Configuration Management

Why Configuration Management ?

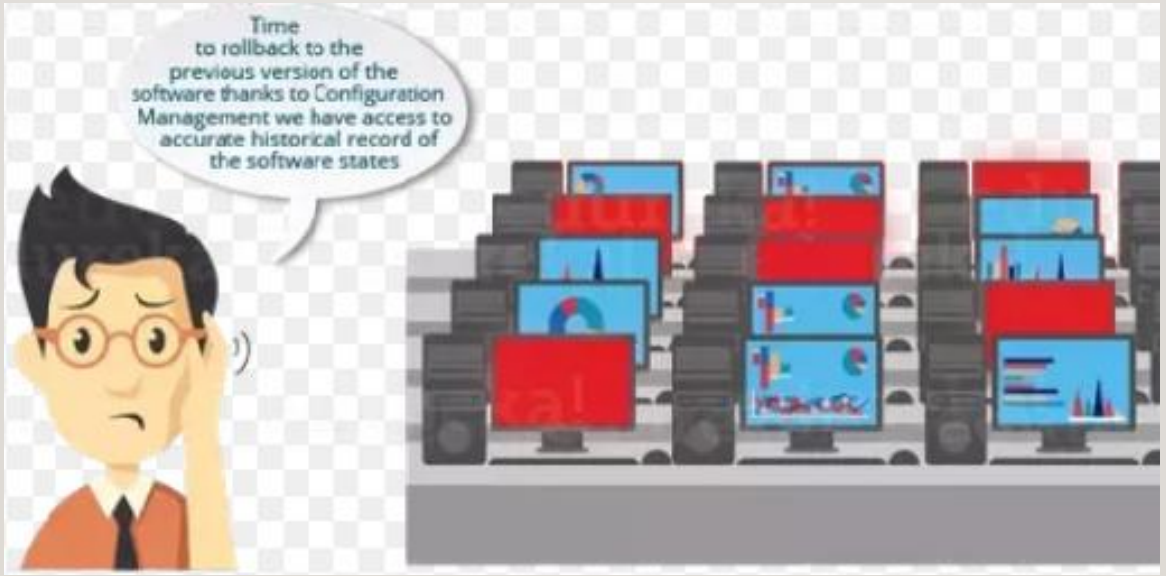
- System Administrators usually perform repetitive tasks such as installing servers, configuring those servers, etc. They can automate this task, by writing scripts, but it is a very hectic job when you are working on a large infrastructure.
- Configuration Management is the practice of handling changes systematically so that a system maintains its integrity over time. Configuration Management (CM) ensures that the current design and build state of the system is known, good & trustedIt allows access to an accurate historical record of system state for project management and audit purposes.

❖ **Configuration Management overcame the following challenges:**

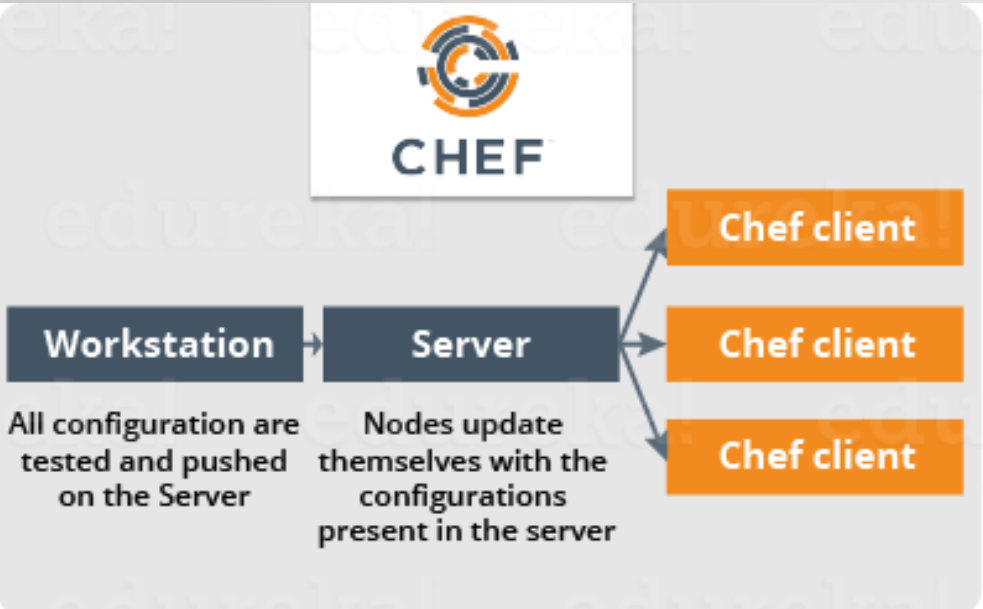
- Figuring out which components to change when requirements change.
- Redoing an implementation because the requirements have changed since the last implementation.
- Reverting to a previous version of the component if you have replaced with a new but flawed version.
- Replacing the wrong component because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

