Episode #43 - 19 Minutes With Ansible (Part 1/4)

About Episode - Duration: 19 minutes, Published: Jan 13, 2015

In this episode series, we will be looking at Ansible, which is an easy to use configuration management and orchestration tool. My goal for this series, is to show you what Ansible is, how it works, and the steps to get going on your own.

Download: mp4 (https://s3.amazonaws.com/sysadmincasts.com/static/videos/43-19-minutes-with-ansible-part-1-4.mp4) or webm (https://s3.amazonaws.com/sysadmincasts.com/static/videos/43-19-minutes-with-ansible-part-1-4.webm)

Get notified about future content via the mailing list (/get-notified), follow @jweissig_ (https://twitter.com/jweissig_) on Twitter for episode updates, or use the RSS feed (/feed.rss).

Links, Code, and Transcript

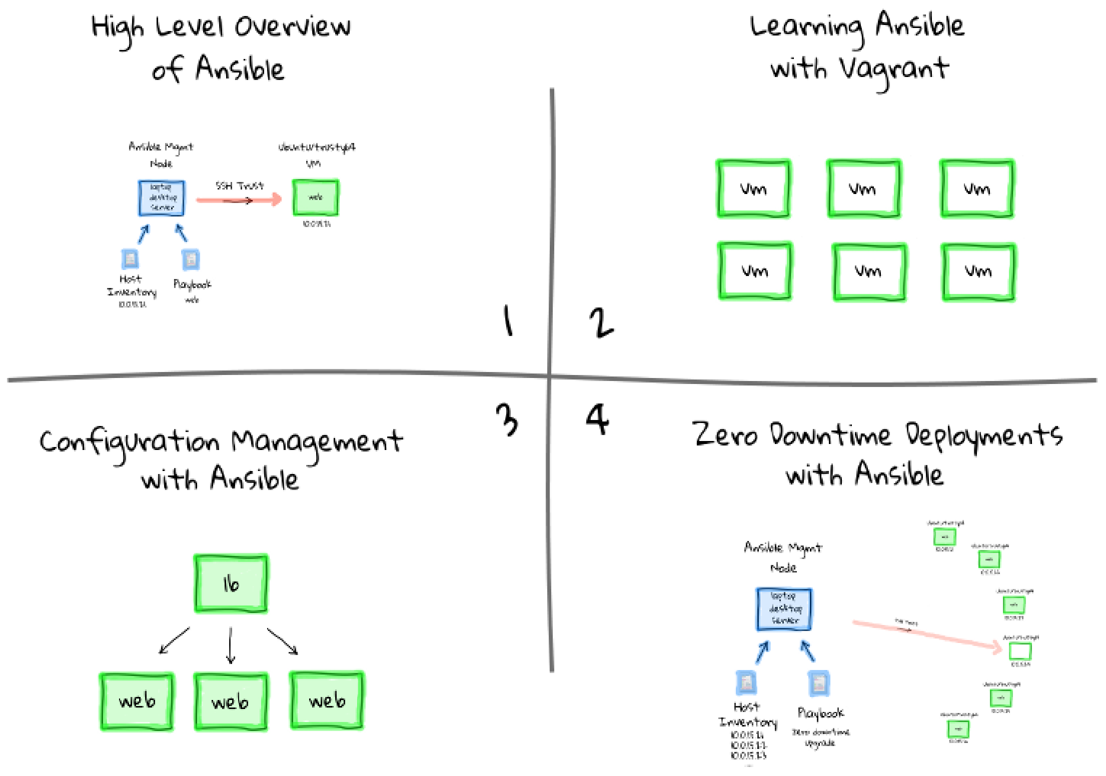
- Ansible is Simple IT Automation (http://www.ansible.com/home)
- Episode #45 - Learning Ansible with Vagrant (Part 2/4) (/episodes/45-learning-ansible-with-vagrant-part-2-4)
- Episode #46 - Configuration Management with Ansible (Part 3/4) (/episodes/46-configuration-management-with-ansible-part-3-4)
- Episode #47 - Zero-downtime Deployments with Ansible (Part 4/4) (/episodes/47-zero-downtime-deployments-with-ansible-part-4-4)
- Ansible Documentation (http://docs.ansible.com/index.html)
- Ansible - Module Index (http://docs.ansible.com/modules_by_category.html)
- Ansible - Playbooks (http://docs.ansible.com/playbooks.html)
- Ansible - Best Practices (http://docs.ansible.com/playbooks_best_practices.html)
- How Twitter Uses Ansible (https://www.youtube.com/watch?v=fwGrKXzocg4)
- Orchestration with Ansible at Fedora Project (http://www.slideshare.net/AdityaPatawari/ansible-33223245)
- Fedora - ansible.git (https://infrastructure.fedoraproject.org/cgit/ansible.git)
- Guardian Developer Blog: Delivering Continuous Delivery, Continuously (http://www.theguardian.com/info/developer-blog/2015/jan/05/delivering-continuous-delivery-continuously)

