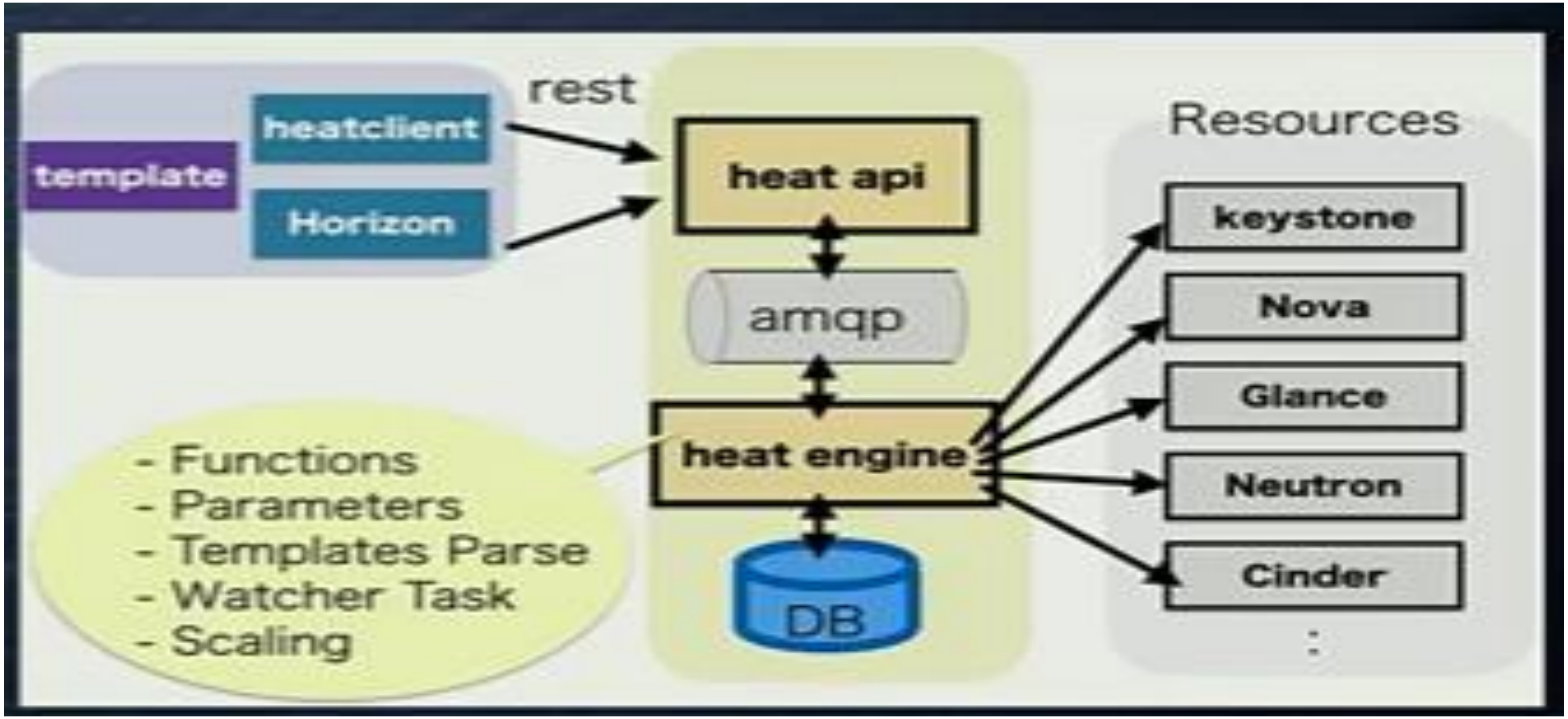
4. Heat (Cont...)

- It enables orchestration engine to launch multiple composite cloud app lications based on templates in the form of text files that can be treate d like code.
- A native Heat template format is evolving, but Heat also endeavours to provide compatibility with the AWS CloudFormation template format, s o that many existing CloudFormation templates can be launched on OpenStack.

heat-engine:

The heat engine does the main work of orchestrating the launch of templates and providing events back to the API consumer.

How heat works?



How heat works? (Cont….)

- Heat manages the whole lifecycle of the application - when you need to change your infrastructure, simply modify the template and use it to update your existing stack.
- A Heat template describes the infrastructure for a cloud application in a text file that is readable and writable by humans, and can be checked into version control, diffed, etc.
- Infrastructure resources that can be described include: servers, floating ips, volumes, security groups, users, etc.
- Heat knows how to make the necessary changes. It will delete all of the resources when you are finished with the application, too.
- Heat primarily manages infrastructure, but the templates integrate well with software configuration management tools such as **Puppet and Chef**.
- Heat also provides an autoscaling service that integrates with Telemetry, so you can include a scaling group as a resource in a template.
- Templates can also specify the relationships between resources (e.g. this volume is connected to this server).
- This enables Heat to call out to the OpenStack APIs to create all of your infrastructure in the correct order to com pletely launch your application.

Heat Orchestration Templates (HOT)

(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 that 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

Basic HOT template

#template structure
heap_template_version:2015-04-30
#validate version of HOT
description:
# Template description (features available or functions)
parameters:
#Template input parameters (Security groups, which one to use, which SSH key to use, etc.)
resources:
#Resources to be created (e.g., virtual machines).
outputs:
#The output (IP address to login the created VM)

Creating and deploying a Heat Template

A template which will create the following and connect them up:

