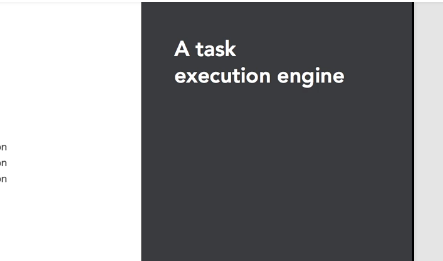
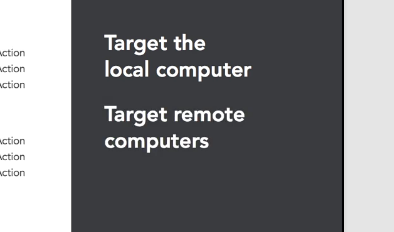
- Ansible, at is core, is a task execution engine. It exists to provide a method for operators, engineers, developers, hobbyists, or whomever, to easily define one or more actions to be performed on one or more computers.

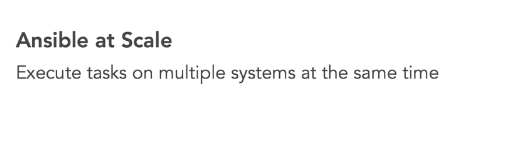


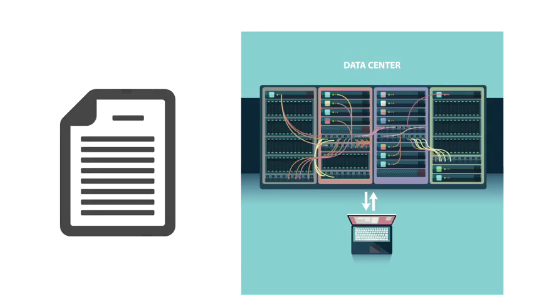
 This capability represents a step beyond simply just logging into each computer in question, and manually typing out the command.

These tasks can target the local system Ansible is running from, as well as other systems Ansible can reach over the network.

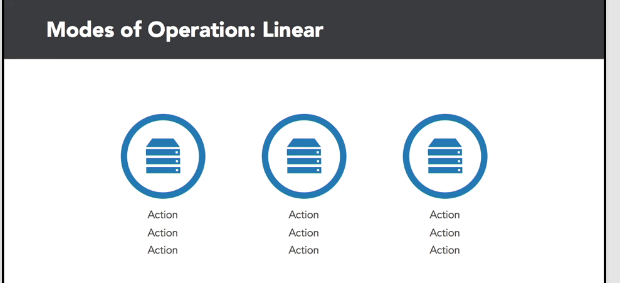


Arguably, the ability to manage remote systems is the most important aspect of Ansible. When combined with the ability to express the tasks to be performed in a simple to read text file, Ansible provides a reusable and repeatable system for managing a fleet of infrastructure.Ansible is designed to scale beyond a small handful of systems to manage. In order to efficiently address whole fleets of systems, Ansible must be able to work on multiple machines at the same time.

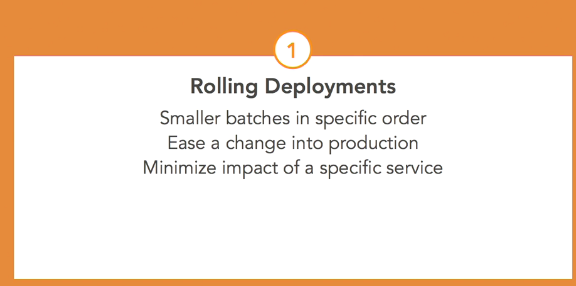




The default operation model allows Ansible to linearly execute a list of tasks on multiple machines at once. As the set of machines complete one task, they are given the next task to complete. This is the default, or linear strategy, and allows for data generated in an action on a particular machine to be used as input data for a later task on a different machine.