19 Minutes With Ansible

In this episode series, we will be looking at Ansible, which is an easy to use configuration management and orchestration tool. My goal for this series, is to show you what Ansible is, how it works, and the steps to get going on your own.

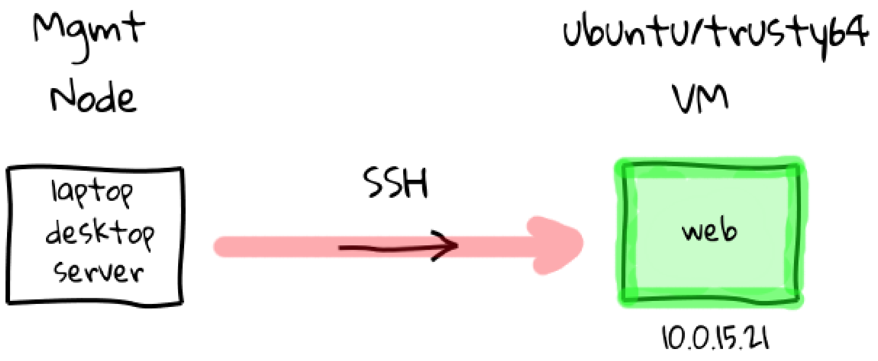
Series Overview

I have split this episode series into four parts, mainly because I wanted to give a bit of a crash course on what configuration management is, along with how Ansible fits into that picture. In this episode, part one, we will look at a high level overview of what Ansible is, how it works, and what you might want to use it for via some example use-case scenarios. I am mostly just going to show you what Ansible is via diagrams in this episode, along with some external reference links, no command line bits just yet. We are also going to look at configuration management in general, and how this is an improvement over doing things manually, in the hopes that this will give Ansible some context. In part two, we are going to setup a bunch of virtual machines using Vagrant. We will work through installing Ansible from scratch, how it operates at the command line, what the configuration files look like, and how communication works between nodes. Then it part three, we will look at using Ansible for configuration management tasks, by taking generic virtual machines from our Vagrant environment, and turning them into a web cluster, using haproxy and some nginx web servers. Finally, in part four, we will use Ansible to do a zero-downtime rolling software deployment across a cluster of web nodes. The idea is that we can deploy code across a fleet of machines, without any downtime by using Ansible to orchestrate the various tasks for us, this will come in really handy for continues deployment workflows. Throughout each of these episodes, I will give you all the code, and commands used, in the hopes you will duplicate my results, as I think it is a great way of learning. So, now that you have a general idea of what this series is about, lets jump into part one.