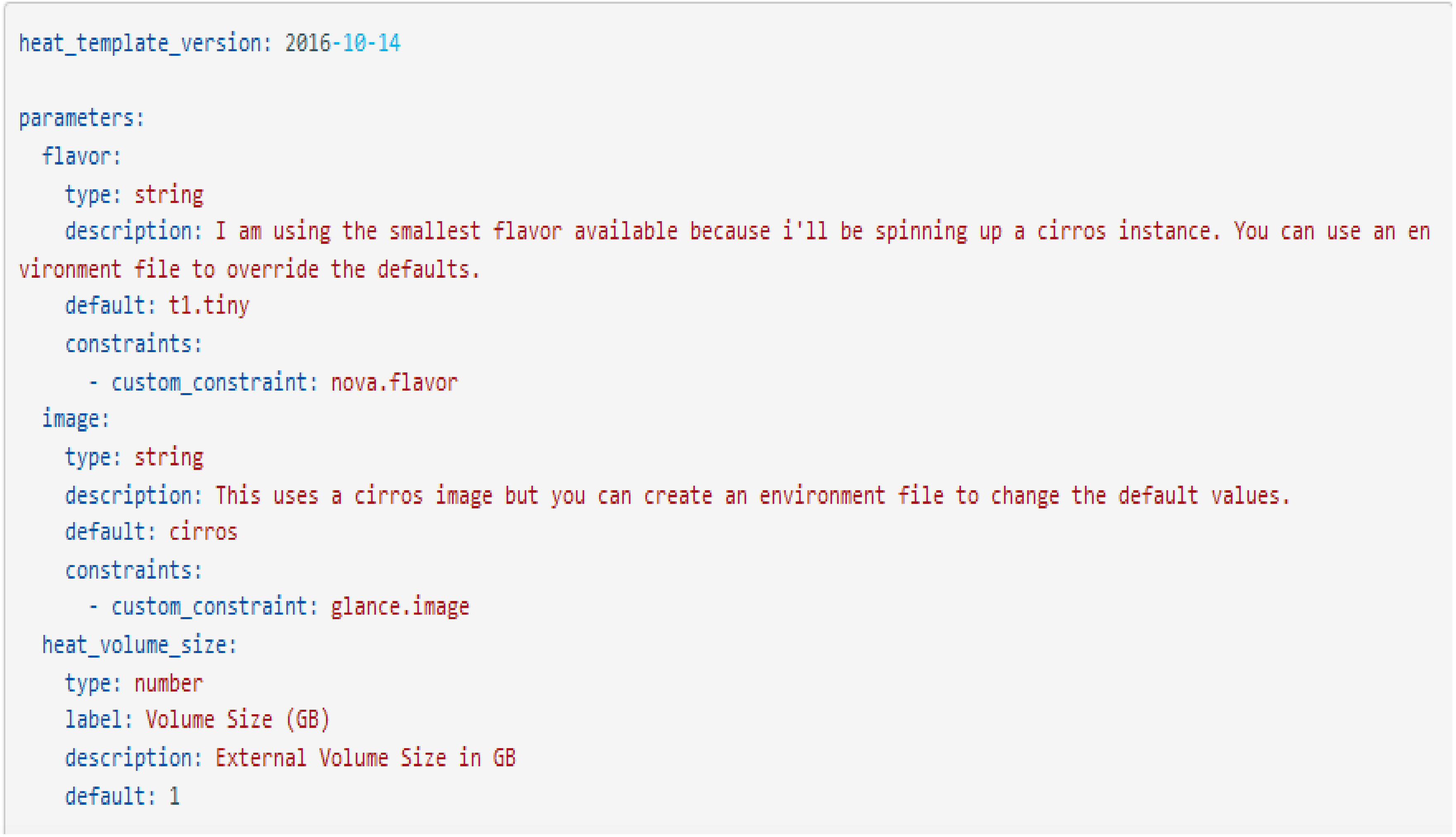
- A network
- A static IP
- A router
- An instance
- A floating IP
- A volume

The Heat template: The template must be in yaml format and be saved with the .yaml file extension.
Note: We should also make sure that you use the correct indentation.

Steps for deploying:

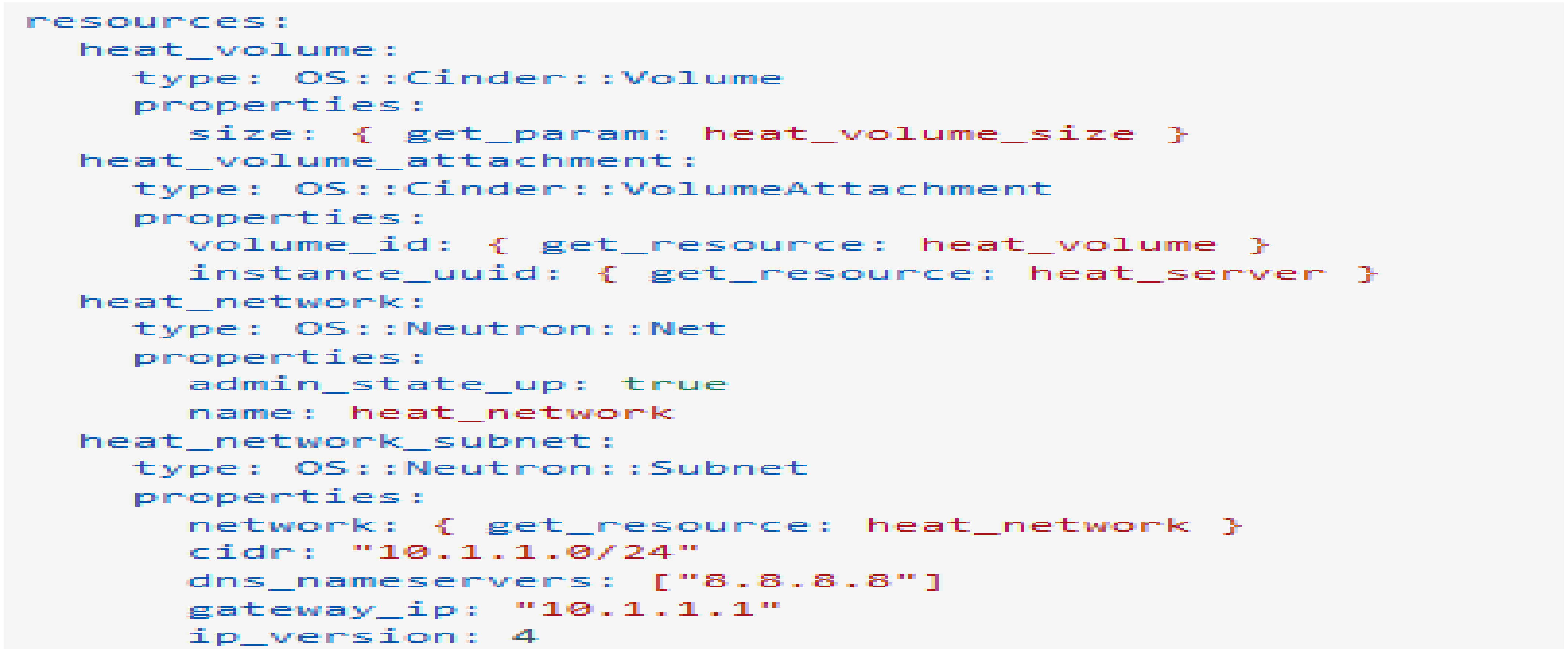
- Define a Heat template's required parameters in a YAML file, and
- Deploy the template via the command line (**OpenStack CLI)**.