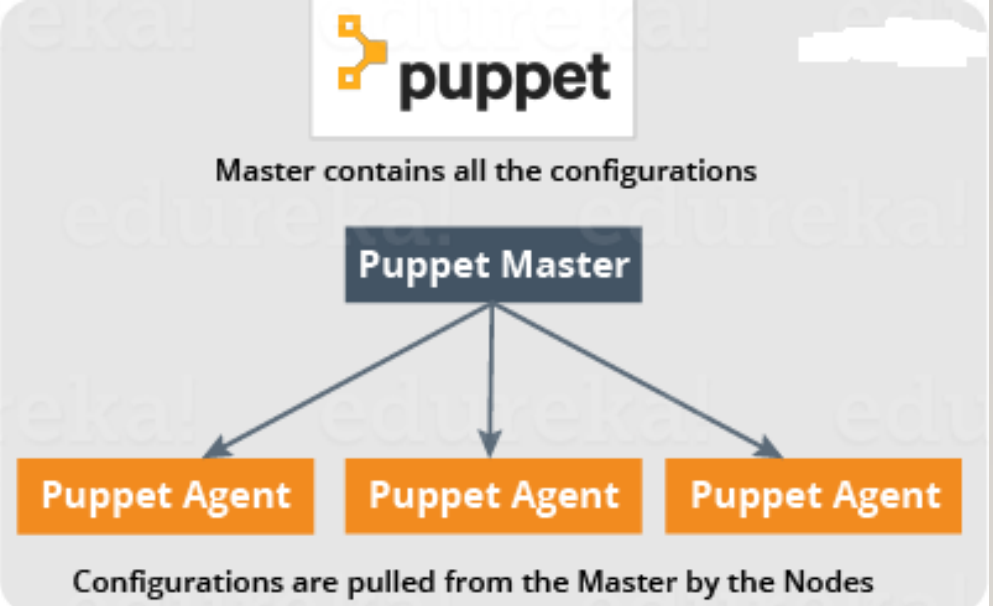
Case Study

- A software “glitch” prevented the NYSE from trading stocks for almost 90 minutes. This led to millions of dollars of loss. A new software installation caused the problem. That software was installed on 8 of its 20 trading terminals and the system was tested out the night before. However, in the morning, it failed to operate properly on the 8 terminals. So there was a need to switch back to the old software. You might think that this was a failure of NYSE’s Configuration Management process, but in reality it was a success. As a result of a proper Configuration Management process, NYSE recovered from that situation in 90 minutes which was pretty fast. Had the problem continued longer, the consequences would have been more severe.