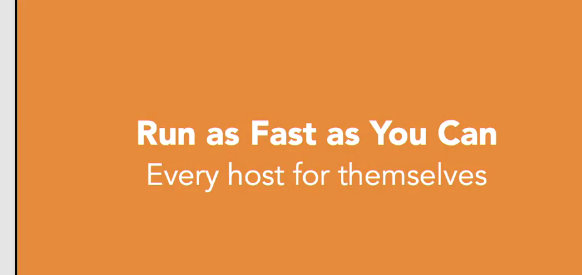


 There are however certain times when systems need to be worked on in smaller batches in a specific order.



This strategy is useful for rolling changes out in small batches into production, or minimizing the impact of an upgrade by disrupting only small portions of a service at any one time. This is the serial strategy. Instead of linearly walking through the task list, one or more hosts in batches are walked through the task list before looping back to the beginning for the next set of hosts.

A third operation strategy was introduced with Ansible 2.0, in which machines complete tasks as fast as they individually can, without waiting for the rest of the hosts to complete a given task.

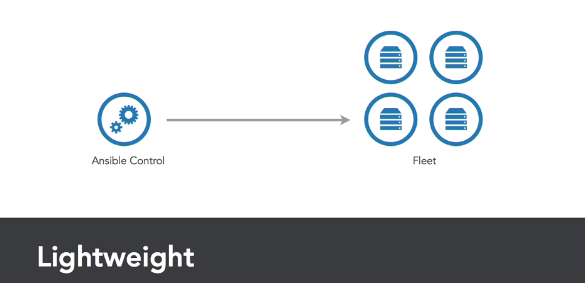


This is known as the free strategy, and is used mostly to reduce interruption of a specific machine or service, rather than the fleet as a whole. When using the strategy, it is not possible for one host to depend on task-generated data from a different host.

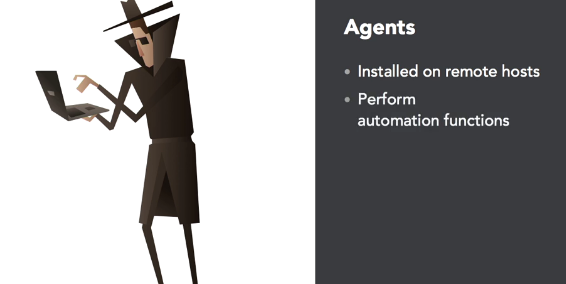
The way humans direct Ansible to accomplish tasks is with YAML formatted files. YAML neatly blends human readability with machine parsing. Perfectly suited as an operator to machine interaction language for Ansible.

2

**Low cost light weight**

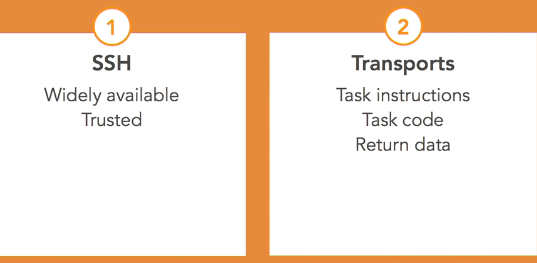


- Ansible is a lightweight fleet management system. There only needs to be one control system where the Ansible engine is installed. The system that will manage itself and other systems in a fleet. The Ansible engine has minimal installation requirements. Python, with a few additional libraries, is all that's required for the core engine. Agent software is not required on the hosts to be managed. Agents in other automation systems are installed on the remote hosts, and told to perform functions on the hosts through its own communication protocol



This requires configuration management before you can perform any configuration management.

The actions to be performed may themselves require additional software on the target hosts, but Ansible can take care of the installation of those requirements as well.



communication with target hosts defaults to SSH, a widely available and trusted communication protocol. Instructions for an action to perform,and the necessary software to perform the desired tasks, are transported over SSH.

Data from the action is returned over SSH as well. If I can SSH to a target system, I can manage it with Ansible.



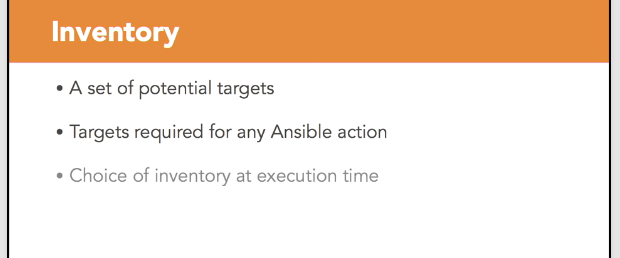
As there is no central authority of what the state is or should be, Ansible then can be run from any system that has access rights to the target hosts.

