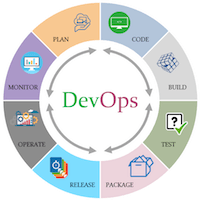
DevOps Tutorial

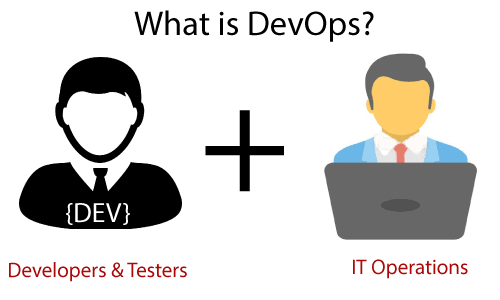


The DevOps is the combination of two words, one is **Development** and other is **Operations**. It is a culture to promote the development and operation process collectively.

The DevOps tutorial will help you to learn DevOps basics and provide depth knowledge of various DevOps tools such as **Git, Ansible, Docker, Puppet, Jenkins, Chef, Nagios**, and **Kubernetes**.

What is DevOps?

The DevOps is a combination of two words, one is software Development, and second is Operations. This allows a single team to handle the entire application lifecycle, from development to **testing, deployment**, and **operations**. DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators.



DevOps promotes collaboration between Development and Operations team to deploy code to production faster in an automated & repeatable way.

DevOps helps to increase organization speed to deliver applications and services. It also allows organizations to serve their customers better and compete more strongly in the market.

DevOps can also be defined as a sequence of development and IT operations with better communication and collaboration.

DevOps has become one of the most valuable business disciplines for enterprises or organizations. With the help of DevOps, **quality**, and **speed** of the application delivery has improved to a great extent.

DevOps is nothing but a practice or methodology of making "**Developers**" and "**Operations**" folks work together. DevOps represents a change in the IT culture with a complete focus on rapid IT service delivery through the adoption of agile practices in the context of a system-oriented approach.

DevOps is all about the integration of the operations and development process. Organizations that have adopted DevOps noticed a 22% improvement in software quality and a 17% improvement in application deployment frequency and achieve a 22% hike in customer satisfaction. 19% of revenue hikes as a result of the successful DevOps implementation.

Why DevOps?

Before going further, we need to understand why we need the DevOps over the other methods.

* The operation and development team worked in complete isolation.
* After the design-build, the testing and deployment are performed respectively. That's why they consumed more time than actual build cycles.
* Without the use of DevOps, the team members are spending a large amount of time on designing, testing, and deploying instead of building the project.
* Manual code deployment leads to human errors in production.
* Coding and operation teams have their separate timelines and are not in synch, causing further delays.

DevOps History

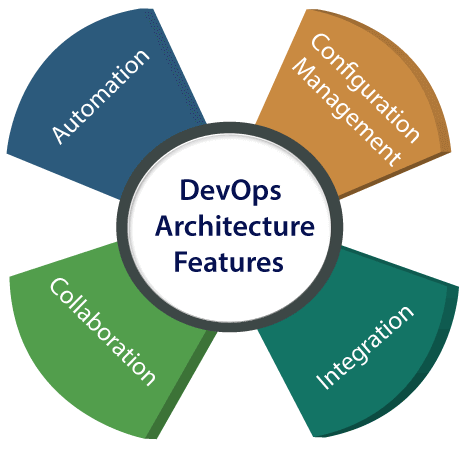
* In 2009, the first conference named **DevOpsdays** was held in Ghent Belgium. Belgian consultant and Patrick Debois founded the conference.
* In 2012, the state of DevOps report was launched and conceived by Alanna Brown at Puppet.
* In 2014, the annual State of DevOps report was published by Nicole Forsgren, Jez Humble, Gene Kim, and others. They found DevOps adoption was accelerating in 2014 also.
* In 2015, Nicole Forsgren, Gene Kim, and Jez Humble founded DORA (DevOps Research and Assignment).
* In 2017, Nicole Forsgren, Gene Kim, and Jez Humble published "Accelerate: Building and Scaling High Performing Technology Organizations".

AD

AD

DevOps Architecture Features

Here are some key features of DevOps architecture, such as:



1) Automation

Automation can reduce time consumption, especially during the testing and deployment phase. The productivity increases, and releases are made quicker by automation. This will lead in catching bugs quickly so that it can be fixed easily. For contiguous delivery, each code is defined through automated tests, cloud-based services, and builds. This promotes production using automated deploys.

2) Collaboration

The Development and Operations team collaborates as a DevOps team, which improves the cultural model as the teams become more productive with their productivity, which strengthens accountability and ownership. The teams share their responsibilities and work closely in sync, which in turn makes the deployment to production faster.