CHEF & PUPPET





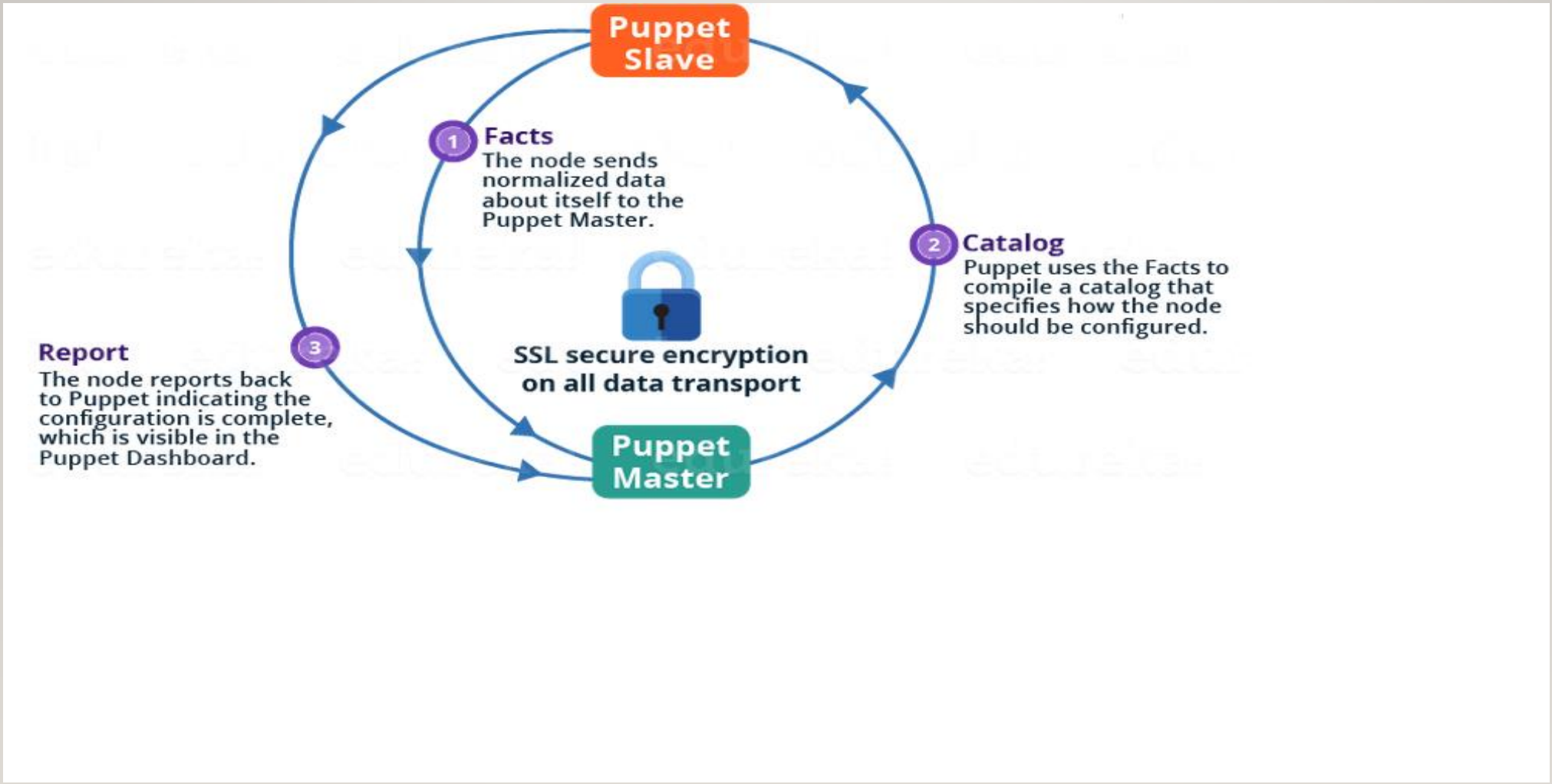
Overview of Puppet

- Puppet is a Configuration Management tool that is used for deploying, configuring and managing servers.

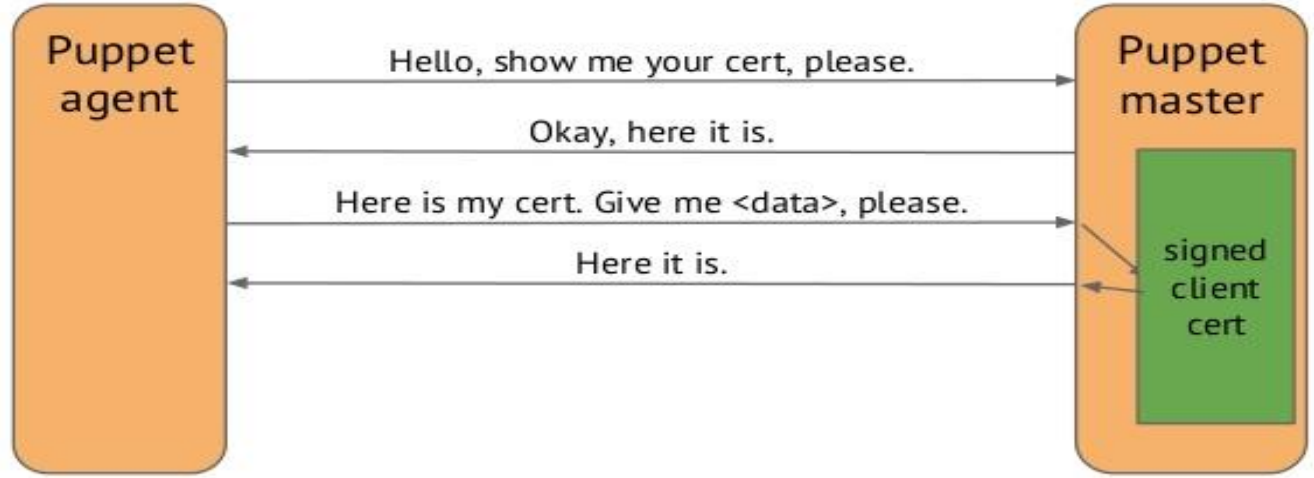
❖ It performs the following functions:

- Defining distinct configurations for each and every host, and continuously checking and confirming whether the required configuration is in place and is not altered (if altered Puppet will revert back to the required configuration) on the host.
- Dynamic scaling-up and scaling-down of machines.
- Providing control over all your configured machines, so a centralized (master-server or repo-based) change gets propagated to all, automatically.
- Puppet uses a Master Slave architecture in which the Master and Slave communicate through a secure encrypted channel with the help of SSL.