Getting Started:

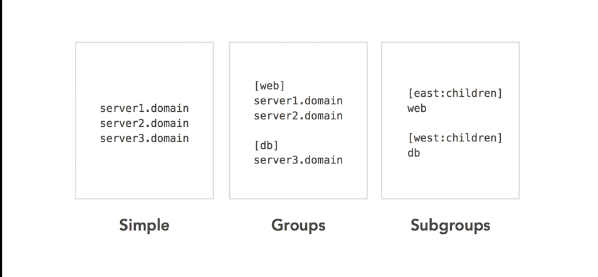
- Ansible needs to be explicitly installed on the control machine,which is the machine that will be processing all of the task execution directions.

Parts of Ansible

- Inventories in Ansible provide a set of potential target hosts to execute tasks on. Ansible must have a target to do anything. Thus the inventory is the most fundamental building block. Everything Ansible does is tied to a host from an inventory



An inventory is a collection of hosts optionally sorted into groups potentially with variable data. A simple inventory is just an uncategorized list of host names or IP addresses. Without any extra details, when Ansible executes a task on a host from the inventory, it will connect to the name of the host. As a fleet grows, it becomes advantageous to categorize hosts within groups.

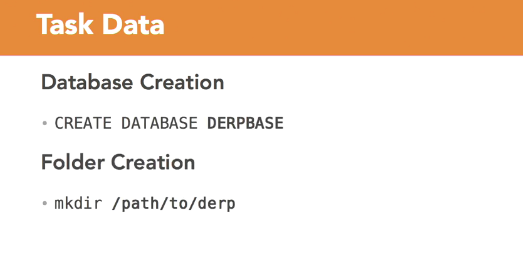


Groups provide an easy to reference target to a set of hosts. Groups can include other groups building up a hierarchy of one's fleet. Hosts are grouped for many reasons such as purpose, locality or operating system

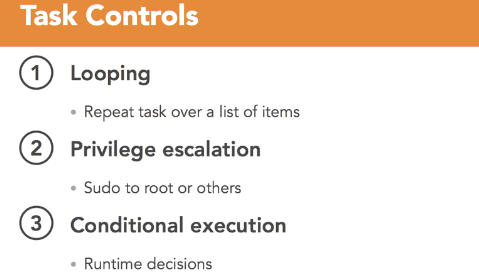
- The actions Ansible takes on target hosts are called tasks.Tasks are a descriptive bit of YAML code that a developer writes to provide just enough data and controls for Ansible to be able to complete the desired action



Data can be considered arguments to an executable script such as the name of a database to create, or the path of a directory to create. Data relates to the code that will execute on the remote host.



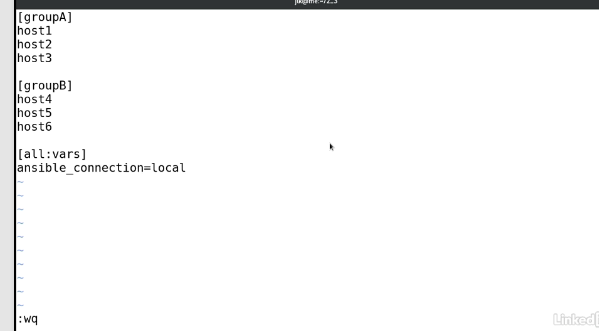
Controls on the other hand relate to the task itself and can influence things such as looping over the task multiple times with different data provided, whether or not to use privilege escalation on the target host, or whether the task should be executed at all.