High Level Overview of Ansible
Ansible Mgmt Node
laptop desktop server
SSH Trust
ubuntu/trusty64 vm
web
10.0.15.21
Host Inventory 10.0.15.21
Playbook web
1
Learning Ansible with Vagrant
vm
vm
vm
vm
vm
vm
2
3
Configuration Management with Ansible
lb
web
web
web
4
Zero Downtime Deployments with Ansible
Ansible Mgmt Node
laptop desktop server
Host Inventory 10.0.15.21 10.0.15.22 10.0.15.23
Playbook zero downtime upgrade
web

Before Configuration Management

I thought it might make sense to give you a brief overview of my take on the problem Ansible is trying to solve. Imagine for a minute that you have two machine. The one of the left could be your laptop, desktop, or even a server you have access too, we will call it the management node. The one on the right, is a fresh Ubuntu virtual machine that we have just installed and booted, in reality this could be bare metal physical box, or even a cloud instance. Lets say we wanted to turn this freshly installed Ubuntu machine into a web server, say for example we are running Apache hosting a website, a rails app, or something similar. How would we go about doing that? Well, we could do this manually by sshing into the machine, using its IP address 10.0.15.21, and running the commands to install your application stack, editing the configuration files by hand, and finally copying over our application code. Once you are all done, you disconnect, and the machine is configured a working. This is pretty common practice, but this is also pretty manual work, and even if we have the steps documented somewhere, each machine is generally its own little snowflake depending on who installed it. This manual work can quickly compound if you need to do this across tens, or hundreds of machines. It is also a real pain when one of the machines dies, because we are not really sure how it was created, or how we go about recreating it quickly. Do not get me wrong, I have manually installed many hundreds of machines, so I know this issue well, and it is extremely unpleasant when something important dies, and you are not really sure how to recreate it quickly. In that, you have a general idea of what was on there, but you engage in the same type of manual work that created this mess in the first place, potentially wasting many hours of your time on drop work type tasks. Constantly installing packages, dependencies, checking to see if it works, installing more packages, and tweaking things manually until the service is restored. It turns into this vicious cycle, and we are recreating the exact scenario that caused this mess in the first place.



Enter Ansible

There has to be a better way right? Well, this is where configuration management tools come in. At a basic level, they are tools designed to automate away much of this manual work. Saving you time, reducing stress, and generally improving the process of creating machines in a timely manner. So, lets chat about Ansible. Ansible is a free and open-source tool, mainly use on UNIX-like machines, which is directly targeted at solving this type of manual work problem. Lets start our example over again, and have a look at how we would solve this problem using Ansible. We start with our two machines again, the one of the left could be our laptop, desktop, or even a server, but this time call it the Ansible management node, because this is where we will install the Ansible software. The one on the right, is our freshly install Ubuntu machine, and this can be anywhere from our local machine running a VM, a physical box, or a cloud instance. Once Ansible is installed on the management node, you will typically need two configuration files, something called a host inventory, and the other a playbook. The host inventory, is basically just a listing of hostnames, or IP addresses, for machines that we want to manage, and how they should be group together. In the example here, our Ubuntu box has the address 10.0.15.21, so we would just add this address to our host inventory, likely under a web group, since we want its end state to be a web server.

Hosts Inventory

Actually, let me just show you what an example host inventory looks like, since this will likely make much more sense. So, the host inventory INI file is basically just a listing of hosts that you want to manage with Ansible, you can group them together under arbitrary headings too. You use these brackets to create a group, then put the group name inside, in this case we have a load balances group, and it has three hosts assigned to it. For hosts you want to manage with Ansible, you can use their fully qualified hostnames names, short hostnames, or IP addresses. You can see down here in the webservers group, we added our Ubuntu virtual machines IP address from the diagrams. You can also call these groups anything you want, lets change the name from webserver, to just web. Now that you know what the host inventory looks like, lets just jump back to the diagrams, and chat about playbooks. I should mention that you do not need to know too much about this right now, we will cover these files heavily in the other episode parts, for right now, it is just important to know these configuration files exist.

```
[loadbalancer]
haproxy-01.example.com
haproxy-02
10.0.15.15

[web]
web1
web2
10.0.15.21

[testing]
demo.example.com
```

