**What is the role of a configuration management tool in DevOps?**

Automation plays an essential role in server configuration management. For that purpose, we use CM tools, they store information about versions and builds of the software and test ware and provide the traceability between software and test ware.

**What is the purpose of CM tools and which one you have used?**

Configuration Management tools' purpose is to automatize deployment and configuration of software on big number of servers. Most CM tools usually use agent architecture which means that every machine being manged needs to have agent installed. My favorite tool is one that uses agentless architecture - Ansible. It only requires SSH and Python. And if raw module is being used, not even Python is required because it can run raw bash commands. Other available and popular CM tools are Puppet, Chef, SaltStack.

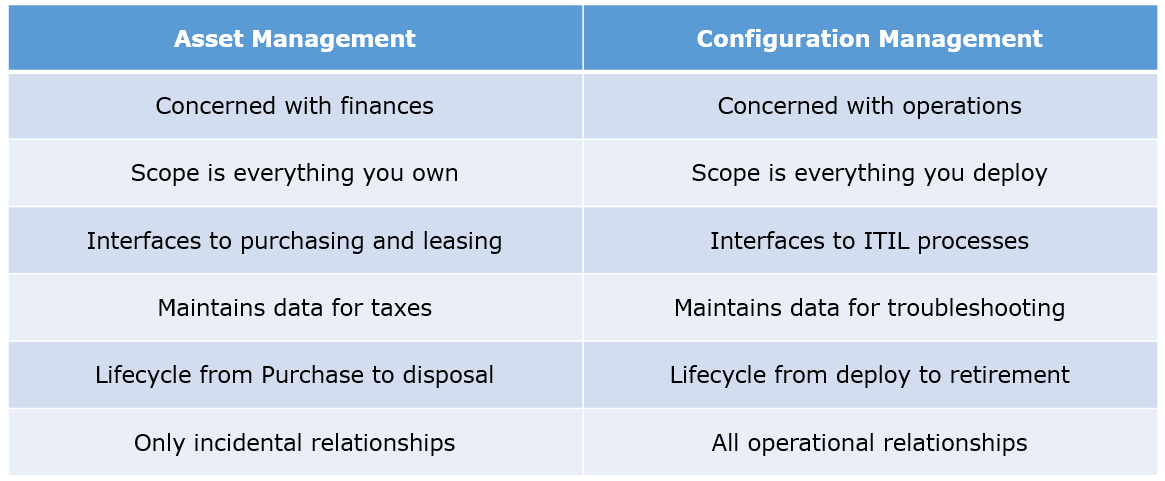
**What are the goals of Configuration management processes?**

The purpose of Configuration Management (CM) is to ensure the integrity of a product or system throughout its life-cycle by making the development or deployment process controllable and repeatable, therefore creating a higher quality product or system. The CM process allows orderly management of system information and system changes for purposes such as to:

* Revise capability,
* Improve performance,
* Reliability or maintainability,
* Extend life,
* Reduce cost,
* Reduce risk and
* Liability, or correct defects.

**What is the difference between Asset management and Configuration Management?**

Given below are few differences between Asset Management and Configuration Management:



**What is the difference between an Asset and a Configuration Item?**

According to me, you should first explain Asset. It has a financial value along with a depreciation rate attached to it. IT assets are just a sub-set of it.

Anything and everything that has a cost and the organization uses it for its asset value calculation and related benefits in tax calculation falls under Asset Management, and such item is called an asset.

Configuration Item on the other hand may or may not have financial values assigned to it. It will not have any depreciation linked to it. Thus, its life would not be dependent on its financial value but will depend on the time till that item becomes obsolete for the organization.

Now you can give an example that can showcase the similarity and differences between both:

1) Similarity: Server – It is both an asset as well as a CI.  
2) Difference: Building – It is an asset but not a CI.  
3) Document – It is a CI but not an asset

**What do you understand by “Infrastructure as code”? How does it fit into the DevOps methodology? What purpose does it achieve?**

Infrastructure as Code (IAC) is a type of IT infrastructure that operations teams can use to automatically manage and provision through code, rather than using a manual process.  
Companies for faster deployments treat infrastructure like software: as code that can be managed with the DevOps tools and processes. These tools let you make infrastructure changes more easily, rapidly, safely and reliably.