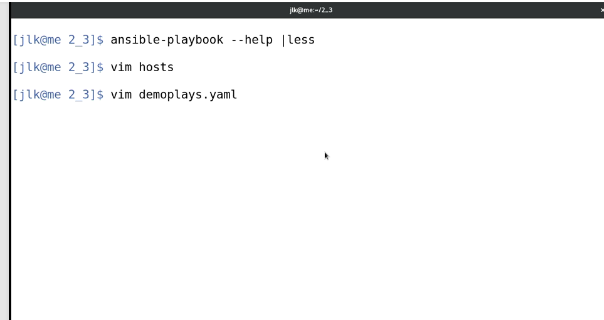
Playbooks

-Playbooks are YAML formatted files that collect one or more plays. Plays are one or more tasks linked to the hosts that they are to be executed on

To demonstrate playbooks, first I need to create an inventory.  The inventory will just be a set of hosts in a couple of groups.



Each host is uniquely named so that we can distinguish it in the output. In addition, we are going to define a variable for all the hosts that instructs

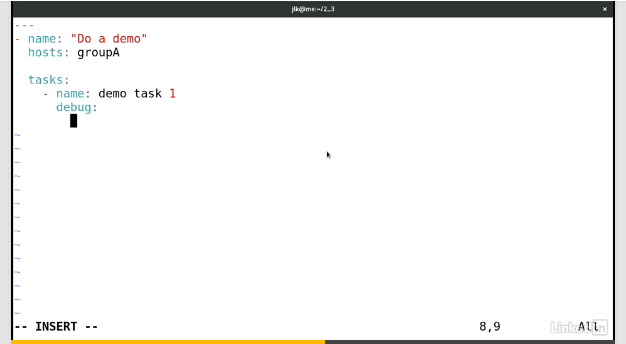


Ansible to connect to these fake hosts locally. I'll need a playbook file, which I'll create with my text editor named demoplays.yaml. YAML files are simply text files, so any text editor will do. I happen to use Vim. After the traditional three dashes to indicate a YAML file, I'll create the first play.

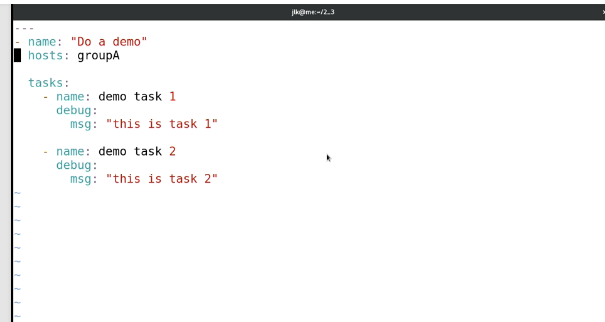
A **traditional three dashes to indicate a YAML file**, is needed to create a play.

**Plays require a name, a host pattern, and some tasks**.

  In the example below,I'll call this first play Do a demo. For a host pattern, I'll use the group A set of hosts I just created.Next, I'll create a section for tasks. Like plays, tasks are expressed as a YAML list and each task should have a name. I'll give my first task a name of demo task 1. Tasks require a module, and for demonstration purposes I'll make use of the debug module.

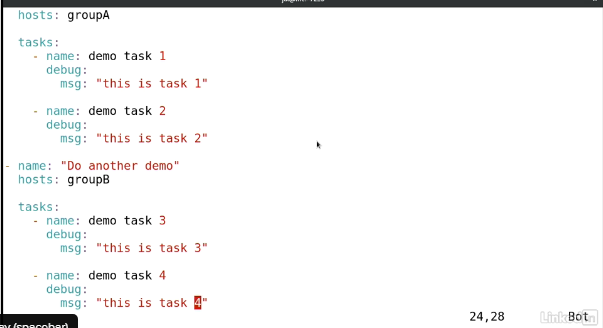


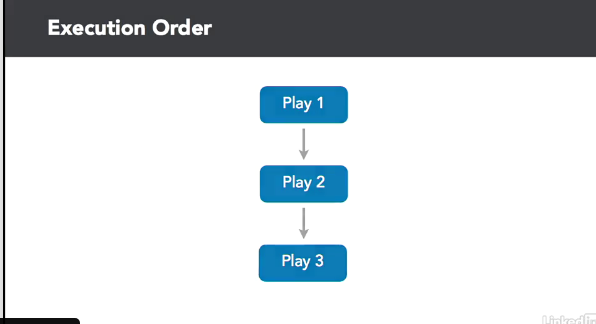
The debug module can take a message argument to display provided text on the screen. I'll display this is task 1.