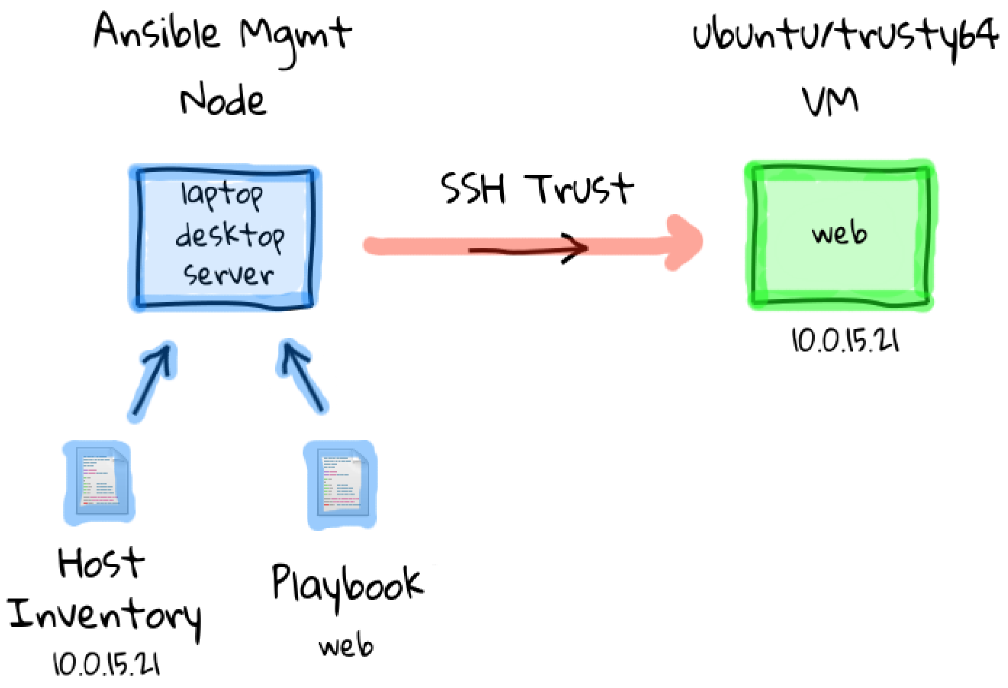
Playbooks

Playbooks are configuration files that outline tasks that should be performed against hosts in the hosts inventory. In our example today, we are setting up a web server on our freshly installed Ubuntu box, so we would likely want to install apache, or nginx, update the web server configuration files, deploy our application content, restart the server, etc. Basically, all of the tasks we would have done manually, just documented in a configuration file called a playbook, so that Ansbile knows what to do. Playbooks allow you to define each of these steps, in a simple, and quick to understand format, that pretty much anyone not familiar with Ansible can read and understand. Lets quickly have a look at an example web server playbook. The configuration format is called YML, but you do not really need to understand this to work with Ansible, as it is pretty straight forward. So, here we have defined our hosts group which we want to run this playbook against, this should look familiar from our hosts inventory. We have this sudo line here, this allows Ansible to connect as a non-root user, then elevate to root for tasks like installing packages, adding users, or deploying configuration files. Next we have a series of tasks that we want to run against each host in hosts inventory web group. You can see here that we want to install nginx, update our custom nginx configuration file, deploy our website code via a git pull from version control, and finally make sure nginx is started. This pretty closely follows what we would manually do in real life. Again, do not worry about understanding all of this right now, we are going to look at many live examples throughout this episode series. I should also mention, that in our example today, we are going to be mainly looking at using playbooks, these are configure files that outlines tasks that should be run, but you can also run Ansible in an ad-hoc mode, basically using it as a parallel command execution engine across a fleet of machines. We will look at this heavily in parts two, three, and four of this episode series, but this ad-hoc mode can be thought of as a smart parallel ssh engine, at least that is what I think of it as, just with a lot more cool functionality built in.

```
---
- hosts: web
  sudo: yes

  tasks:

  - name: install nginx
    apt: name=nginx state=installed update_cache=yes

  - name: write our nginx.conf
    template: src=templates/nginx.conf.j2 dest=/etc/nginx/nginx.conf
    notify: restart nginx

  - name: deploy website content
    git: repo=https://github.com/jweissig/episode-47.git
         dest=/usr/share/nginx/html/
         version=release-0.01

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

  handlers:

  - name: restart nginx
    service: name=nginx state=restarted
```