**Which among Puppet, Chef, SaltStack and Ansible is the best Configuration Management (CM) tool? Why?**

This depends on the organization’s need so mention few points on all those tools:

Puppet is the oldest and most mature CM tool. Puppet is a Ruby-based Configuration Management tool, but while it has some free features, much of what makes Puppet great is only available in the paid version.

Organizations that don’t need a lot of extras will find Puppet useful, but those needing more customization will probably need to upgrade to the paid version.

Chef is written in Ruby, so it can be customized by those who know the language. It also includes free features, plus it can be upgraded from open source to enterprise-level if necessary. On top of that, it’s a very flexible product.

Ansible is a very secure option since it uses Secure Shell. It’s a simple tool to use, but it does offer many other services in addition to configuration management. It’s very easy to learn, so it’s perfect for those who don’t have a dedicated IT staff but still need a configuration management tool.

SaltStack is python based open source CM tool made for larger businesses, but its learning curve is low.

**In terms of automation, discuss about the two differences between Puppet, Ansible and Chef**

**Push vs Pull Strategy**

* Chef: Chef client queries Chef server for the latest set of recipes (configuration instructions) that apply to the current node.
* Ansible uses a Push strategy. The machine where Ansible is installed uses SSH to copy files, remotely install packages, etc. on target machines the client machine requires no special setup outside of a working installation of Python 2.5+.
* Puppet nodes use a Pull strategy as nodes periodically check into a puppet master server to “pull” resource definitions.

**Server Nodes:**

* Chef infrastructure uses a Chef Server, the main hub where Chef propagates and stores system configuration information and policies and a Chef Client installed on every node being managed.
* Ansible has no concept of master/slave server, nor special agent executables to install: just proper SSH keys/credentials in order to connect to the nodes.
* It stores the details of the nodes : ip
* Connecton port
* Puppet infrastructure is made up of one or more “puppetmaster” servers, along with a special agent package installed on each client node.

**Language and Extensibility:**

* Chef: uses Ruby as programming language that is the authoring syntax for Chef cookbooks. Put it straight Chef lets you run wild with Ruby.
* Puppet uses its own DSL language which is a subset of Ruby. Thus, adding extra complex functionality is done through Ruby modules. That being said there's a stricter control on what you are doing with Ruby.
* Ansible playbooks are YAML files. In terms of extensibility, Ansible is built upon Python for which most organization will have some experience.

**Resources & Ordering**

* Chef: always executes recipes in the order you specify them. It will not arbitrarily reorder things. So, if you want one recipe to be run before another, just load them in that order.
* Ansible: The playbooks using a traditional top-to-bottom, as they appear in the file. This is more intuitive for developers coming from other languages.
* Puppet: Resources defined in a Puppet manifest are not applied in order of their appearance (ex: top->bottom). Instead resources are applied randomly, unless explicit resource ordering is used.

**Resource Dependency**

* Chef is  also able to declare dependencies between resources. Dependency failures are breakages in your dependency graph, which keep the current project’s pipeline from being able to ship safely. These failures are tracked because through Chef Automation.
* Ansible is merely a thin-wrapper for executing commands over SSH, so there is no resource dependency graph built internally.
* Puppet internally creates a directed graph of all of the defined resources along with the order they should be applied in. Puppet can even generate a graph file so that one can visualize everything that Puppet manages. On the other hand, building this graph is susceptible to “multiple resource definition” errors or conflicts due to circular dependencies

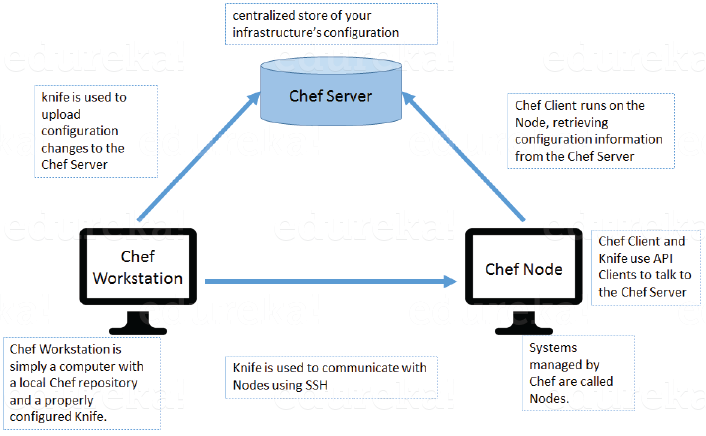
**Chef**

**What is Chef?**

It is a powerful automation platform that transforms infrastructure into code.