basic-stack.yaml
(Note: A Stack means a group of connected cloud resources)



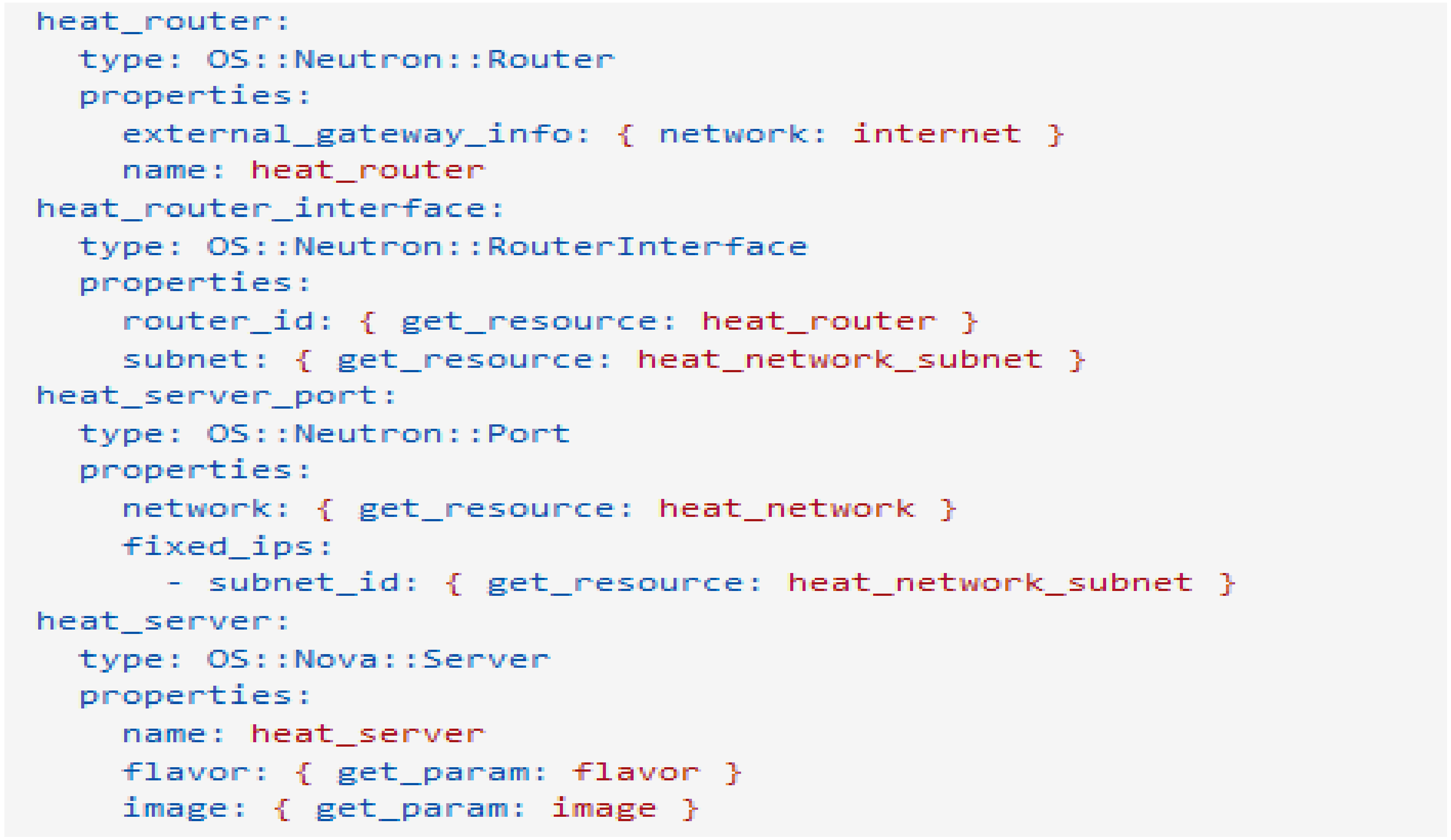
```yaml
heat_template_version: 2016-10-14

parameters:
  flavor:
    type: string
    description: I am using the smallest flavor available because i'll be spinning up a cirros instance. You can use an en
vironment file to override the defaults.
    default: t1.tiny
    constraints:
      - custom_constraint: nova.flavor
  image:
    type: string
    description: This uses a cirros image but you can create an environment file to change the default values.
    default: cirros
    constraints:
      - custom_constraint: glance.image
  heat_volume_size:
    type: number
    label: Volume Size (GB)
    description: External Volume Size in GB
    default: 1
```

basic-stack.yaml (cont….)