With Configuration Management

So, now that you have a basic idea of the prerequisites for Ansible, those being that you have to install Ansible onto a management node, define a hosts inventory, and have some playbooks defined. Lets, have a look at how we would turn our Ubuntu virtual machine into a web server using Ansible. You run Ansible Playbook on the management node, it looks through the playbook that you have defined as a command argument, and notices that we are targeting nodes in the web group. Ansible then reads in the hosts inventory to find nodes assigned to the web group. At this point, Ansible is ready to get to work, so it will remotely connect via ssh to the defined machines, typically you will want to have some type of ssh trust established via pre-shared keys, so that you do not have to enter the password all the time. Ansible will then start to step through the playbook tasks, one task at a time, going through them sequentially, from top to bottom, just like you would have done if logging in manually. So, it installs the packages, updates the configuration files, deploys our website code by using git, and finally starts our web service. When Ansible is happy that everything worked as expected you will get a status report saying that everything is good.



So, that is basically the default Ansible workflow in a nutshell. But, you will soon notice that Ansible is a pretty flexible tool, and there are exceptions to pretty much everything I have shown you so far. For example, the hosts inventory can be a database if you have thousands of hosts, you also do not always need playbooks, you can run ad-hoc commands too. You can swap out ssh for a queue type system, if you find that performance is an issue for your number of hosts. But, for these examples today, I thought I would just show you what I think is the default mode of operation, when first playing around with Ansible.

How is Ansible is different?

You might be wondering how Ansible is different from other configuration management tools, like Puppet and Chef for example, so I thought it would be useful to mention a couple items, before we move onto more complex examples. Probably the most glaring difference is that Ansible pushes the configuration out to each managed machine via ssh. Ansible only requires that you install the Ansible software onto a management node, and that the remote machines is running ssh with python installed, which every major distribution come with by default, there are no remote agents that you need to install, and everything is done via ssh from the management node. This make getting going with Ansible very easy, and upgrades are a snap, because you only need to update the Ansible install on one machine.

The Ansible documentation is absolutely fantastic, you can read the manual pages and quickly understand how Ansible functions, I honestly cannot say enough good things about them. You will often hear to term, batteries included, when reading about Ansible, that is because there are over 250 helper modules, or functions included with Ansible. These allow you to construct playbooks to smartly add users, install ssh keys, tweak permissions, deploy code, install packages, interact with cloud providers for things like launch instances or modifying a load balancer, etc. Each module has a dedicated page on the Ansible documentation site, along with detailed examples, and I have found this a major bonus to working with Ansible. I should mention that, even though Ansible is using ssh to connect to these remote machines, your playbooks and ad-hoc commands will almost always be using these 250 plus modules to smartly do things. I guess what I am trying to say, is that Ansible is not simply running remote commands like a shell script would do, for automating package installs, adding users, etc. There is tons of logic built into these modules, and I encourage you to check these manual pages, just to get a sense of what can be done.