Puppet Architecture

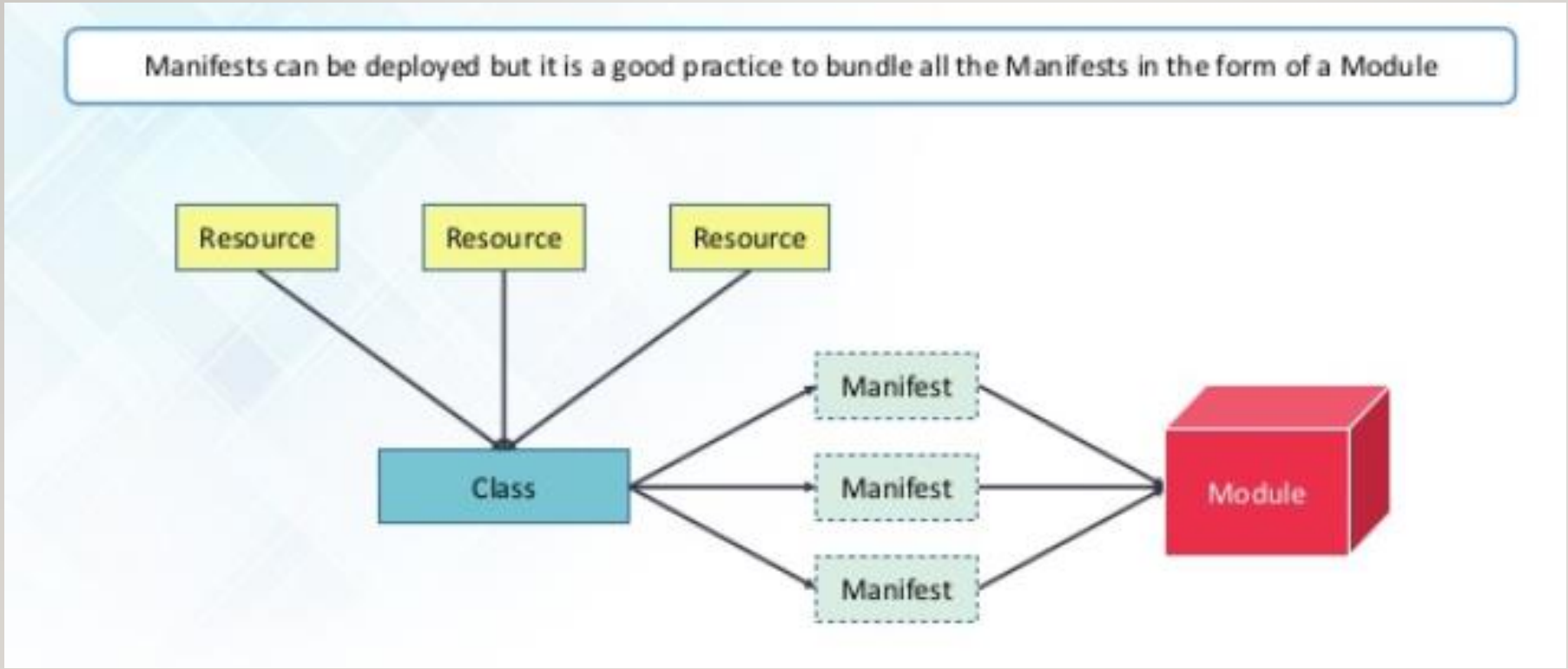


Puppet Master Slave Communication



- Puppet Slave asks for Puppet Master certificate.
- After receiving Puppet Master certificate, Master requests for Slave certificate.
- Once Master has signed the Slave certificate, Slave requests for configuration/data.
- Finally, Puppet Master will send the configuration to Puppet Slave.

Resources , Classes , Manifest and Modules



Resources , Classes , Manifest and Modules

- **Manifests:** All the configuration details of Slave is available in Puppet Master, written in the native Puppet language. These details are written in the language which Puppet can understand and are termed as Manifests. They are composed of Puppet code and their filenames use the .pp extension. These are basically Puppet programs.

For example: Write a Manifest in Puppet Master that creates a file and installs Apache server on all Puppet Slaves connected to the Puppet Master.