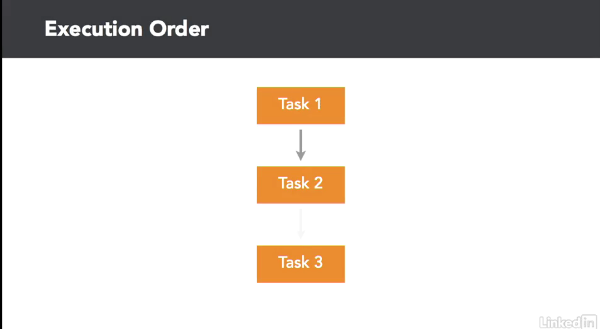
I'll duplicate this task for a second task, altering the name and the output string.

 Next, I'll add a second play with a similar name. Using hosts from groupB, I'll copy all the tasks from the first play, and again, I'll alter the task names and the output strings to indicate their location in the file.

First I'll copy this entire play. Change the name of the play to distinguish.Change the host to groupB. And change the task details to indicate that it's from the second play.

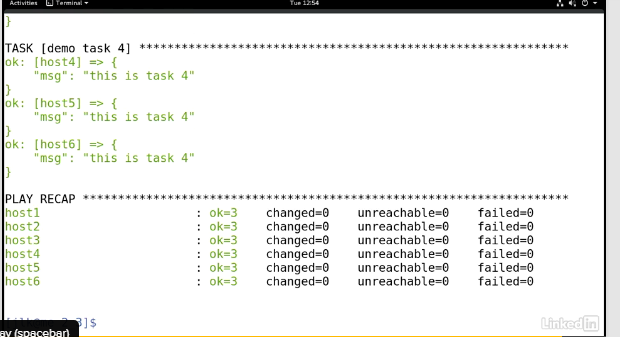


Plays are executed from top to bottom and tasks are also executed from top to bottom So my play and task numbering will make sense. 

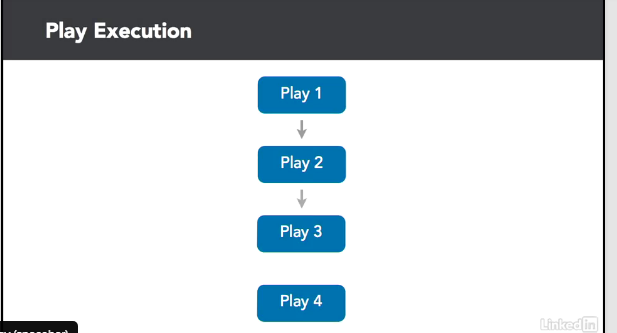


I'm going to save my playbook and execute it to show the output. I need to tell Ansible playbook the inventory file to use and the playbook file to parse.Inventory is provided with the -i argument and the playbook is a freeform argument. Each play is displayed on screen and as each host completes each task, the result is displayed on screen as well. An implicit task is show, the setup task, which gathers facts about each host.



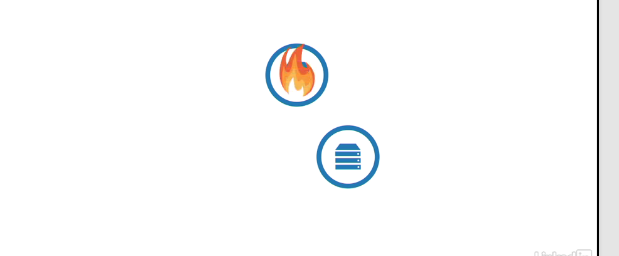


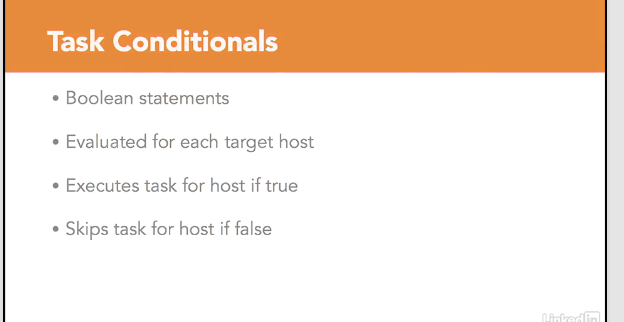
Control task and play behaviour



- Playbook execution flows through each play in a playbook until all plays are complete. Execution will abort if a fatal error state has been met.