Chef is a tool for which you write scripts that are used to automate processes. What processes? Pretty much anything related to IT.

* **Chef Server:**The Chef Server is the central store of your infrastructure’s configuration data. The Chef Server stores the data necessary to configure your nodes and provides search, a powerful tool that allows you to dynamically drive node configuration based on data.
* **Chef Node:** A Node is any host that is configured using Chef-client. Chef-client runs on your nodes, contacting the Chef Server for the information necessary to configure the node. Since a Node is a machine that runs the Chef-client software, nodes are sometimes referred to as “clients”.
* **Chef Workstation:** A Chef Workstation is the host you use to modify your cookbooks and other configuration data.



**What is a resource in Chef?**

A Resource represents a piece of infrastructure and its desired state, such as a package that should be installed, a service that should be running, or a file that should be generated.  
You should explain about the functions of Resource for that include the following points:

* Describes the desired state for a configuration item.
* Declares the steps needed to bring that item to the desired state.
* Specifies a resource type such as package, template, or service.
* Lists additional details (also known as resource properties), as necessary.
* Are grouped into recipes, which describe working configurations.

**What do you mean by recipe in Chef?**

A Recipe is a collection of Resources that describes a particular configuration or policy.

A Recipe describes everything that is required to configure part of a system.  
After the definition, explain the functions of Recipes by including the following points:

* Install and configure software components.
* Manage files.
* Deploy applications.
* Execute other recipes.

**How does a Cookbook differ from a Recipe in Chef?**

“Recipe is a collection of Resources, and primarily configures a software package or some piece of infrastructure. A Cookbook groups together Recipes and other information in a way that is more manageable than having just Recipes alone.”

**What happens when you don’t specify a Resource’s action in Chef?**

when you don’t specify a resource’s action, Chef applies the default action.  
Now explain this with an example, the below resource:  
**file ‘C:\Users\Administrator\chef-repo\settings.ini’ do**  
**content ‘greeting=hello world’**  
**end**  
is same as the below resource:  
**file ‘C:\Users\Administrator\chef-repo\settings.ini’ do**  
**action: create**  
**content ‘greeting=hello world’**  
**end**

**Ohai**

Ohai is a tool that is used to collect system configuration data, which is provided to the chef-client for use within cookbooks. Ohai is run by the chef-client at the beginning of every Chef run to determine system state. Ohai includes many built-in plugins to detect common configuration details as well as a plugin model for writing custom plugins.

The types of attributes Ohai collects include but are not limited to:

* Operating System
* Network
* Memory
* Disk
* CPU
* Kernel
* Host names
* Fully qualified domain names
* Virtualization
* Cloud provider metadata

Attributes that are collected by Ohai are automatic level attributes, in that these attributes are used by the chef-client to ensure that these attributes remain unchanged after the chef-client is done configuring the node.

Ohai collects data for many platforms, including AIX, Darwin, Linux, FreeBSD, OpenBSD, NetBSD, Solaris, and any Microsoft Windows operating systems.

**Data Bags**

A data bag is a global variable that is stored as JSON data and is accessible from a Chef server. A data bag is indexed for searching and can be loaded by a recipe or accessed during a search.