3) Integration

Applications need to be integrated with other components in the environment. The integration phase is where the existing code is combined with new functionality and then tested. Continuous integration and testing enable continuous development. The frequency in the releases and micro-services leads to significant operational challenges. To overcome such problems, continuous integration and delivery are implemented to deliver in a **quicker, safer**, and **reliable manner**.

4) Configuration management

It ensures the application to interact with only those resources that are concerned with the environment in which it runs. The configuration files are not created where the external configuration to the application is separated from the source code. The configuration file can be written during deployment, or they can be loaded at the run time, depending on the environment in which it is running.

DevOps Advantages and Disadvantages

Here are some advantages and disadvantages that DevOps can have for business, such as:

Advantages

* DevOps is an excellent approach for quick development and deployment of applications.
* It responds faster to the market changes to improve business growth.
* DevOps escalate business profit by decreasing software delivery time and transportation costs.
* DevOps clears the descriptive process, which gives clarity on product development and delivery.
* It improves customer experience and satisfaction.
* DevOps simplifies collaboration and places all tools in the cloud for customers to access.
* DevOps means collective responsibility, which leads to better team engagement and productivity.

Disadvantages

* DevOps professional or expert's developers are less available.
* Developing with DevOps is so expensive.
* Adopting new DevOps technology into the industries is hard to manage in short time.
* Lack of DevOps knowledge can be a problem in the continuous integration of automation projects.

Prerequisite

To learn DevOps, you should have basic knowledge of **Linux**, and at least one **Scripting language**.

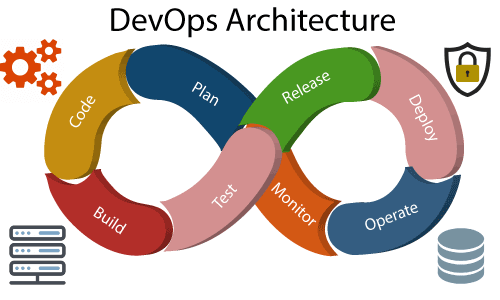
Audience

Our DevOps tutorial is designed to help beginners and professionals.

Problem

We assure you that you will not find any issue with this DevOps tutorial. But if there is any mistake or error, please post the error in the contact form.

# DevOps Architecture

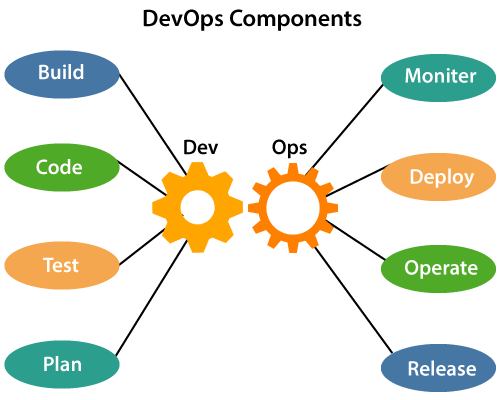


Development and operations both play essential roles in order to deliver applications. The deployment comprises analyzing the **requirements, designing, developing**, and **testing** of the software components or frameworks.

The operation consists of the administrative processes, services, and support for the software. When both the development and operations are combined with collaborating, then the DevOps architecture is the solution to fix the gap between deployment and operation terms; therefore, delivery can be faster.

DevOps architecture is used for the applications hosted on the cloud platform and large distributed applications. Agile Development is used in the DevOps architecture so that integration and delivery can be contiguous. When the development and operations team works separately from each other, then it is time-consuming to **design, test**, and **deploy**. And if the terms are not in sync with each other, then it may cause a delay in the delivery. So DevOps enables the teams to change their shortcomings and increases productivity.

Below are the various components that are used in the DevOps architecture:



### 1) Build

Without DevOps, the cost of the consumption of the resources was evaluated based on the pre-defined individual usage with fixed hardware allocation. And with DevOps, the usage of cloud, sharing of resources comes into the picture, and the build is dependent upon the user's need, which is a mechanism to control the usage of resources or capacity.

### 2) Code

Many good practices such as Git enables the code to be used, which ensures writing the code for business, helps to track changes, getting notified about the reason behind the difference in the actual and the expected output, and if necessary reverting to the original code developed. The code can be appropriately arranged in **files, folders**, etc. And they can be reused.

### 3) Test

The application will be ready for production after testing. In the case of manual testing, it consumes more time in testing and moving the code to the output. The testing can be automated, which decreases the time for testing so that the time to deploy the code to production can be reduced as automating the running of the scripts will remove many manual steps.

### 4) Plan

DevOps use Agile methodology to plan the development. With the operations and development team in sync, it helps in organizing the work to plan accordingly to increase productivity.