- **Module:** A Puppet Module is a collection of Manifests and data (such as facts, files, and templates), and they have a specific directory structure. Modules are useful for organizing your Puppet code, because they allow you to split your code into multiple Manifests. Modules are self-contained bundles of code and data.
- **Resource:** Resources are the fundamental unit for modeling system configurations. Each Resource describes some aspect of a system, like a specific service or package.

- **Facter:** Facter gathers basic information (facts) about Puppet Slave such as hardware details, network settings, OS type and version, IP addresses, MAC addresses, SSH keys, and more. These facts are then made available in Puppet Master's Manifests as variables.

Resources , Classes , Manifest and Modules

- **Mcollective:** It is a framework that allows several jobs to be executed in parallel on multiple Slaves. It performs various functions like:
  - Interact with clusters of Slaves, whether in small groups or very large deployments.
  - Use a broadcast paradigm to distribute requests. All Slaves receive all requests at the same time, requests have filters attached, and only Slaves matching the filter will act on requests.
  - Use simple command-line tools to call remote Slaves.
  - Write custom reports about your infrastructure.

- **Catalogs:** A Catalog describes the desired state of each managed resource on a Slave. It is a compilation of all the resources that the Puppet Master applies to a given Slave, as well as the relationships between those resources. Catalogs are compiled by a Puppet Master from Manifests and Slave-provided data (such as facts, certificates, and an environment if one is provided), as well as an optional external data (such as data from an external Slave classifier, exported resources, and functions). The Master then serves the compiled Catalog to the Slave when requested.

Key Benefits of Puppet

- **Large installed base:** Puppet is used by more than 30,000 companies worldwide including Google, Red Hat, Siemens, etc. along with several universities like Stanford and Harvard law school. An average of 22 new organizations per day use Puppet for the first time.
- **Large developer base:** Puppet is so widely used that lots of people develop for it. Puppet has many contributors to its core source code.
- **Long commercial track record:** Puppet has been in commercial use since 2005, and has been continually refined and improved. It has been deployed in very large infrastructures (5,000+ machines) and the performance and scalability lessons learned from these projects have contributed in Puppet's development.
- **Documentation:** Puppet has a large user-maintained wiki with hundreds of pages of documentation and comprehensive references for both the language and its resource types. In addition, it's actively discussed on several mailing lists and has a very popular IRC channel, so whatever your Puppet problem, it's easy to find the answer.
- **Platform support:** Puppet Server can run on any platform that supports ruby for ex: CentOS, Microsoft Windows Server, Oracle Enterprise Linux etc. It not only supports the new operating systems but it can also run on relatively old and out-of-date OS and Ruby versions as well.

Zynga – Puppet Case Study

- Every month, more than 215 million people play games made by Zynga -- that's 10 percent of the world's internet population enjoying Farmville, Mafia Wars, Zynga Poker, FrontierVille and more. TechCrunch reported in September that Zynga's properties move 1 petabyte of data every day and are adding as many as 1,000 servers each week. To accommodate their growing traffic, Zynga uses Puppet for configuration management for their tens of thousands of machines.

Mark Stockford, Vice President of Production Operations at Zynga said:

Puppet is fantastic at configuration management and everyone is really excited about it. We selectedPuppet for its flexibility, features and ease of use.

Zynga – Puppet Case Study

- **Speed of Recovery** – The production operations team can rapidly deploy the right configuration to the right box. If a system gets inappropriately reconfigured Puppet will automatically revert it back to a last stable state, or provide the details necessary to manually remediate a system rapidly.
- **Speed of Deployment** – Puppet has provided significant time savings in the way the operations team delivers services for the gaming studios.
- **Consistency of Servers** – Puppet's model-driven framework ensures consistent deployments. According to *Mark Stockford, Vice President Production Operations*, Zynga *"It is evident that we have experienced time savings. The beauty of using Puppet is that it allows us to deliver consistent configurations across our servers in a short period every time."*
- **Collaboration** – Having a model-driven approach makes it easy to share configurations across the organization, enabling developers and operations teams to work together to ensure new service delivery is of extremely high quality. Over a dozen people from Zynga's team got trained in Puppet. This knowledge has been disseminated throughout the team and to the operations teams within each individual gaming studio.