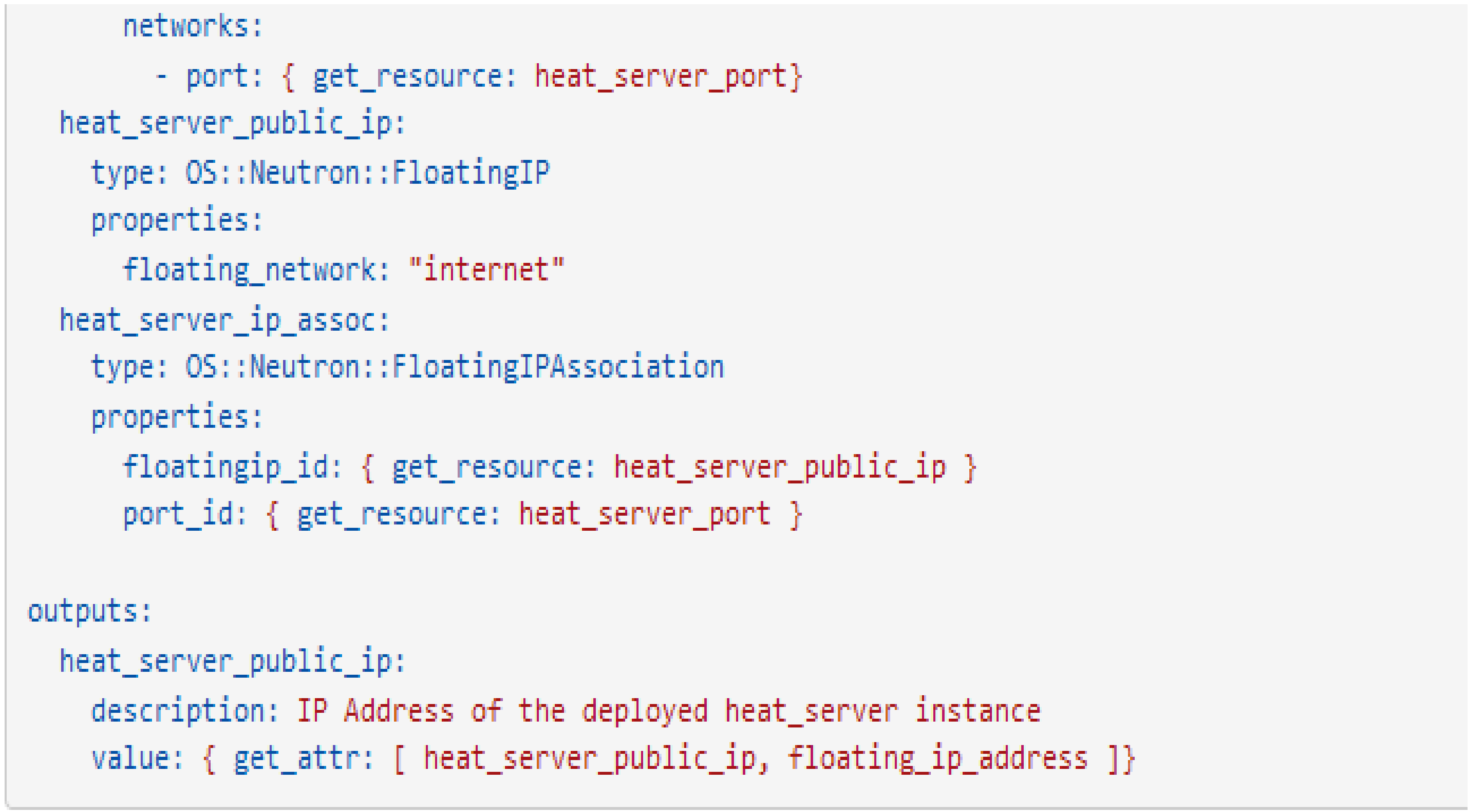
```
resources:
  heat_volume:
    type: OS::Cinder::Volume
    properties:
      size: { get_param: heat_volume_size }
  heat_volume_attachment:
    type: OS::Cinder::VolumeAttachment
    properties:
      volume_id: { get_resource: heat_volume }
      instance_uuid: { get_resource: heat_server }
  heat_network:
    type: OS::Neutron::Net
    properties:
      admin_state_up: true
      name: heat_network
  heat_network_subnet:
    type: OS::Neutron::Subnet
    properties:
      network: { get_resource: heat_network }
      cidr: "10.1.1.0/24"
      dns_nameservers: ["8.8.8.8"]
      gateway_ip: "10.1.1.1"
      ip_version: 4
```

basic-stack.yaml (cont….)



```
heat_router:
  type: OS::Neutron::Router
  properties:
    external_gateway_info: { network: internet }
    name: heat_router
heat_router_interface:
  type: OS::Neutron::RouterInterface
  properties:
    router_id: { get_resource: heat_router }
    subnet: { get_resource: heat_network_subnet }
heat_server_port:
  type: OS::Neutron::Port
  properties:
    network: { get_resource: heat_network }
    fixed_ips:
      - subnet_id: { get_resource: heat_network_subnet }
heat_server:
  type: OS::Nova::Server
  properties:
    name: heat_server
    flavor: { get_param: flavor }
    image: { get_param: image }
```

basic-stack.yaml (cont….)