### 5) Monitor

Continuous monitoring is used to identify any risk of failure. Also, it helps in tracking the system accurately so that the health of the application can be checked. The monitoring becomes more comfortable with services where the log data may get monitored through many third-party tools such as **Splunk**.

### 6) Deploy

Many systems can support the scheduler for automated deployment. The cloud management platform enables users to capture accurate insights and view the optimization scenario, analytics on trends by the deployment of dashboards.

### 7) Operate

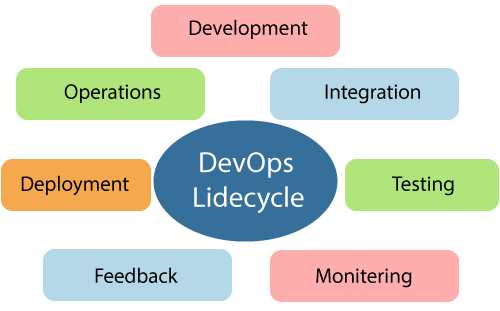
DevOps changes the way traditional approach of developing and testing separately. The teams operate in a collaborative way where both the teams actively participate throughout the service lifecycle. The operation team interacts with developers, and they come up with a monitoring plan which serves the IT and business requirements.

### 8) Release

Deployment to an environment can be done by automation. But when the deployment is made to the production environment, it is done by manual triggering. Many processes involved in release management commonly used to do the deployment in the production environment manually to lessen the impact on the customers.

# DevOps Lifecycle

DevOps defines an agile relationship between operations and Development. It is a process that is practiced by the development team and operational engineers together from beginning to the final stage of the product.



Learning DevOps is not complete without understanding the DevOps lifecycle phases. The DevOps lifecycle includes seven phases as given below:

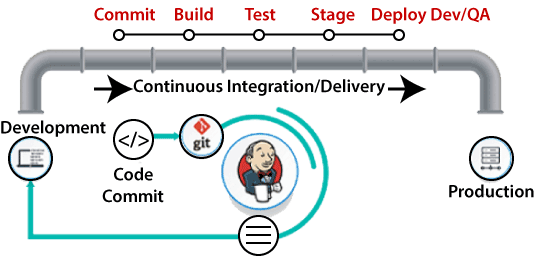
### 1) Continuous Development

This phase involves the planning and coding of the software. The vision of the project is decided during the planning phase. And the developers begin developing the code for the application. There are no DevOps tools that are required for planning, but there are several tools for maintaining the code.

### 2) Continuous Integration

This stage is the heart of the entire DevOps lifecycle. It is a software development practice in which the developers require to commit changes to the source code more frequently. This may be on a daily or weekly basis. Then every commit is built, and this allows early detection of problems if they are present. Building code is not only involved compilation, but it also includes **unit testing, integration testing, code review**, and **packaging**.

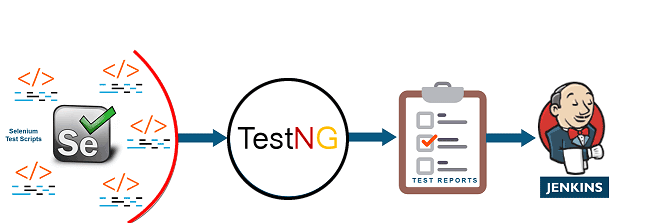
The code supporting new functionality is continuously integrated with the existing code. Therefore, there is continuous development of software. The updated code needs to be integrated continuously and smoothly with the systems to reflect changes to the end-users.



Jenkins is a popular tool used in this phase. Whenever there is a change in the Git repository, then Jenkins fetches the updated code and prepares a build of that code, which is an executable file in the form of war or jar. Then this build is forwarded to the test server or the production server.

### 3) Continuous Testing

This phase, where the developed software is continuously testing for bugs. For constant testing, automation testing tools such as **TestNG, JUnit, Selenium**, etc are used. These tools allow QAs to test multiple code-bases thoroughly in parallel to ensure that there is no flaw in the functionality. In this phase, **Docker** Containers can be used for simulating the test environment.



**Selenium** does the automation testing, and TestNG generates the reports. This entire testing phase can automate with the help of a Continuous Integration tool called **Jenkins**.

Automation testing saves a lot of time and effort for executing the tests instead of doing this manually. Apart from that, report generation is a big plus. The task of evaluating the test cases that failed in a test suite gets simpler. Also, we can schedule the execution of the test cases at predefined times. After testing, the code is continuously integrated with the existing code.

### 4) Continuous Monitoring

Monitoring is a phase that involves all the operational factors of the entire DevOps process, where important information about the use of the software is recorded and carefully processed to find out trends and identify problem areas. Usually, the monitoring is integrated within the operational capabilities of the software application.