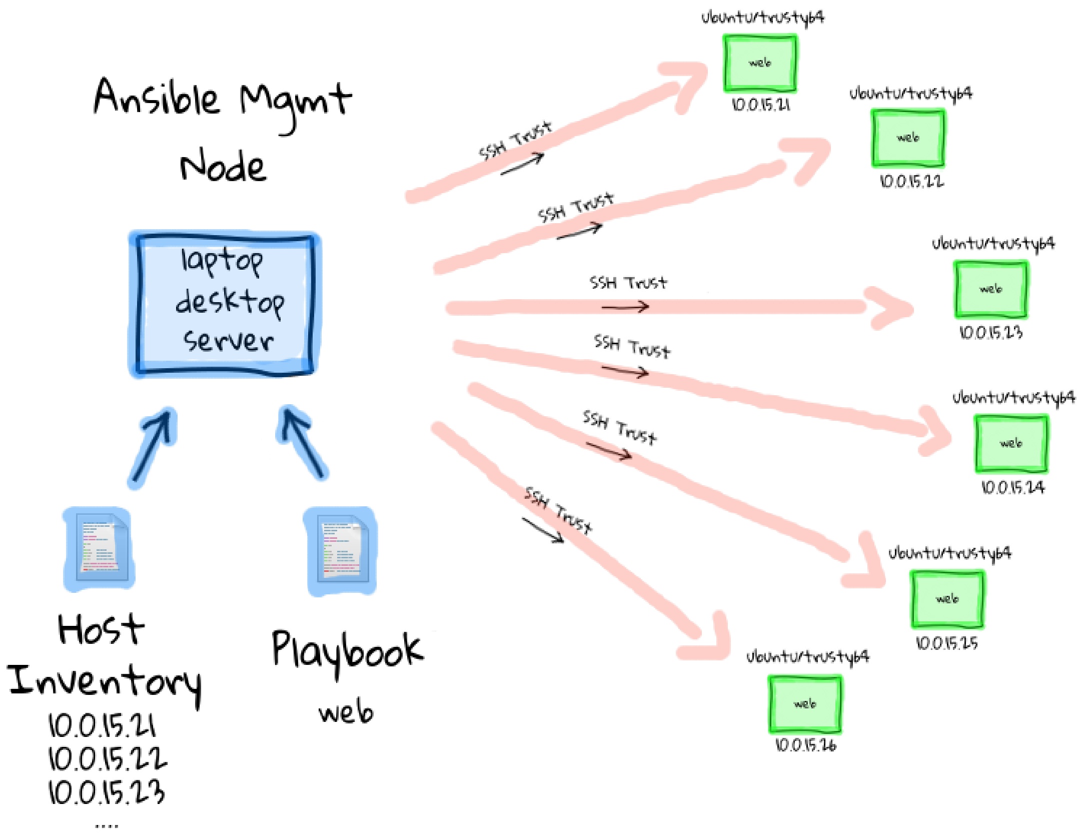
One of the biggest pros to using Ansible, is that since playbooks are basically configuration files, you do not need to be an experience configuration management expert, the bar is set pretty low for getting going, and you can quickly read and understand existing playbooks. I think is a big reason Ansible is quickly becoming popular, in that you are not really writing code, just working with something that looks like a configuration file.

Ansible is also great for both ops and dev, because you do not have to give out root. Ansible is just using ssh to log into the remote machines, so there is clear separation of duties if needed, based on what account Ansible is using. One added bonus of using any type of configuration management tool, including Ansible, is that these configuration playbooks are self documenting. I guess what I mean, is that you have a clear pictures of how these machines are made, and how to re-make them, if needed, because you have a step by step outline in the playbook. One last pro for Ansible before we move on, there is a published Ansible Best Practices guide, and it is absolutely fantastic in the advice it offers. Things like how to layout your files, using version control, naming, how to keep things simple. This is some of the best published, and put together advice, I have seen out of all the popular configuration management software, by far.