Fatal error states can include events like every host of a play encountering an error, or if a play is unable to start because all hosts in the target set have already encountered an error. This is because Ansible removes hosts that have encountered an error from the potential target set.





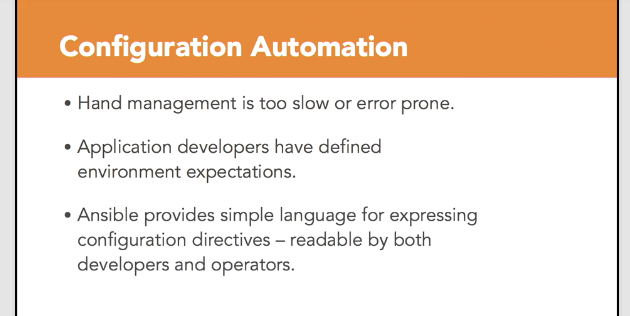
 However, if I use a task conditional and instruct Ansible to only run the second task on the second host, I can see how only some of the hosts fail, and that execution will continue on to the next play.

A task conditional allows me to provide a Boolean statement that is evaluated for each host. If it evaluates to true, the task is executed for the host. Otherwise, it is skipped.

What is Ansible Good for?

Not so long ago, systems were often managed by hand, and expected to live a long and productive life. Applications would come and go, but the system ran on. There was heavy division between application developers,and the system operators running the systems the applications would run on.

 One major shift forward has been the practice of configuration management automation to meet increasing infrastructure demands.



 Ansible performs well as a configuration management tool, with directives expressed in a language that both developers and operators can understand. With configuration management handled, attention shifted to provisioning of infrastructure.

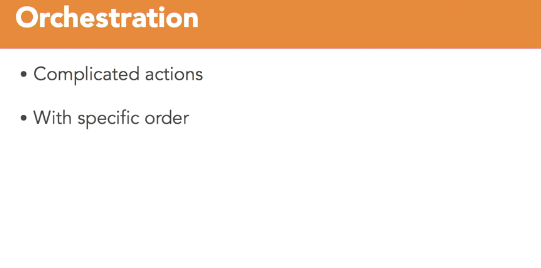
Infrastructure is often whole physical machines, or hand managed virtual machines, provisioned from limited capacity

 Now, we've entered the era of cloud computing, where infrastructure is on-demand.Development and operations work closely together under the common term, DevOps.

 Ansible, again, is a well-suited tool for this era. It has modules to create infrastructure, as well as modules to assert the configuration on that infrastructure. The same language can be used to express both parts of the puzzle.

Repeatable experimentation is possible, simply by proposing code changes to a repository. Developers and operators can collaborate on the code that defines the infrastructure necessary for the applications, which can be executed to create the infrastructure and configuration for the experiment.

Orchestration

 -

Ansible is well suited to perform orchestration automation.Orchestration, is a complicated set of coordinated actions.

 To demonstrate orchestration with Ansible Playbooks, first you need an inventory,  Using an editor,you can open a new file named web app, and create three hosts in a web group and two hosts in the lb group to represent a pair of load balancers.

You can also add an all group variable to instruct Ansible to connect locally to these fake hosts.



You will also need a Playbook and and you can call it web-deploy.yaml.



 The example scenario is a three-node web cluster fronted by a pair of load balancers. To minimize downtime, you will want to disable the node to be upgraded on the load balancer as you upgrade it, allowing the remaining nodes to take on the traffic. I can accomplish this all with a single play which we will name deploy.