Data bags are generally used to hold global information (“data bag items”) pertinent to your infrastructure that are not properties of the nodes themselves. In the most of the scenarios, you will continue to model most of your infrastructure using node attributes. Here are a few guidelines for whether a data bag item should be used to represent a piece of data:

* **If it is global across all of your infrastructure**, and you think you might need to change that item en-masse at some point. Examples: An external service API key which does not vary per environment; an office gateway’s external IP address; a license key.
* **If it needs to be encrypted.** Data bag items can not only be encrypted, but each item can have a different encryption key if desired. Encrypted data bag items give you the ability to secure sensitive information on the Chef server, so that no intruder could reveal your secrets even if they gained access to the Chef server. Also, no man-in-the-middle attack could reveal sensitive information by sniffing the traffic between the Chef client and the server; ciphertext is decoded only on the client. (This is less of a concern given that client-server communication is performed over SSL).
* **If it needs to be written to by another system** and we want to isolate the scope of the data that system can write to. Example: application release information which could eventually be written by a continuous integration pipeline.
* **If an external team needs to update limited pieces of information** and that team does not normally write Chef recipes. Example: the DBA that needs to occasionally modify a zse password or connection string

**Create a Data Bag**

A data bag can be created in two ways: using knife or manually. In general, using knife to create data bags is recommended, but as long as the data bag folders and data bag item JSON files are created correctly, either method is safe and effective.

knife can be used to create data bags and data bag items when the knife data bag subcommand is run with the create argument. For example:

$ knife data bag create DATA\_BAG\_NAME (DATA\_BAG\_ITEM)

### **Directory Structure**

All data bags are stored in the data\_bags directory of the chef-repo. This directory structure is understood by knife so that the full path does not need to be entered when working with data bags from the command line. An example of the data\_bags directory structure:

- data\_bags

- admins

- charlie.json

- bob.json

- tom.json

- db\_users

- charlie.json

- bob.json

- sarah.json

- db\_config

- small.json

- medium.json

- large.json

where admins, db\_users, and db\_config are the names of individual data bags and all of the files that end with .json are the individual data bag items.

**Data bag encryption:**

A data bag item may be encrypted using shared secret encryption. This allows each data bag item to store confidential information (such as a database password) or to be managed in a source control system (without plain-text data appearing in revision history). Each data bag item may be encrypted individually; if a data bag contains multiple encrypted data bag items, these data bag items are not required to share the same encryption keys

**Knife options:**

knife can encrypt and decrypt data bag items when the knife data bag subcommand is run with the create, edit, from file, or show arguments and the following options:

| **Option** | **Description** |
| --- | --- |
| --secret SECRET | The encryption key that is used for values contained within a data bag item. If secret is not specified, the chef-client looks for a secret at the path specified by the encrypted\_data\_bag\_secret setting in the client.rb file. |
| --secret-file FILE | The path to the file that contains the encryption key. |

**Chef-apply**

chef-apply is an executable program that runs a single recipe from the command line:

* Is part of the Chef development kit
* A great way to explore resources
* Is **NOT** how Chef is run in production

**Options**

This command has the following syntax:

$ chef-apply name\_of\_recipe.rb

This tool has the following options:

-e RECIPE\_TEXT, --execute RECIPE\_TEXT

Execute a resource using a string.

-l LEVEL, --log\_level LEVEL

The level of logging to be stored in a log file.

-s, --stdin

Execute a resource using standard input.

-v, --version

The version of the chef-client.

-W, --why-run

Run the executable in why-run mode, which is a type of chef-client run that does everything except modify the system. Use why-run mode to understand why the chef-client makes the decisions that it makes and to learn more about the current and proposed state of the system.

-h, --help

Show help for the command.

**Test Kitchen:**

Use Test Kitchen to automatically test cookbook data across any combination of platforms and test suites:

* Defined in a .kitchen.yml file
* Uses a driver plugin architecture
* Supports cookbook testing across many cloud providers and virtualization technologies
* Supports all common testing frameworks that are used by the Ruby community
* Uses a comprehensive set of base images provided by Bento

The key concepts in Kitchen are:

* A platform is the operating system or target environment on which a cookbook is to be tested
* A suite is the chef-client configuration, a run-list, and (optionally) node attributes
* An instance is the combination of a specific platform and a specific suite, with each instance being assigned an auto-generated name
* A driver is the lifecycle that implements the actions associated with a specific instance—create the instance, do what is needed to converge on that instance (such as installing the chef-client, uploading cookbooks, starting the chef-client run, and so on), setup anything else needed for testing, verify one (or more) suites post-converge, and then destroy that instance
* A provisioner is the component on which the chef-client code will be run, either using chef-zero or chef-solo via the chef\_zero and chef\_solo provisioners, respectively.