Overview of Chef

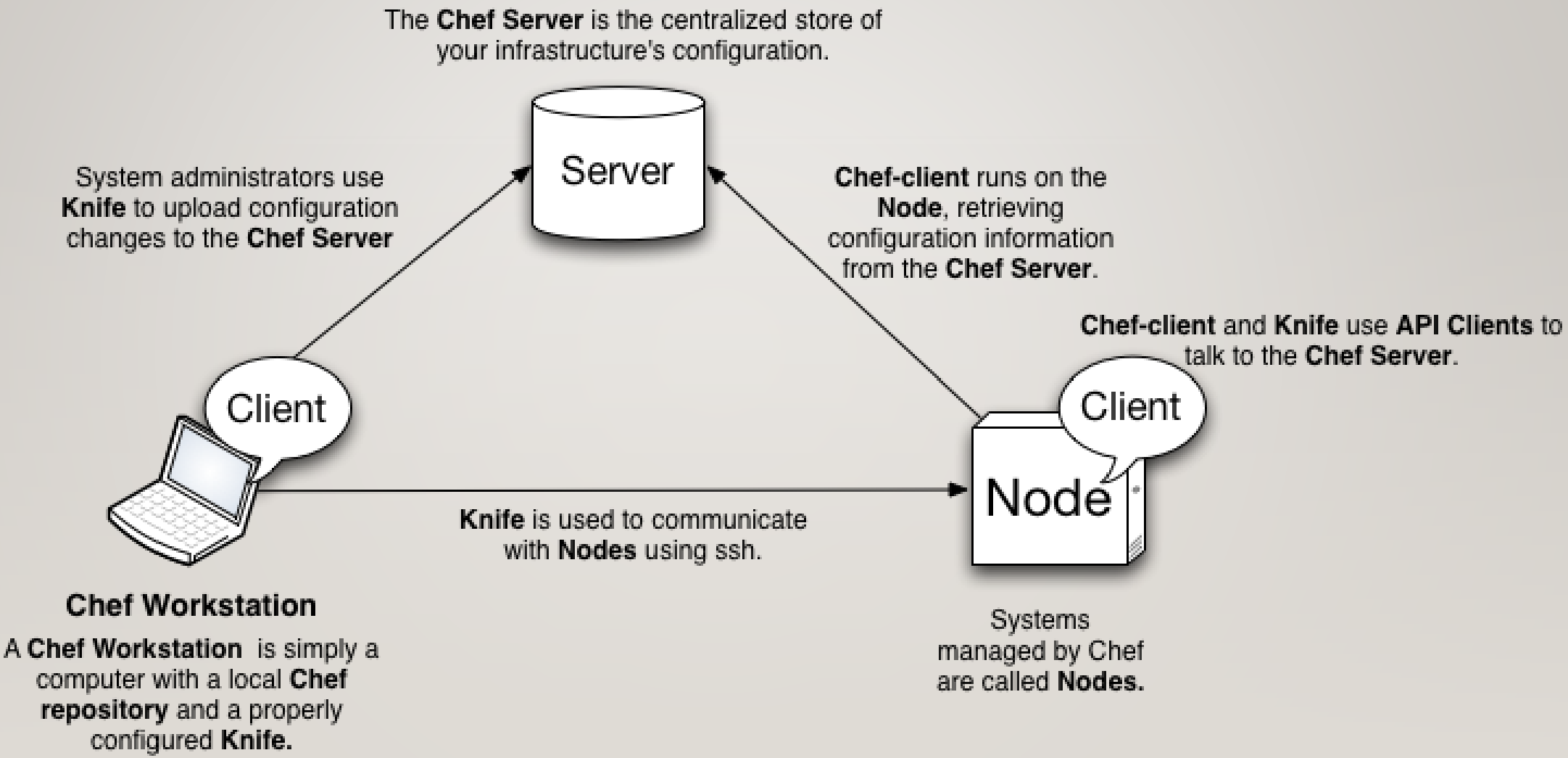
Chef is an automation tool that provides a way to define infrastructure as code. Infrastructure as code (IAC) simply means that managing infrastructure by writing code (Automating infrastructure) rather than using manual processes. It can also be termed as programmable infrastructure. Chef uses a pure-Ruby, domain-specific language (DSL) for writing system configurations. Below are the types of automation done by Chef, irrespective of the size of infrastructure:

- Infrastructure configuration
- Application deployment

Key Metrics

- Chef supports multiple platforms like AIX, RHEL/CentOS, FreeBSD, OS X, Solaris, Microsoft Windows and Ubuntu. Additional client platforms include Arch Linux, Debian and Fedora.
- 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.
- Chef has an active, smart and fast growing community support.
- Used by giants like Mozilla, Expedia, Facebook, HP Public Cloud, Prezi, Xero, Ancestry.com, Rackspace, Get Satisfaction, IGN, Marshall University, Socrata, University of Minnesota, Wharton School of the University of Pennsylvania, Bonobos, Splunk, Citi, DueDil, Disney, and Cheezburger