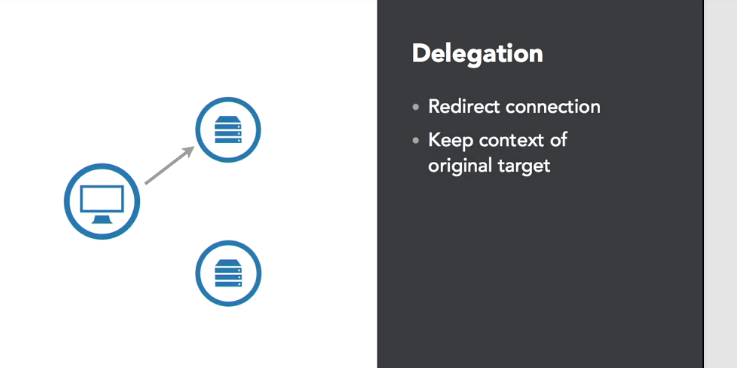


I only need to target the web hosts where the website runs. I'm going to use serial execution mode with a batch size of 1,so that each host is individually taken through all the tasks before operating on the next host so that I can roll this change out. For my tasks, first I need to disable the web node on the load balancer. Again, I'm going to use the debug module to simulate this action.

In our message, we will use a variable inventory host name which references the specific host we're operating on. This variable goes inside must dash brackets to indicate a template. This action needs to happen on the load balancer itself so we can use a task directive of delegate to.

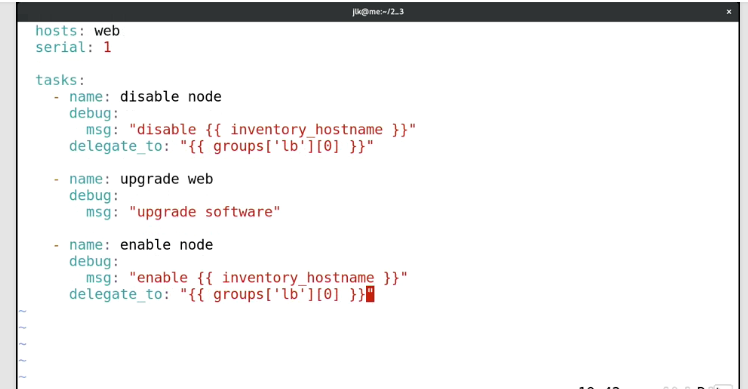


Delegation tells Ansible to change where it will connect, but otherwise act as if it were working on the node it would have targeted.



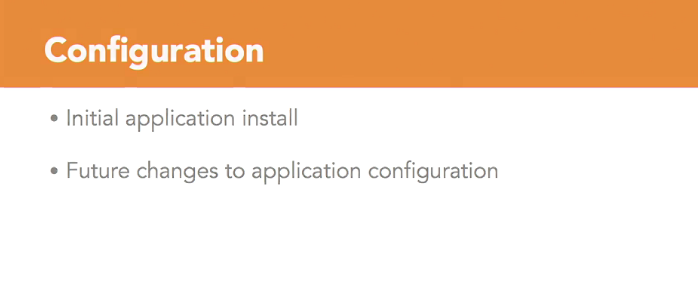
For delegation, we are addressing the first host of the load balancer group. Since my load balancers are a synced pair, we only need to make the change on one load balancer. The other one will pick up the change. I'm using templating syntax to reference that host by accessing the group's variable and the lb group within. The first host is represented at the zero index point. Now that the node is disabled, I can upgrade the software.

Again, I'll use the debug task to simulate the upgrade. This could be a series of tasks which Ansible will walk through using a runtime variable for which version to deploy. But for my demo, a single task will suffice. Finally, I re-enable the node, once again delegating the task to the load balancer host. Now I can save this file and leave the editor so that I can execute the Playbook.



If I look closely at the output, I can see our task delegation in action and yet the inventory host name variable still references the original host, giving context for what to do on the load balancer. I can also see that host one completes all the tasks and then host two starts through. If I did not use serial, every host would have been removed from the load balanceras the first task, resulting in complete outage of my web app.

Configuration Management



- [Instructor] Ansible is well-suited to perform configuration management. Successful configuration management is handling not just the initial application install, but any future changes to the configuration as well.