**Bento**

[Bento](https://github.com/chef/bento) is a project that contains a set of base images that are used by Chef for internal testing and to provide a comprehensive set of base images for use with Kitchen. By default, Kitchen uses the base images provided by Bento. (Custom images may also be built using Packer.)

**Testing Framework**

An integration test is an executable test that fails when the assumptions defined by the test are proven to be false. Each test is written in Ruby and must be located in the /tests directory within the cookbook to be tested.

The following frameworks are good options for building integration tests with Kitchen:

| **Test Framework** | **Description** |
| --- | --- |
| [Bats](https://github.com/sstephenson/bats) | bats (or Bash Automated Testing System) is an testing framework for Bash. Bats is also the default framework for Kitchen. |
| [Minitest](https://github.com/seattlerb/minitest) | A small, fast, testing framework. |
| [Rspec](http://rspec.info) | The primary testing framework for Ruby, using the words describe and it to express tests as conversation. bats, Minitest, Serverspec are all based on RSpec. |
| [Serverspec](http://serverspec.org) | RSpec-based tests for servers. |

## .kitchen.yml

Use a .kitchen.yml file to define what is required to run Kitchen, including drivers, provisioners, platforms, and test suites.

### **Syntax**

The basic structure of a .kitchen.yml file is as follows:

driver:

name: driver\_name

provisioner:

name: provisioner\_name

verifier:

name: verifier\_name

transport:

name: transport\_name

platforms:

- name: platform-version

driver:

name: driver\_name

- name: platform-version

suites:

- name: suite\_name

run\_list:

- recipe[cookbook\_name::recipe\_name]

attributes: { foo: "bar" }

excludes:

- platform-version

- name: suite\_name

driver:

name: driver\_name

run\_list:

- recipe[cookbook\_name::recipe\_name]

attributes: { foo: "bar" }

includes:

- platform-version

where:

* driver\_name is the name of a driver that will be used to create platform instances used during cookbook testing. This is the default driver used for all platforms and suites **unless** a platform or suite specifies a driver to override the default driver for that platform or suite; a driver specified for a suite will override a driver set for a platform
* provisioner\_name specifies how the chef-client will be simulated during testing. chef\_zero and chef\_solo are the most common provisioners used for testing cookbooks
* verifier\_name specifies which application to use when running tests, such as inspec
* transport\_name specifies which transport to use when executing commands remotely on the test instance. winrm is the default transport on Windows. The ssh transport is the default on all other operating systems.
* platform-version is a the name of a platform on which Kitchen will perform cookbook testing, for example, ubuntu-12.04 or centos-6.4; depending on the platform, additional driver details—for example, instance names and URLs used with cloud platforms like OpenStack or Amazon EC2—may be required
* platforms may define Chef server attributes that are common to the collection of test suites
* suites is a collection of test suites, with each suite\_name grouping defining an aspect of a cookbook to be tested. Each suite\_name must specify a run-list, for example:
* run\_list:
* - recipe[cookbook\_name::default]
* - recipe[cookbook\_name::recipe\_name]
* Each suite\_name grouping may specify attributes as a Hash: { foo: "bar" }
* A suite\_name grouping may use excludes and includes to exclude/include one (or more) platforms. For example:
* excludes:
* - platform-version
* - platform-version # for additional platforms

**Chef solo:**

chef-solo is a command that executes chef-client in a way that does not require the Chef server in order to converge cookbooks. chef-solo uses chef-client’s Chef local mode, and does not support the following functionality present in chef-client / server configurations:

* Centralized distribution of cookbooks
* A centralized API that interacts with and integrates infrastructure components
* Authentication or authorization

chef-solo is an open source version of the chef-client that allows using cookbooks with nodes without requiring access to a Chef server. chef-solo runs locally and requires that a cookbook (and any of its dependencies) be on the same physical disk as the node.

**Chef Zero:**

Chef Zero is a simple, easy-install, in-memory Chef server that can be useful for Chef Client testing and chef-solo-like tasks that require a full Chef Server.