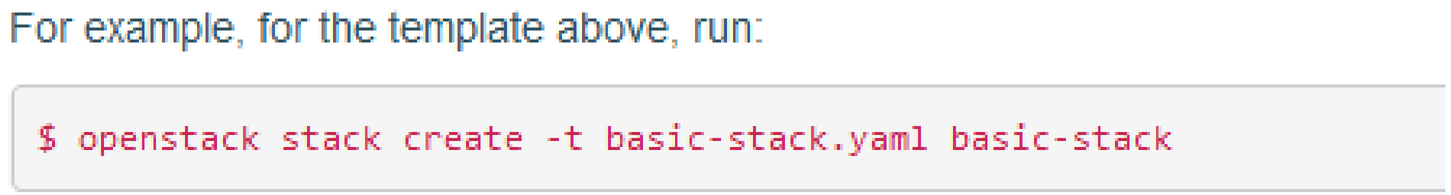


```yaml
      networks:
        - port: { get_resource: heat_server_port}
  heat_server_public_ip:
    type: OS::Neutron::FloatingIP
    properties:
      floating_network: "internet"
  heat_server_ip_assoc:
    type: OS::Neutron::FloatingIPAssociation
    properties:
      floatingip_id: { get_resource: heat_server_public_ip }
      port_id: { get_resource: heat_server_port }

outputs:
  heat_server_public_ip:
    description: IP Address of the deployed heat_server instance
    value: { get_attr: [ heat_server_public_ip, floating_ip_address ]}
```

Deploying the stack

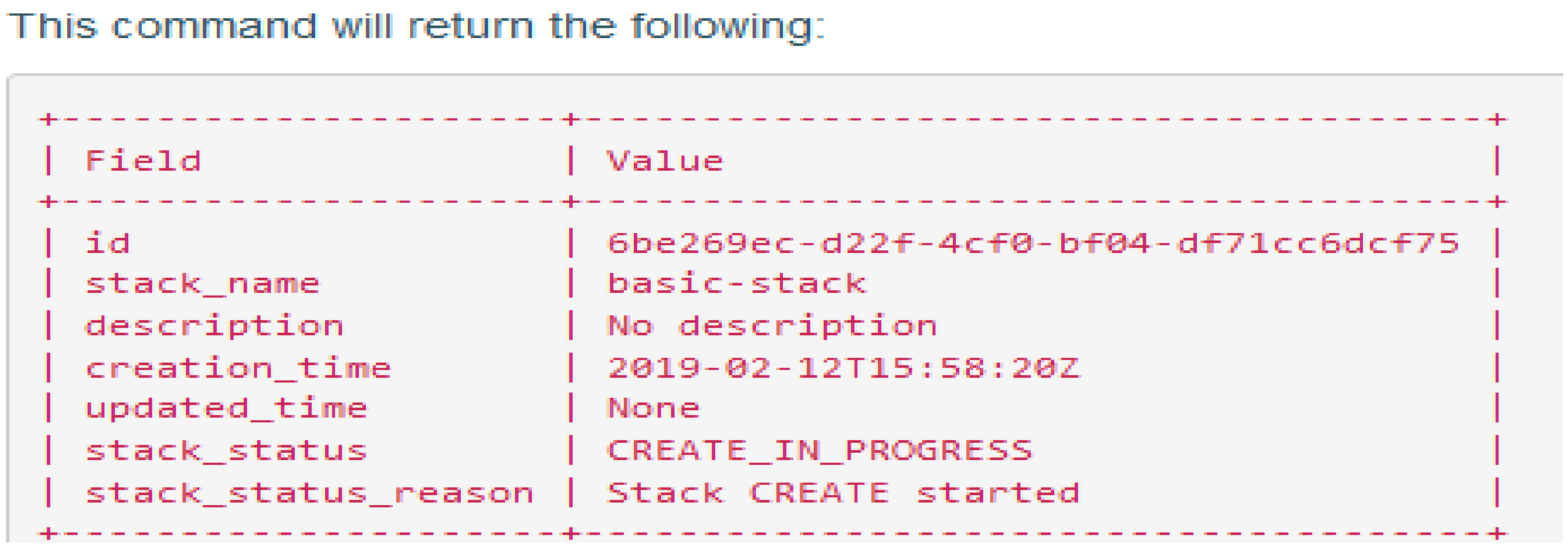
Create a stack by running the following command in the OpenStack CLI:

openstack stack create -t <template name> <stack name>



For example, for the template above, run:

```
$ openstack stack create -t basic-stack.yaml basic-stack
```