It may occur in the form of documentation files or maybe produce large-scale data about the application parameters when it is in a continuous use position. The system errors such as server not reachable, low memory, etc are resolved in this phase. It maintains the security and availability of the service.

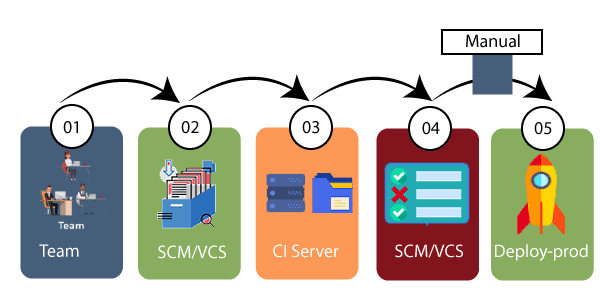
### 5) Continuous Feedback

The application development is consistently improved by analyzing the results from the operations of the software. This is carried out by placing the critical phase of constant feedback between the operations and the development of the next version of the current software application.

The continuity is the essential factor in the DevOps as it removes the unnecessary steps which are required to take a software application from development, using it to find out its issues and then producing a better version. It kills the efficiency that may be possible with the app and reduce the number of interested customers.

### 6) Continuous Deployment

In this phase, the code is deployed to the production servers. Also, it is essential to ensure that the code is correctly used on all the servers.



The new code is deployed continuously, and configuration management tools play an essential role in executing tasks frequently and quickly. Here are some popular tools which are used in this phase, such as **Chef, Puppet, Ansible**, and **SaltStack**.

Containerization tools are also playing an essential role in the deployment phase. **Vagrant** and **Docker** are popular tools that are used for this purpose. These tools help to produce consistency across development, staging, testing, and production environment. They also help in scaling up and scaling down instances softly.

Containerization tools help to maintain consistency across the environments where the application is tested, developed, and deployed. There is no chance of errors or failure in the production environment as they package and replicate the same dependencies and packages used in the testing, development, and staging environment. It makes the application easy to run on different computers.

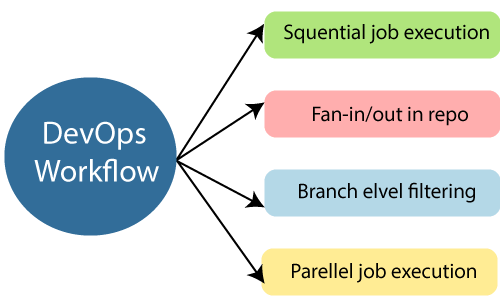
### 7) Continuous Operations

All DevOps operations are based on the continuity with complete automation of the release process and allow the organization to accelerate the overall time to market continuingly.

It is clear from the discussion that continuity is the critical factor in the DevOps in removing steps that often distract the development, take it longer to detect issues and produce a better version of the product after several months. With DevOps, we can make any software product more efficient and increase the overall count of interested customers in your product.

DevOps Workflow

DevOps workflow provides a visual overview of the sequence in which input is provided. Also, it tells about which one action is performed, and output is generated for an operations process.



DevOps workflow allows the ability to separate and arrange the jobs which are top requested by the users. Also, it gives the ability to mirror their ideal process in the configuration jobs.

DevOps Principles

The main principles of DevOps are Continuous delivery, automation, and fast reaction to the feedback.

1. **End to End Responsibility:** DevOps team need to provide performance support until they become the end of life. It enhances the responsibility and the quality of the products engineered.
2. **Continuous Improvement:** DevOps culture focuses on continuous improvement to minimize waste. It continuously speeds up the growth of products or services offered.
3. **Automate Everything:** Automation is an essential principle of the DevOps process. This is for software development and also for the entire infrastructure landscape.
4. **Custom Centric Action:** DevOps team must take customer-centric for that they should continuously invest in products and services.
5. **Monitor and test everything:** The DevOps team needs to have robust monitoring and testing procedures.
6. **Work as one team:** In the DevOps culture role of the designers, developers, and testers are already defined. All they needed to do is work as one team with complete collaboration.

These principles are achieved through several DevOps practices, which include frequent deployments, QA automation, continuous delivery, validating ideas as early as possible, and in-team collaboration.