**Chef-apply**

chef-apply is an executable program that runs a single recipe from the command line:

* Is part of the Chef development kit
* A great way to explore resources
* Is **NOT** how Chef is run in production

This command has the following syntax:

$ chef-apply name\_of\_recipe.rb

**In chef if you wanted to create a cookbook or recipe to configure apache what you would do?**

First, I will check with the supermarket, if I find the required cookbook then I will customize it according to my requirements, and try to see if there are any other dependencies in that cookbook and try to download those dependencies also, and also check the resources in the recipe, if there is anything to be customized then I will customize those recipes too.

**What resources u use to configure the apache?**

First, I will create a cookbook and then by default I will get a default.rb file in the recipes directory of the cookbook. Then in that recipes I will include the resources like package (to install apache), and services to start the apache. So, these are the resources used to configure apache.

**How do you change the values in httpd.conf (modifying the configuration files)?**

First thing if I wanted to change the values in some configurations files like for apache its httpd.conf, then I will change the values in the attributes, or if I wanted to do something dynamically I will use templates so that I can provide the place holders that are filled by the node attributes when the recipe runs.

**What is attribute precedence.**

Before the chef-client run all the attributes collected by ohai, and other attributes mentioned in the cookbooks and environments are rebuilt. After that all the attributes are then merged and applied to a node according to the attribute precedence.

Attribute types:

1. Default (it resets at the start of chef-client and has lowest precedence),
2. force-default (it ensures that the attributes mentioned in the cookbook takes precedence over a default attribute set by a role or environment),
3. normal (it has higher attribute precedence than a default one),
4. override (it has highest precedence than default, force default, normal attributes. A cookbook should ensure that it should use over ride attribute only when required),
5. force override (it ensures that attribute mentioned in the cookbook takes precedence over override attributes set by role or environment),
6. automatic (it contains data that is identified by ohai , it cannot be modified and always has highest precedence)

**Ansible**

**What is Ansible module?**

Modules are considered to be the units of work in Ansible. Each module is mostly standalone and can be written in a standard scripting language such as Python, Perl, Ruby, bash, etc. One of the guiding properties of modules is idempotency, which means that even if an operation is repeated multiple times e.g. upon recovery from an outage, it will always place the system into the same state.

**What are playbooks in Ansible?**

Playbooks are Ansible’s configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process. Playbooks are designed to be human-readable and are developed in a basic text language.  
At a basic level, playbooks can be used to manage configurations of and deployments to remote machines.

**How do I see a list of all of the ansible\_ variables?**

Ansible by default gathers “facts” about the machines under management, and these facts can be accessed in Playbooks and in templates.

To see a list of all the facts that are available about a machine, you can run the “setup” module as an ad-hoc action:  
**Ansible -m setup hostname**This will print out a dictionary of all of the facts that are available for that particular host.

**How can I set deployment order for applications?**

WebLogic Server 8.1 allows you to select the load order for applications. See the Application MBean Load Order attribute in Application. WebLogic Server deploys server-level resources (first JDBC and then JMS) before deploying applications. Applications are deployed in this order: connectors, then EJBs, then Web Applications. If the application is an EAR, the individual components are loaded in the order in which they are declared in the application.xml deployment descriptor.

**Can I refresh static components of a deployed application without having to redeploy the entire application?**

Yes, you can use weblogic.Deployer to specify a component and target a server, using the following syntax:  
java weblogic.Deployer -adminurl http://admin:7001 -name appname -targets server1,server2 -deploy jsps/\*.jsp

**How do I turn the auto-deployment feature off?**

The auto-deployment feature checks the applications folder every three seconds to determine whether there are any new applications or any changes to existing applications and then dynamically deploys these changes.

The auto-deployment feature is enabled for servers that run-in development mode. To disable auto-deployment feature, use one of the following methods to place servers in production mode:

* In the Administration Console, click the name of the domain in the left pane, then select the Production Mode checkbox in the right pane.
* At the command line, include the following argument when starting the domain’s Administration Server:  
  -Dweblogic.ProductionModeEnabled=true
* Production mode is set for all WebLogic Server instances in each domain.

**When should I use the \_stage option?**

Set -external\_stage using weblogic.Deployer if you want to stage the application yourself, and prefer to copy it to its target by your own means.