This command will return the following:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| id                  | 6be269ec-d22f-4cf0-bf04-df71cc6dcf75 |
| stack_name          | basic-stack                          |
| description         | No description                       |
| creation_time       | 2019-02-12T15:58:20Z                 |
| updated_time        | None                                 |
| stack_status        | CREATE_IN_PROGRESS                   |
| stack_status_reason | Stack CREATE started                 |
+---------------------+--------------------------------------+
```

References of Heat

https://docs.ukcloud.com/articles/openstack/ostack-how-create-heat-template.html
https://clouddocs.f5.com/cloud/openstack/v1/heat/how-to-deploy-heat-stack.html
Ref:https://wiki.openstack.org/wiki/Heat
https://developer.ibm.com/articles/cl-cloud-orchestration-technologies-trs/
https://livebook.manning.com/book/openstack-in-action/chapter-12/11

5. Juju

What is Juju?

- Juju is a state-of-the-art, open source modelling tool for operating software in the cloud.
- Juju allows us to deploy, configure, manage, maintain, and scale cloud applications quickly and efficiently on public clouds as well as physical servers, OpenStack, and containers.
- We can use Juju from the command line or through its beautiful GUI.
- The goal of Juju is to reuse of common operational code in widely different environments.
- It is an open source automatic service orchestration management tool developed by Canonical, the developers of the Ubuntu OS.

Application modelling?

- In modern environments, Even simple applications may require several other applications in order to function - li ke a database and a web server.
- For modeling a more complex system, e.g. OpenStack, many more applications need to be installed, configured and connected to each other.
- **Juju's application modelling provides tools to express the intent of how to deploy such applications and to subsequently scale and manage them.**

- With Juju, you create a model of the relationships between applications that make up our solution and we have a mapping of the parts of that model to machines. Juju then applies the necessary configuration management scripts to each machine in the model.

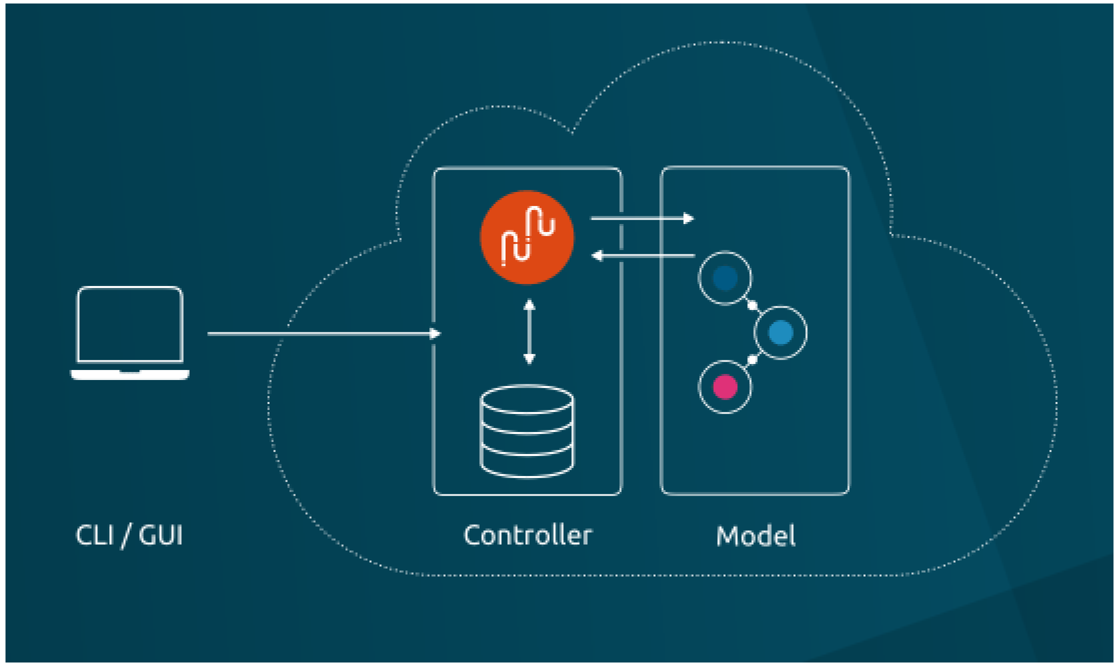
- Application-specific knowledge such as dependencies, scale-out practices, operational events like backups and updates, and integration options with other pieces of software are encapsulated in Juju's 'charms'.

How Juju works?

Juju controller: the heart of Juju

Juju's controller manages all the machines in our running models, responding to the events that are triggered throughout the system.

It also manages scale out, configuration and placement of all your models and applications.



Terminologies of Juju

Charms
Charms are sets of scripts for deploying and operating software. With event handling built in, they can declare interfaces that fit charms for other services, so relationships can be formed.

Bundles
Bundles are collections of charms that link services together, so you can deploy whole chunks of app infrastructure in one go.

What they do:
Install
Configure
Connect
Upgrade and update
Scale out and scale back
Perform health checks
Undertake operational actions

charms

- Juju utilizes **charms**, which are open source tools that simplify specific deployment and management tasks.
- The charm defines everything you all collaboratively know about deploying that particular application brilliantly
- Charms encapsulate application configurations, define how services are deployed, how they connect to other services, and how they are scaled. Also define how services integrate, and how their service units react to events in the distributed environment, as orchestrated by Juju.

- After a service is deployed, Juju can define relationships between services and expose some services to the outside world.

Notes:

- 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.
- 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.**
- **Charm store** includes a collection of charms that let you deploy whatever services you like in Juju.

Charms (Cont…)

What language are charms written in?