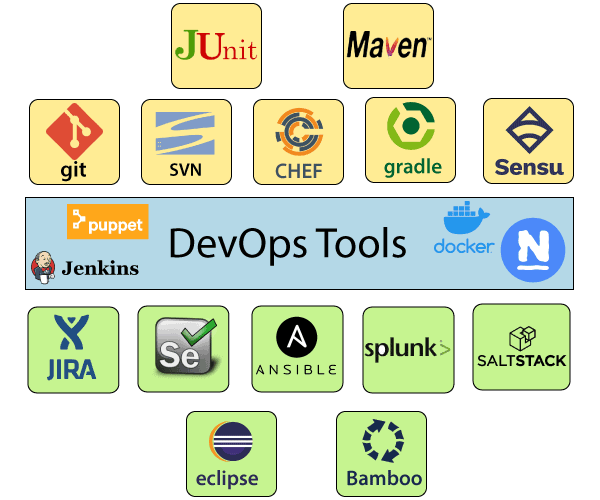
DevOps Practices

Some identified DevOps practices are:

* Self-service configuration
* Continuous build
* Continuous integration
* Continuous delivery
* Incremental testing
* Automated provisioning
* Automated release management

DevOps Tools

Here are some most popular DevOps tools with brief explanation shown in the below image, such as:



1) Puppet

Puppet is the most widely used DevOps tool. It allows the delivery and release of the technology changes quickly and frequently. It has features of versioning, automated testing, and continuous delivery. It enables to manage entire infrastructure as code without expanding the size of the team.

**Features**

* Real-time context-aware reporting.
* Model and manage the entire environment.
* Defined and continually enforce infrastructure.
* Desired state conflict detection and remediation.
* It inspects and reports on packages running across the infrastructure.
* It eliminates manual work for the software delivery process.
* It helps the developer to deliver great software quickly.

2) Ansible

Ansible is a leading DevOps tool. Ansible is an open-source IT engine that automates application deployment, cloud provisioning, intra service orchestration, and other IT tools. It makes it easier for DevOps teams to scale automation and speed up productivity.

Ansible is easy to deploy because it does not use any **agents** or **custom** **security** infrastructure on the client-side, and by pushing modules to the clients. These modules are executed locally on the client-side, and the output is pushed back to the Ansible server.

**Features**

* It is easy to use to open source deploy applications.
* It helps in avoiding complexity in the software development process.
* It eliminates repetitive tasks.
* It manages complex deployments and speeds up the development process.

AD

3) Docker

Docker is a high-end DevOps tool that allows building, ship, and run distributed applications on multiple systems. It also helps to assemble the apps quickly from the components, and it is typically suitable for container management.

**Features**

* It configures the system more comfortable and faster.
* It increases productivity.
* It provides containers that are used to run the application in an isolated environment.
* It routes the incoming request for published ports on available nodes to an active container. This feature enables the connection even if there is no task running on the node.
* It allows saving secrets into the swarm itself.

4) Nagios

Nagios is one of the more useful tools for DevOps. It can determine the errors and rectify them with the help of network, infrastructure, server, and log monitoring systems.

**Features**

* It provides complete monitoring of desktop and server operating systems.
* The network analyzer helps to identify bottlenecks and optimize bandwidth utilization.
* It helps to monitor components such as services, application, OS, and network protocol.
* It also provides to complete monitoring of Java Management Extensions.

5) CHEF

A chef is a useful tool for achieving scale, speed, and consistency. The chef is a cloud-based system and open source technology. This technology uses Ruby encoding to develop essential building blocks such as recipes and cookbooks. The chef is used in infrastructure automation and helps in reducing manual and repetitive tasks for infrastructure management.

Chef has got its convention for different building blocks, which are required to manage and automate infrastructure.

**Features**

* It maintains high availability.
* It can manage multiple cloud environments.
* It uses popular Ruby language to create a domain-specific language.
* The chef does not make any assumptions about the current status of the node. It uses its mechanism to get the current state of the machine.

6) Jenkins

Jenkins is a DevOps tool for monitoring the execution of repeated tasks. Jenkins is a software that allows continuous integration. Jenkins will be installed on a server where the central build will take place. It helps to integrate project changes more efficiently by finding the issues quickly.

**Features**

* Jenkins increases the scale of automation.
* It can easily set up and configure via a web interface.
* It can distribute the tasks across multiple machines, thereby increasing concurrency.
* It supports continuous integration and continuous delivery.
* It offers 400 plugins to support the building and testing any project virtually.
* It requires little maintenance and has a built-in GUI tool for easy updates.

7) Git

Git is an open-source distributed version control system that is freely available for everyone. It is designed to handle minor to major projects with speed and efficiency. It is developed to co-ordinate the work among programmers. The version control allows you to track and work together with your team members at the same workspace. It is used as a critical distributed version-control for the DevOps tool.

**Features**

* It is a free open source tool.
* It allows distributed development.
* It supports the pull request.
* It enables a faster release cycle.
* Git is very scalable.
* It is very secure and completes the tasks very fast.

8) SALTSTACK

Stackify is a lightweight DevOps tool. It shows real-time error queries, logs, and more directly into the workstation. SALTSTACK is an ideal solution for intelligent orchestration for the software-defined data center.