**Devops Example : Deploying Applications with Ansible**

Ansible is a lightweight, extensible solution for automating your application provisioning. Ansible has no dependencies other than Python and SSH. It doesn’t require any agents to be set up on the remote hosts and it doesn’t leave any traces after it runs either. It allows you to significantly simplify our operations by creating easy YAML based playbooks. It’s good for configuration automation, deployments and orchestration.

**Components of Ansible**

**Playbooks:** Ansible playbooks are a way to send commands to remote computers in a scripted way. Instead of using Ansible commands individually to remotely configure computers from the command line, you can configure entire complex environments by passing a script to one or more systems.

Ansible playbooks are written in the YAML data serialization format. If you don't know what a data serialization format is, think of it as a way to translate a programmatic data structure (lists, arrays, dictionaries, etc) into a format that can be easily stored to disk. The file can then be used to recreate the structure at a later point. JSON is another popular data serialization format, but YAML is much easier to read.

Let's look at a basic playbook that allow us to install a web application (nginx) in a multiple hosts :

hosts: webservers  
tasks:  
- name: Installs nginx web server  
apt: pkg=nginx state=installed update\_cache=true  
notify:  
- start nginx

handlers:  
- name: start nginx  
service: name=nginx state=started

**The hosts file :** (by default under /etc/ansible/hosts) this is the Ansible Inventory file, and it stores the hosts, and their mappings to the host groups (webservers ,databases etc)

[webservers] 10.0.15.22  
# example of setting a host inventory by IP address.  
# also demonstrates how to set per-host variables.

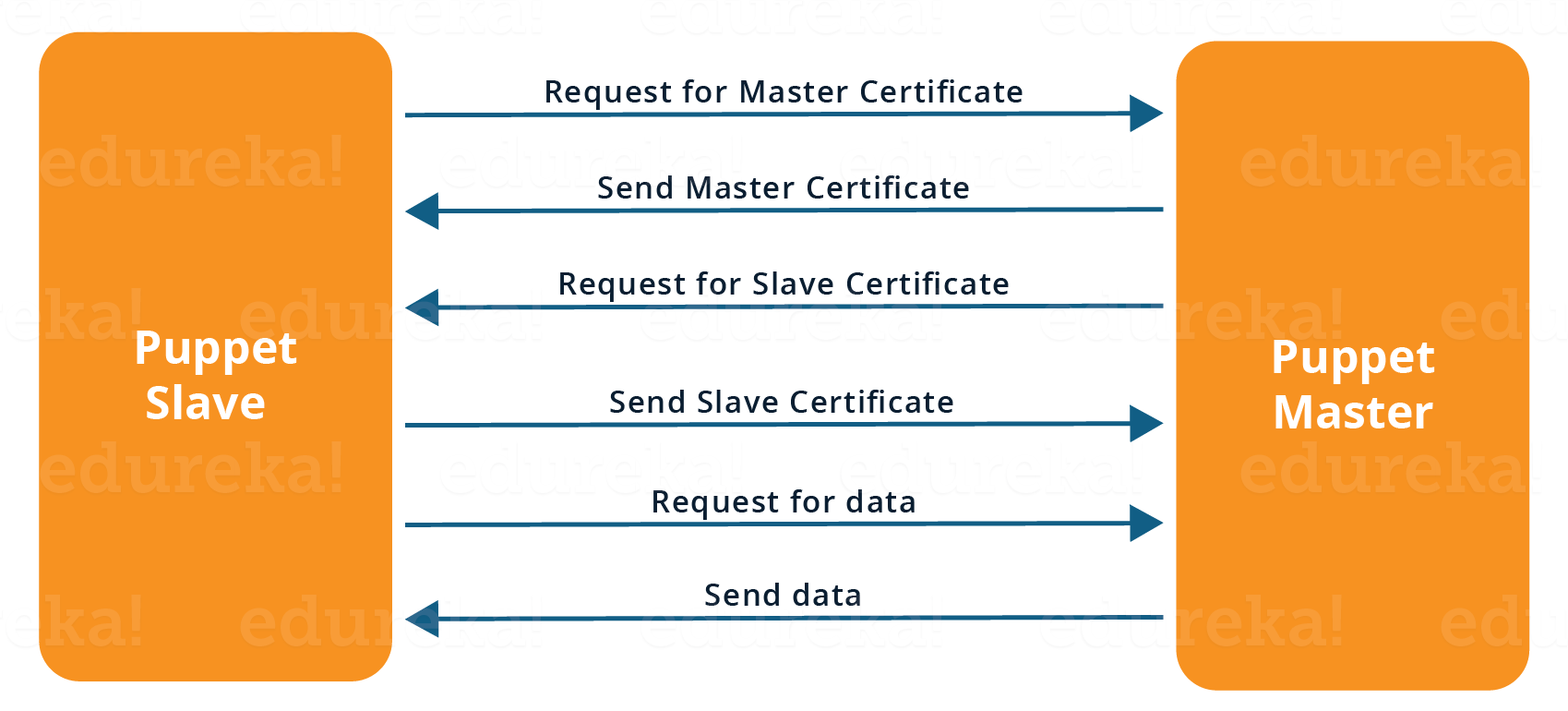
[repository\_servers] example-repository  
#example of setting a host by hostname. Requires local lookup in /etc/hosts  
# or DNS.  
[dbservers] db01