Chef Architecture



Recipes and Cookbooks

- Recipes: A Recipe is a collection of resources that describes a particular configuration or policy. It describes everything that is required to configure part of a system. The user writes Recipes that describe how Chef manages applications and utilities and how they are to be configured.
- These Recipes describe a series of resources that should be in a particular state, i.e. Packages that should be installed, services that should be running, or files that should be written.
- Cookbooks: Multiple Recipes can be grouped together to form a Cookbook. A Cookbook defines a scenario and contains everything that is required to support that scenario:
  - Recipes, which specifies the resources to use and the order in which they are to be applied
  - Attribute values
  - File distributions
  - Templates
  - Extensions to Chef, such as libraries, definitions, and custom resources

DEMO

Creating AWS Stack using Chef

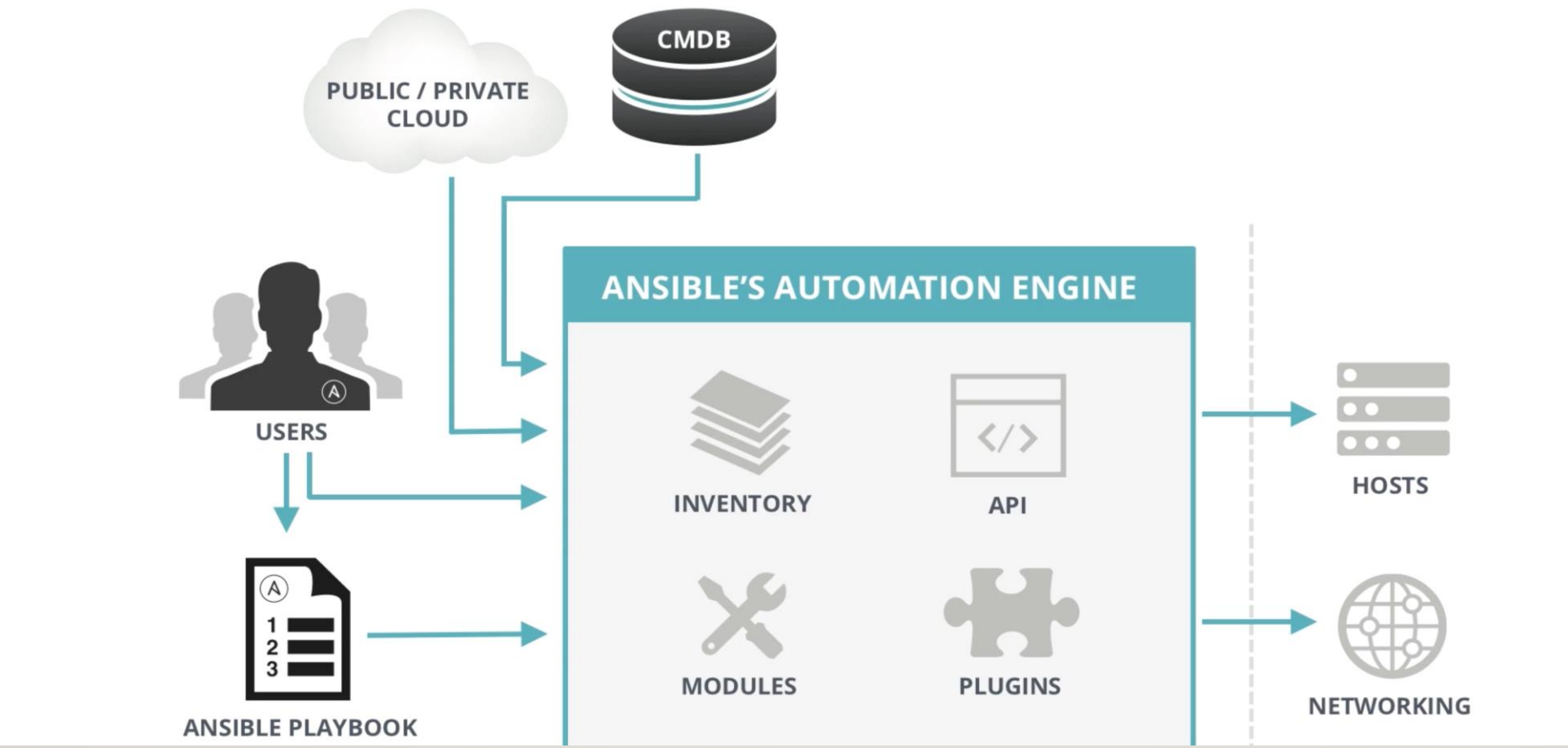
What is Ansible ?