**Features**

* It eliminates messy configuration or data changes.
* It can trace detail of all the types of the web request.
* It allows us to find and fix the bugs before production.
* It provides secure access and configures image caches.
* It secures multi-tenancy with granular role-based access control.
* Flexible image management with a private registry to store and manage images.

9) Splunk

Splunk is a tool to make machine data usable, accessible, and valuable to everyone. It delivers operational intelligence to DevOps teams. It helps companies to be more secure, productive, and competitive.

**Features**

* It has the next-generation monitoring and analytics solution.
* It delivers a single, unified view of different IT services.
* Extend the Splunk platform with purpose-built solutions for security.
* Data drive analytics with actionable insight.

10) Selenium

Selenium is a portable software testing framework for web applications. It provides an easy interface for developing automated tests.

AD

**Features**

* It is a free open source tool.
* It supports multiplatform for testing, such as Android and ios.
* It is easy to build a keyword-driven framework for a WebDriver.
* It creates robust browser-based regression automation suites and tests.

DevOps Automation

Automation is the crucial need for DevOps practices, and automate everything is the fundamental principle of DevOps. Automation kick starts from the code generation on the developers machine, until the code is pushed to the code and after that to monitor the application and system in the production.

Automating infrastructure set up and configurations, and software deployment is the key highlight of DevOps practice. DevOps practice id is dependent on automation to make deliveries over a few hours and make frequent deliveries across platforms.

Automation in DevOps boosts speed, consistency, higher accuracy, reliability, and increases the number of deliveries. Automation in DevOps encapsulates everything right from the building, deploying, and monitoring.

DevOps Automation Tools

In large DevOps team that maintain extensive massive IT infrastructure can be classified into six categories, such as:

* Infrastructure Automation
* Configuration Management
* Deployment Automation
* Performance Management
* Log management
* Monitoring

Below are few tools in each of these categories let see in brief, such as:

Infrastructure Automation

**Amazon Web Services (AWS):** Being a cloud service, you don't need to be physically present in the data center, they are easy to scale on-demand, and there are no up-front hardware costs. It can be configured to provide more servers based on traffic automatically.

Configuration Management

**Chef:** Chef is a handy DevOps tool for achieving speed, scale, and consistency. It can be used to ease out of complex tasks and perform configuration management. With the help of this tool, the DevOps team can avoid making changes across ten thousand servers. Rather, they need to make changes in one place, which is automatically reflected in other servers.

Deployment Automation

**Jenkins:** It facilitates continuous integration and testing. It helps to integrate project changes more efficiently by quickly finding issues as soon as built is deployed.

Performance Management

**App Dynamic:** It offers real-time performance monitoring. The data collected by this tool help developers to debug when issues occur.

Log Management

**Splunk:** This DevOps tool solves issues such as storing, aggregating, and analyzing all logs in one place.

Monitoring

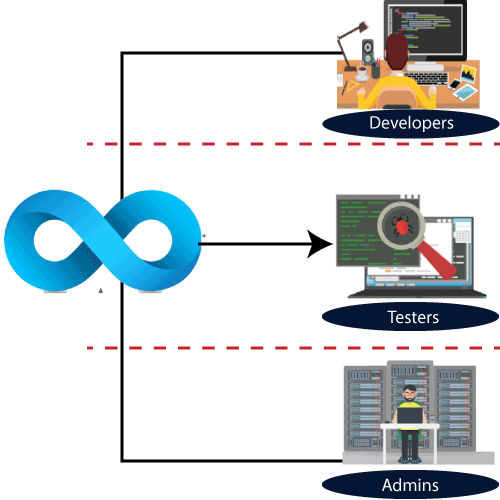
**Nagios:** It notified people when infrastructure and related service go down. Nagios is a tool for this purpose, which helps the DevOps team to find and correct problems.

DevOps Engineers

DevOps Engineer is an IT professional who works with system operators, software developers, and other production IT staff to administer code releases.

DevOps engineer understands the software development lifecycle and various automation tools for developing digital pipelines.

DevOps have hard as well as soft skills to communicate and collaborate with development, testing, and operations teams.



DevOps engineers need to code occasionally from scratch, and they must have the basics of software development languages.

The DevOps engineer will work with development team staff to tackle the coding and scripting needed to connect elements of code, like libraries or software development kits.

A bachelor's degree in computer science or related fields is generally required for DevOps engineers. Many companies prefer those who have a master's degree and at least three to five years of work experience in this field. HTTP, HTML, CSS, SSL, XML, Linux, Java, Amazon Web Services (AWS), NoSQL technologies, DNS, and web app development.

DevOps Engineer Roles and Responsibilities

DevOps engineers work full time. They are responsible for the production and continuing maintenance of a software application platform.