Taking it a Step Further

Okay, so lets crank out a couple more use-case scenarios, now that we have a basic understanding of what Ansible is, how it works, and why you might want to use it for automating your infrastructure bits. Once you have your playbooks figured out, you can use them as a starting point to launch any number of new machines, or turn existing machines, into something that you want. Lets say for example, that instead of launch just a single new machine, maybe you need a bunch of extra capacity in your web tier, lets say for a holiday rush. So, we launch a bunch of new generic virtual machines, and to manage these new machines with Ansible, we just add them to our hosts inventory like we did before, except this time we add all six machines. This might be where you want to use a database, or some type of cloud module to poll the new instances that you created, so that you do not have to do this manually. Once all of these addresses are added into the hosts inventory, lets re-use our web playbook. Then lets follow the same process as before, run ansible playbook on our management node, that pulls in our web playbook, and that playbook references our updated hosts inventory. You can probably guess what is going to happen next. Again, Ansible connects to each of these machines in parallel through an ssh trust, from there it starts to run our defined tasks from the playbook, things like installing a web server, deploying our configuration file, starting the service, etc. Finally, each of these tasks completes, we have our installed and ready to use machines, along with a summary from Ansible of how everything went.

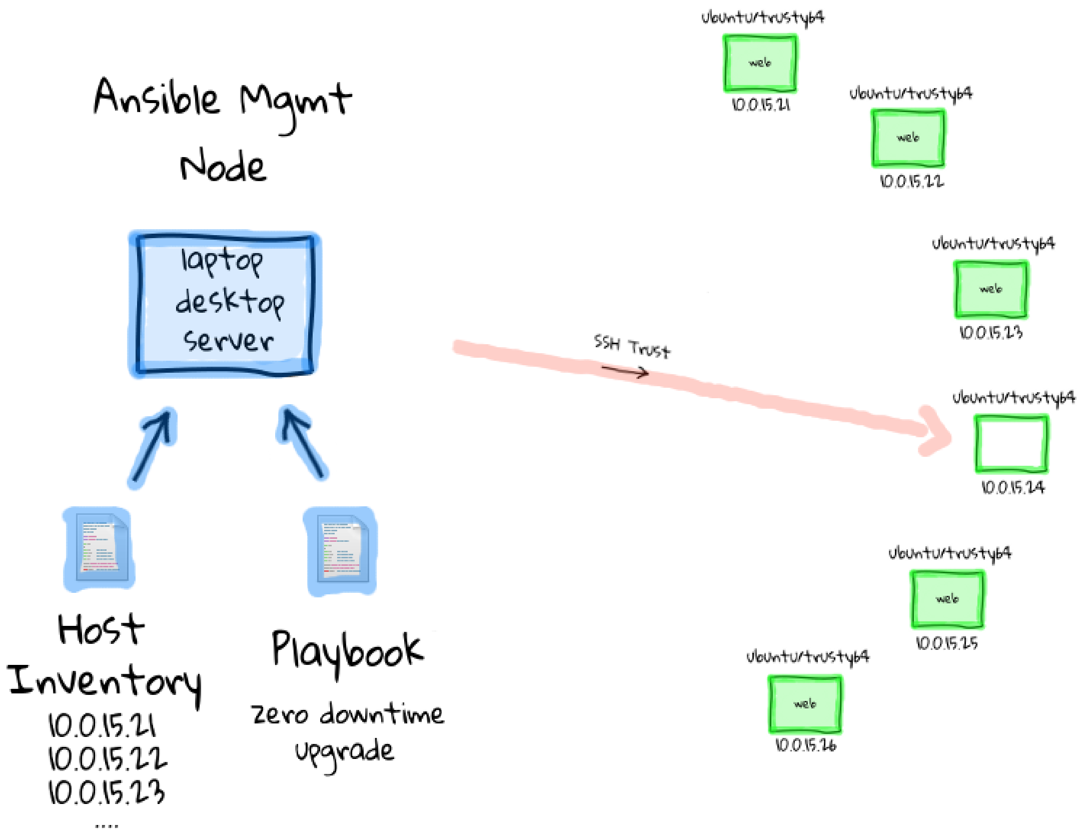
So, we can just reuse our previous playbook, where we deployed one machine, and use that same thing to deploy an entire set of new machines. The limit of how many machines you can talk to via ssh, is really based off your management nodes resources, things like bandwidth, and cpu. There are real world use cases where Ansible manages many thousands of nodes, people like Twitter, and the Fedora project both use Ansible. If you are looking for a little more details about how Twitter uses Ansible, check out this Youtube video, and I always find it interesting to see how other poeple are using these tools. There is also a pretty decent slide deck about how the Fedora project is using Ansible too, and the Fedora project actually publishes their hosts inventory, and playbooks, which are absolutely amazing resources for seeing how larger projects are implementing Ansible at scale. As usual, all of these links are in the episode notes below, so do not worry about writing these down. One thing that I forgot to mention about Ansible, is that once you run these commands to configure a machine, Ansible stop caring about them. It is not going to check on them again, unless you tell it to, and this is a difference from other configuration management tools, which typically have an agent sitting on each machine, that checks in once and a while. You can configure Ansible to push changes out via cron, or some type of continuous integration system, if you are interested in that though. Personally, I kind of like this type of system, because you can force things out quickly via a playbook run, rather than having machine eventually catch up. This also allows for one off ad-hoc type upgrades, say for example you needed to patch OpenSSL, like in the Heartbleed drama. You just create a new playbook configuration file that outlines your changes, and I have found these runs are especially powerful for these common sysadmin tasks. I think this feeds nicely into our next and final example.