Charms can be written in any language or configuration management scripting system. Chef and Puppet are common, **more complex charms tend to use Python.**
Charms have been written in **Ruby, PHP,** and many charms are a collection of simple bash scripts.

What charms are currently available?
Charms are available for hundreds of common and popular cloud-oriented applications such as **MySQL, MongoDB,** and others, with new ones being added every day. Check out the public charm store for an up to the minute list of charms: Juju Charm Store

Application scenario with Charms and Bundles

- Modern applications are typically composed of many applications - databas es, front-ends, big data stores, logging systems, key value stores.
- Each application will be defined by a **single charm**.
- To describe the whole application you need to describe the set of charms an d their relationships - what is connected to what - and we use **a bundle for** t hat.

Example Application with Juju (Charms and Bundles)

- A **content-management system** bundle could specify a database and a content management server, together with key-value stores and front-end load-balancing systems.
- Each of those applications are described by a **charm**; the bundle describes **the set of charms**, their configuration, and the relationships between them.
- This allows development teams to share not only the core primitive for each application, but also enables sharing higher-level models of several applications.
- It allows you to replicate complex application models in a cloud just by dropping the same bundle onto your Juju GUI.

- Bundles can be shared publicly or privately.
- New bundles are added to the public collection every week. And you can now bootstrap Juju, launch the GUI and deploy a bundle with a single command. That means you can go from zero to a full cloud dep loyment in seconds with the right bundle.

Benefits of Juju

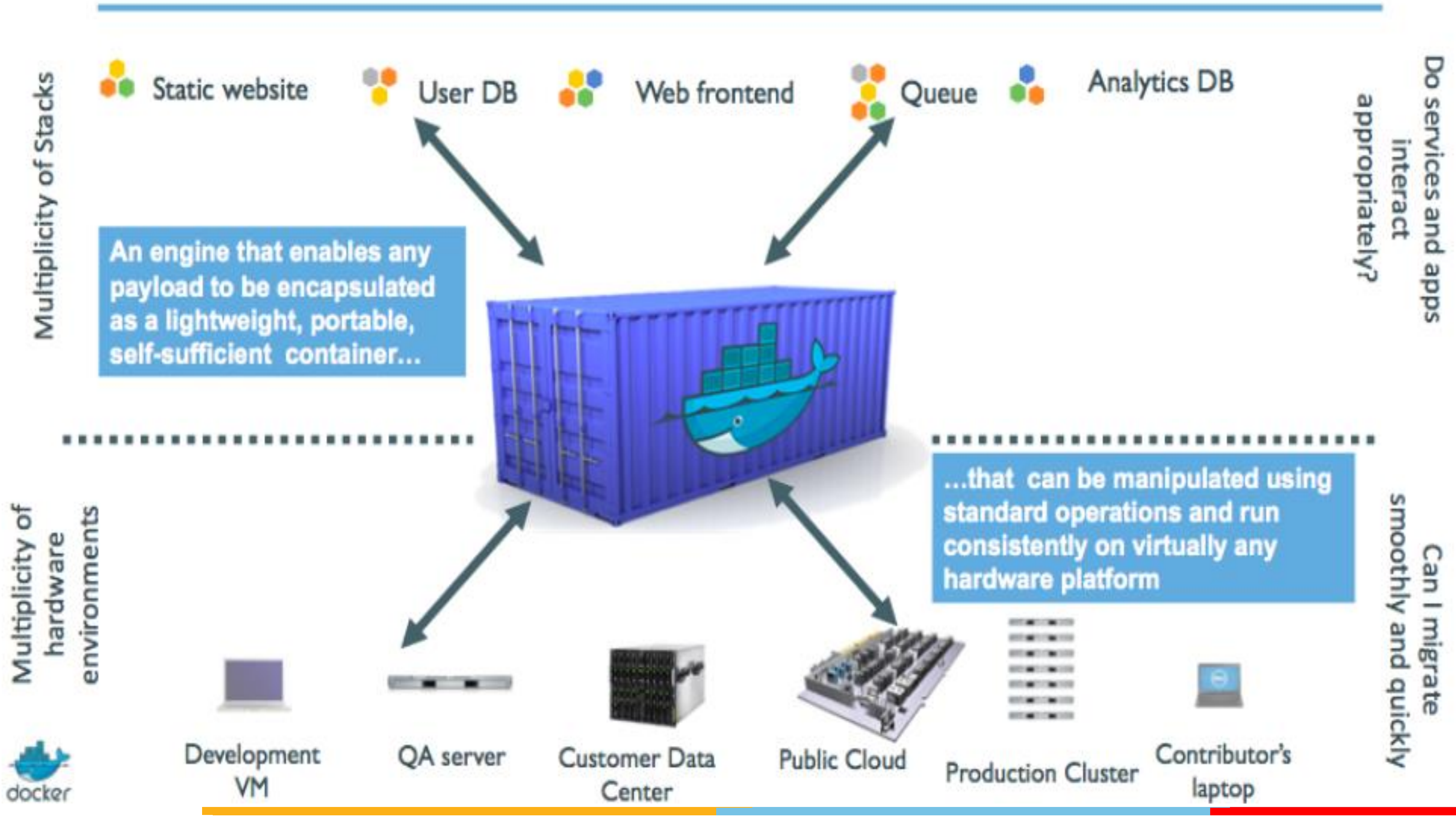
- Juju is the fastest way to model and deploy applications or solutions on all major public clouds and containers.
- It helps to reduce deployment time from days to minutes.
- Juju works with existing configuration management tools, and can scale workloads up or down very easily.
- No prior knowledge 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.**
- 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.

References of Juje:
https://developer.ibm.com/articles/cl-cloud-orchestration-technologies-trs/
https://juju.is/docs/writing-your-first-juju-charm
https://www.youtube.com/watch?v=Lt9-a6pDxsA

Dockers

- It is an open-source project that automates the deployment of applications inside software containers.
- All applications have their own dependencies, which include both software and hardware resources.
- **Docker is a mechanism that helps in isolating the dependencies per each application by packing them into containers.**
- It provides tools for simplifying DevOps by enabling developers to create templates called images that can be used to create lightweight virtual machines called containers, which include their applications and all of their applications' dependencies.

A shipping container system for applications


Multiplicity of Stacks
Static website
User DB
Web frontend
Queue
Analytics DB
Do services and apps interact appropriately?
An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…
…that can be manipulated using standard operations and run consistently on virtually any hardware platform
Multiplicity of hardware environments
Development VM
QA server
Customer Data Center
Public Cloud
Production Cluster
Contributor's laptop
Can I migrate smoothly and quickly
docker

What benefits the Docker (Container) provide?

- Docker enables developers to easily **pack, ship, and run any application as a lightweight, portable, self-sufficient container, which can run virtually anywhere.** Containers do this by enabling developers to isolate code into a single container. This makes it easier to modify and update the program.
- Containerization is an approach of running applications on an OS such that the application is isolated from the rest of the system.
- Main features of Docker are **Develop, ship and run** anywhere.
- They aim at facilitating developers to easily develop applications, ship them along with their dependencies into containers which can then be deployed anywhere. It is aimed to address portability issue.