- Ansible is an open source IT Configuration Management, Deployment & Orchestration tool. It aims to provide large productivity gains to a wide variety of automation challenges. This tool is very simple to use yet powerful enough to automate complex multi-tier IT application environments.

❖ **Configuration Management:**

- It establishes and maintains consistency of the product performance by recording and updating detailed information which describes an enterprise's hardware and software. Such information typically includes the versions and updates that have been applied to installed software packages and the locations and network addresses of hardware devices.
- **Application Deployment:** When you define your application with Ansible, and manage the deployment with Ansible Tower, teams are able to effectively manage the entire application life cycle from development to production.
- **Orchestration**: Ansible provides Orchestration in the sense of aligning the business request with the applications, data, and infrastructure. It defines the policies and service levels through automated workflows, provisioning, and change management. This creates an application-aligned infrastructure that can be scaled up or down based on the needs of each application.

Ansible Architecture



As you can see, in the diagram above, the Ansible automation engine has a direct interaction with the users who write playbooks to execute the Ansible Automation engine. It also interacts with cloud services and Configuration Management Database (CMDB)

ANSIBLE TERMS

- **Controller Machine**: The machine where Ansible is installed, responsible for running the provisioning on the servers you are managing.
- **Inventory**: An initialization file that contains information about the servers you are managing.
- **Playbook**: The entry point for Ansible provisioning, where the automation is defined through tasks using YAML format.
- **Task**: A block that defines a single procedure to be executed, e.g. Install a package.
- **Module**: A module typically abstracts a system task, like dealing with packages or creating and changing files. Ansible has a multitude of built-in modules, but you can also create custom ones.
- **Role**: A pre-defined way for organizing playbooks and other files in order to facilitate sharing and reusing portions of a provisioning.
- **Play**: A provisioning executed from start to finish is called a play. In simple words, execution of a playbook is called a play.
- **Facts**: Global variables containing information about the system, like network interfaces or operating system.
- **Handlers**: Used to trigger service status changes, like restarting or stopping a service.

Advantages of using Ansible

- **Simple:** Ansible uses a simple syntax written in YAML called ***playbooks***. YAML is a human-readable data serialization language. It is extraordinarily simple. So, no special coding skills are required and even people in your IT organization, who do not know what is Ansible can likely read a playbook and understand what is happening
- **Agentless:** Ansible is completely agentless. There are no agents/software or additional firewall ports that you need to install on the client systems or hosts which you want to automate
- **Powerful & Flexible:** Ansible has powerful features that can enable you to model even the most complex IT workflows. Ansible provides you with hundreds of modules to manage them. Together Ansible's capabilities allow you to orchestrate the entire application environment regardless of where it is deployed.
- **Efficient:** No extra software on your servers means more resources for your applications. Also, since Ansible modules work via JSON, Ansible is extensible with modules written in a programming language you already know. Ansible introduces modules as basic building blocks for your software. So, you can even customize it as per your needs. For e.g. If you have an existing message sending module which sends messages in plain-text, and you want to send images too, you can add image sending features on top of it.

Writing Ansible Playbooks

- Playbooks in Ansible are written in YAML format. It is a human-readable data serialization language. It is commonly used for configuration files. It can also be used in many applications where data is being stored.
- For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a “hash” or a “dictionary”.

❖ **Hosts And Users:**

- For each play in a playbook, you get to choose which machines in your infrastructure to target and which remote user to complete the tasks.
- Generally the hosts are a list one or more groups or host patterns, separated by colons. The remote user is just the name of the user account.

❖ **Variables:**

- Ansible uses variables which are defined previously to enable more flexibility in playbooks and roles. They can be used to loop through a set of given values, access various information like the host name of a system and replace certain strings in templates with specific values.
- Ansible already defines a rich set of variables, individual for each system. Whenever Ansible will run on a system, all facts and information about the system are gathered and set as variables.

WRITING ANSIBLE PLAY BOOKS

❖ **Tasks:**

- Tasks allow you to break up bits of configuration policy into smaller files. Task includes pull from other files. Tasks in Ansible go with pretty much the English meaning of it.
- E.g: Install <package_name>, update <software_name> etc.

❖ **Handlers:**

- Handlers are just like regular tasks in an Ansible playbook, but are only run if the Task contains a notify directive and also indicates that it changed something. For example, if a config file is changed, then the task referencing the config file may notify a service restart handler.

Playbook Example

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Thank You