**The SSH key :** For the first run, we'll need to tell ansible the SSH and Sudo passwords, because one of the thing that the common role does is to configure passwordless sudo, and deploy a SSH key. So in this case ansible can execute the playbook’s commands in the remote nodes (hosts ) and deploy the web application nginx.

**Puppet**

**What is Puppet?**

It is a Configuration Management tool which is used to automate administration tasks.  
Now you should describe its architecture and how Puppet manages its Agents. Puppet has a Master-Slave architecture in which the Slave has to first send a Certificate signing request to Master and Master has to sign that Certificate in order to establish a secure connection between Puppet Master and Puppet Slave as shown on the diagram below. Puppet Slave sends request to Puppet Master and Puppet Master then pushes configuration on Slave.  
Refer the diagram below that explains the above description.



**Before a client can authenticate with the Puppet Master, its certs need to be signed and accepted. How will you automate this task?**

The easiest way is to enable auto-signing in puppet.conf.  
Do mention that this is a security risk. If you still want to do this:

* Firewall your puppet master – restrict port tcp/8140 to only networks that you trust.
* Create puppet masters for each ‘trust zone’, and only include the trusted nodes in that Puppet masters manifest.
* Never use a full wildcard such as \*.

**Describe the most significant gain you made from automating a process through Puppet.**

I automated the configuration and deployment of Linux and Windows machines using Puppet. In addition to shortening the processing time from one week to 10 minutes, I used the roles and profiles pattern and documented the purpose of each module in README to ensure that others could update the module using Git. The modules I wrote are still being used, but they’ve been improved by my teammates and members of the community

**Which open source or community tools do you use to make Puppet more powerful?**

Changes and requests are ticketed through Jira and we manage requests through an internal process. Then, we use Git and Puppet’s Code Manager app to manage Puppet code in accordance with best practices. Additionally, we run all of our Puppet changes through our continuous integration pipeline in Jenkins using the beaker testing framework.

**What are Puppet Manifests?**

Every node (or Puppet Agent) has got its configuration details 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.

You can write a manifest in Puppet Master that creates a file and installs apache on all Puppet Agents (Slaves) connected to the Puppet Master.

**What is Puppet Module and How it is different from Puppet Manifest?**

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.

It is considered best practice to use Modules to organize almost all of your Puppet Manifests.  
Puppet programs are called Manifests which are composed of Puppet code and their file names use the .pp extension.

**What is Facter in Puppet?**

“Facter gathers basic information (facts) about Puppet Agent 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.”

**How will you Manage if Puppet Module Fails?**

Most of the time the module which are going to write is well tested,

Initially after writing manifests I mean after developing the Puppet Module I will do puppet apply on the same node if it is lab node, or if I have a client I can run puppet agent -t –noop, that will not actually apply the module to the node but a kind of test whether its working or not.

**How you made available the Puppet Modules?**

We use bit bucket to maintain everything, initial script, python script, as Puppet has its own repo and bit bucket. From my machine, I will just once I finished testing everything I pushed to the master server from then Application manager/ My manager will take care of it.