Zero-downtime Rolling Deployments

Up until this point we have used something called a web playbook, but lets switch gears for a minute, and look at using a different playbook, remember that these are just configuration files that you can easily edit, or create entirely new once, since they are just files. So, lets change that from a web playbook to something called the zero downtime upgrade playbook, we will actually cover this in part four of our episode series. The idea for this example is that this is how many people are implementing continues deployments across their infrastructure. You will typically have a load balancer in front of all these web boxes, then you will use some type of tool, in our case Ansible, to notify the load balancer that this machine should be taken out of the pool, then you will update the software, and add it to the pool again. I will show you a live demo of this in part four, along with all the code, so that you can do this on your own. It is pretty neat that we can use Ansible to orchestrate this entire sequence of events for us.

I should mention that Ansible has this concept of pre and post task handlers, so we will define our tasks as deploying new software to a machine, but the pre task will be to notify the load balancer that we want to remove this node from server, then we will update the software on our node, and finally the post task handler will add it back into the pool. Lets just look at how this would work across our cluster of machines here. So we run Ansible Playbook again, this time it pulls in our zero-downtime upgrade playbook, then we pull in our hosts inventory, again you could use some type of database if you have a very dynamic environment. Finally, we use our ssh trust based on a preshared key, to connect to each box, one after another, removing it from the load balancer, updating the software, re-adding it to the load balancer, then moving onto the next node.



I have actually done these types of tasks manually before, and it really sucks, because there are so many moving parts, and errors are really easy to make. This is a major limiting factor to releasing software updates out into production, because it takes a long time, and you are fearful of breaking something. I actually found a really good article which talks about this, and I thought I would end the episode on it. The Guardian released a blog posting recently about how they went from 25 manual deployments of their site per year, to over 24,000 highly automated deployments, using an in house build continues deployment tool. This is pretty much exactly the use case that I just talked about. Although, The Guardian is not using Ansible, there struggle really rings a bell for what many people are going though. If you have the time, I highly suggest reading their story, as I think it illustrates much of the thinking, rational, and work that does into deploying something like this.

Okay, so that pretty much wraps up with episode, and hopefully I have wet your appetite for the types of problems that Ansible is able to solve. The remaining episode parts should be out shortly, sometime in the next week, so check back soon.