Below are some roles, responsibilities, and skills which are expected from DevOps engineers, such as:

* Manage projects effectively through an open standard based platform.
* Increases project visibility through traceability.
* Improve quality and reduce the development cost with collaboration.
* DevOps should have the soft skill of problem solver and a quick learner.
* Analyze, design, and evaluate automation scripts and systems.
* Able to perform system troubleshooting and problem-solving across the platform and application domains.
* Ensuring the critical resolution of system issues by using the best cloud security solution services.

DevOps Pipeline

A pipeline in software engineering team is a set of automated processes which allows DevOps professionals and developer to reliably and efficiently compile, build, and deploy their code to their production compute platforms.

The most common components of a pipeline in DevOps are build automation or continuous integration, test automation, and deployment automation.

A pipeline consists of a set of tools which are classified into the following categories such as:

* Source control
* Build tools
* Containerization
* Configuration management
* Monitoring

Continuous Integration Pipeline

Continuous integration (CI) is a practice in which developers can check their code into a version-controlled repository several times per day. Automated build pipelines are triggered by these checks which allows fast and easy to locate error detection.

Some significant benefits of CI are:

* Small changes are easy to integrate into large codebases.
* More comfortable for other team members to see what you have been working.
* Fewer integration issues allowing rapid code delivery.
* Bugs are identified early, making them easier to fix, resulting in less debugging work.

AD

Continuous Delivery Pipeline

Continuous delivery (CD) is the process that allows operation engineers and developers to deliver bug fixes, features, and configuration change into production reliably, quickly, and sustainably. Continuous delivery offers the benefits of code delivery pipelines, which are carried out that can be performed on demand.

Some significant benefits of the CD are:

* Faster bug fixes and features delivery.
* CD allows the team to work on features and bug fixes in small batches, which means user feedback received much quicker. It reduces the overall time and cost of the project.

DevOps Methodology

We have a demonstrated methodology that takes an approach to cloud adoption. It accounts for all the factors required for successful approval such as people, process, and technology, resulting in a focus on the following critical consideration:

* **The Teams:** Mission or project and cloud management.
* **Connectivity:** Public, on-premise, and hybrid cloud network access.
* **Automation:** Infrastructure as code, scripting the orchestration and deployment of resources.
* **On-boarding Process:** How the project gets started in the cloud.
* **Project Environment:** TEST, DEV, PROD (identical deployment, testing, and production).
* **Shared Services:** Common capabilities provided by the enterprise.
* **Naming Conventions:** Vital aspect to track resource utilization and billing.
* **Defining Standards Role across the Teams:** Permissions to access resources by job function

Azure DevOps

Azure DevOps is also known as Microsoft visual studio team services (VSTS). It is a set of collaborative development tools built for the cloud. VSTS was commonly used as a standalone term, and Azure DevOps is a platform which is made up of a few different products, such as:

* Azure Test Plans
* Azure Boards
* Azure Repos
* Azure Pipeline
* Azure Artifacts

Azure DevOps is everything that needs to turn an idea into a working piece software. You can plan a project with azure tools.

The azure pipeline is the CI component of azure DevOps. The azure pipeline is Microsoft's cloud-native continuous integration server, which allows teams to continuously build, test, and deploy all from the cloud. An azure pipeline can connect to any number of source code repositories such as Azure Repos, GitHub, Tests, to grab code and artifacts for application delivery.

Azure DevOps Server

Azure DevOps Server is a Microsoft product that provides version control, requirements management, reporting, lab management, project management, testing, automated builds, and release management capabilities. It covers the entire application of lifecycle and enables DevOps capabilities.

Azure DevOps can be used as a back-end to the numerous integrated development environments, but it is modified for Microsoft visual studio and eclipse on all platforms.

Azure DevOps Services

Microsoft announced the release of the software as a service offering of visual studio on the Microsoft Azure platform at the time Microsoft called it a visual studio online.

Microsoft offers visual studio, basic, and stakeholder subscriber access levels for the Azure DevOps services. The basic plan is free of cost for up to five users. Users with a visual studio subscription can be added to a project with no additional charge.

AWS DevOps

AWS is the best cloud service provider, and DevOps is the implementation of the software development lifecycle.

Here are some reasons which make AWS DevOps a highly popular combination, such as:

* AWS CloudFormation
* AWS EC2
* AWS CloudWatch
* AWS CodePipeline

Let's see all of these by one in brief such as:

AWS CloudFormation

DevOps team is required to create and release cloud instances and services more frequently in comparison to development teams. Templates of AWS resources such as EC2 instances, ECS containers, and S3 storage buckets let you set up the entire stack without having to bring everything together.