Note: The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.

Ref: http://www.ce.uniroma2.it/courses/sdcc1819/slides/Docker.pdf and Also**, refer to module 2 slides for more information about Dockers.**

Container orchestration (e.g., Docker Swarm)

Container Orchestration
It is the automatic process of managing or scheduling the work of individual containers for applications based on **microservices** within multiple clusters.
Examples of container orchestration platforms are based on open-source versions like **Kubernetes, Docker Swarm or the commercial version from Red Hat OpenShift**.

Docker container orchestration tool, also known as **Docker** Swarm, can package and run applications as containers, find existing **container** images from others, and deploy a **container** on a laptop, server or cloud (public cloud or private).

Reference: https://avinetworks.com/glossary/container-orchestration/

Why Do We Need Container Orchestration?

Container orchestration is used to automate the following tasks at scale:

- Configuring and scheduling of containers
- Provisioning and deployments of containers
- Availability of containers
- The configuration of applications in terms of the containers that they run in
- Scaling of containers to equally balance application workloads across infrastructure
- Allocation of resources between containers
- Load balancing, traffic routing and service discovery of containers
- Health monitoring of containers
- Securing the interactions between containers.

How Does Container Orchestration Work?

- Configurations files tell the container orchestration tool how to network between containers and where to store logs.

- The orchestration tool also schedules deployment of containers into clusters and determines the best host for the container. After a host is decided, the orchestration tool manages the lifecycle of the container based on predeter mined specifications. Container orchestration tools work in any environment that runs containers.

Orchestration tools for Docker include the following:
• Docker Machine — Provisions hosts and installs Docker Engine.
• Docker Swarm — Clusters multiple Docker hosts under a single host. It can also integrate with any tool that works with a single Docker host.
• Docker Compose — Deploys multi-container applications by creating the required containers.

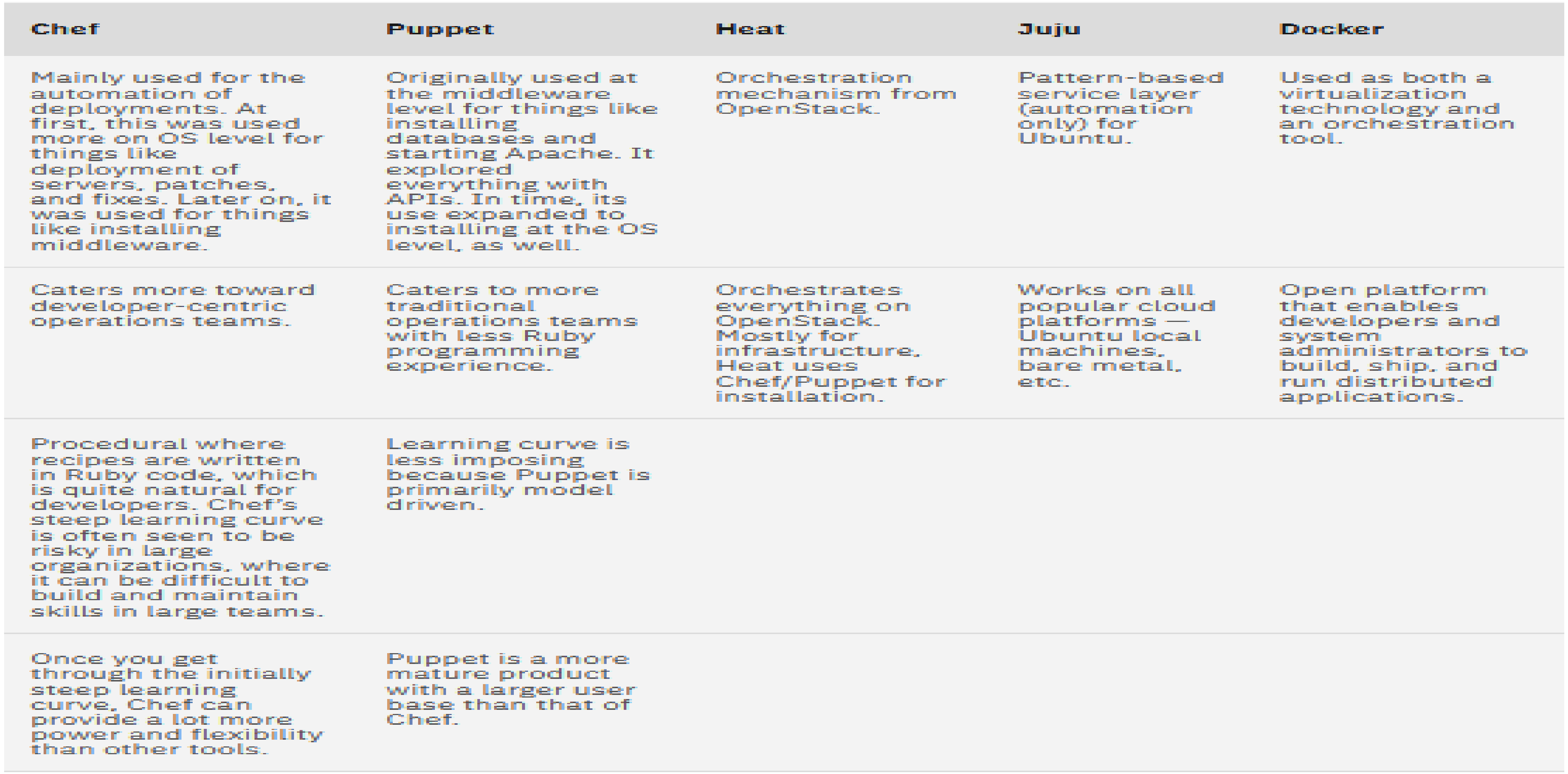
Benefits of Containerized Orchestration Tools

Increased portability — Scale applications with a single command and only scale specific functions without affecting the entire application.
- **Simple and fast deployment** — Quickly create new containerized applications to address growing traffic.
- **Enhanced productivity** — Simplified installation process and decreased dependency errors.
- Improved security — Share specific resources without risking internal or external security.

Application isolation improves web application security by separating each application’s process into different containers.

Note: **Refer to the slides of Module 2** for more information about the **Docker files and their creation.**

Comparison of cloud Orchestration tools



Chef	Puppet	Heat	Juju	Docker
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.	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.	Orchestration mechanism from OpenStack.	Pattern-based service layer (automation only) for Ubuntu.	Used as both a virtualization technology and an orchestration tool.
Caters more toward developer-centric operations teams.	Caters to more traditional operations teams with less Ruby programming experience.	Orchestrates everything on OpenStack. Mostly for infrastructure. Heat uses Chef/Puppet for installation.	Works on all popular cloud platforms — Ubuntu local machines, bare metal, etc.	Open platform that enables developers and system administrators to build, ship, and run distributed applications.
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.	Learning curve is less imposing because Puppet is primarily model driven.			
Once you get through the initially steep learning curve, Chef can provide a lot more power and flexibility than other tools.	Puppet is a more mature product with a larger user base than that of Chef.			