AWS EC2

You can run containers inside EC2 instances. Hence you can leverage the AWS security and management features.

AWS CloudWatch

This monitoring tool tracks every resource that AWS has to offer. It makes it easy to use third-party tools for monitoring such as sumo logic etc.

AWS CodePipeline

Code Pipeline is an essential feature from AWS, which highly simplifies the way you manage your CI/CD toolset. It integrates with tools such as **Jenkins, GitHub**, and CodeDeploy that enable you to visually control the flow of app updates from build to production.

DevOps Training Certification

DevOps training certification helps anyone to make a career as a DevOps engineer. DevOps certifications are available from Red Hat, Amazon web services, DevOps institution, and Microsoft academy.

Let's see all of these certifications one by one in brief such as:

Red Hat Certification

Red Hat offers a different level of certifications for DevOps professional as follows:

* Red Hat certificate of expertise in the Ansible automation.
* Red Hat certificate of expertise in Platform-as-a-service.
* Red Hat certificate of expertise in Container Administrator.
* Red Hat certificate of expertise in Configuration Management.
* Red Hat certificate of expertise in the Containerized Application Development.

Amazon Web Service Certification

This certificate tests you on how to use the most common DevOps patterns to develop, deploy, and maintain applications on AWS. It also evaluates you on the core principle of the DevOps methodology.

Amazon web service certificate has two requisites. First, the certification fee is $300, and the second time duration is 170 minutes.

DevOps Institution

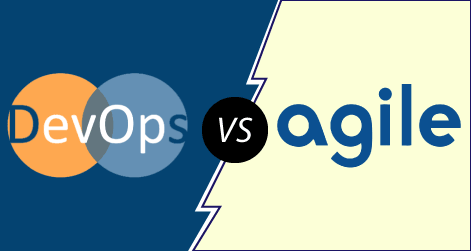
The DevOps institution is a global learning community around emerging DevOps practices. This organization is setting the quality standards for DevOps competency-based qualifications.

Some offered certification courses are:

* DevOps Leader
* DevOps Test Engineer
* DevOps Foundation Certified
* DevOps Foundation
* Certified Agile Process Owner
* Certified Agile Service Manager
* Continuous Delivery Architecture
* DevSecOps Engineer

DevOps vs Agile

DevOps and Agile are the two software development methodologies with similar aims, getting the end-product as quickly and efficiently as possible. While many organizations are hoping to employ these practices, there is often some confusion between both methodologies.



What does each methodology enclose? Where do they overlap? Can they work together, or should we choose one over the other?

Before move further, take a glance at DevOps and Agile.

What is DevOps?

The DevOps is a combination of two words, one is software Development, and second is Operations. This allows a single team to handle the entire application lifecycle, from development to **testing, deployment**, and **operations**. DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators.

DevOps promotes collaboration between Development and Operations team to deploy code to production faster in an automated & repeatable way.

DevOps helps to increase organization speed to deliver applications and services. It also allows organizations to serve their customers better and compete more strongly in the market.

DevOps can also be defined as a sequence of development and IT operations with better communication and collaboration.

DevOps has become one of the most valuable business disciplines for enterprises or organizations. With the help of DevOps, **quality**, and **speed** of the application delivery has improved to a great extent.

DevOps is nothing but a practice or methodology of making "**Developers**" and "**Operations**" folks work together. DevOps represents a change in the IT culture with a complete focus on rapid IT service delivery through the adoption of agile practices in the context of a system-oriented approach.

What is Agile?

The Agile involves continuous iteration of development and testing in the **SDLC** process. Both development and testing activities are concurrent, unlike the waterfall model. This software development method emphasizes on incremental, iterative, and evolutionary development.

It breaks the product into small pieces and integrates them for final testing. It can be implemented in many ways, such as **Kanban, XP, Scrum**, etc.

The Agile software development focus on the four core values, such as:

* Working software over comprehensive documentation.
* Responded to change over following a plan.
* Customer collaboration over contract negotiation.
* Individual and team interaction over the process and tools.

**Below are some essential differences between the DevOps and Agile:**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **DevOps** | **Agile** |
| Definition | DevOps is a practice of bringing development and operation teams together. | Agile refers to the continuous iterative approach, which focuses on collaboration, customer feedback, small, and rapid releases. |
| Purpose | DevOps purpose is to manage end to end engineering processes. | The agile purpose is to manage complex projects. |
| Task | It focuses on constant testing and delivery. | It focuses on constant changes. |
| Team size | It has a large team size as it involves all the stack holders. | It has a small team size. As smaller is the team, the fewer people work on it so that they can move faster. |
| Team skillset | The DevOps divides and spreads the skill set between development and the operation team. | The Agile development emphasizes training all team members to have a wide variety of similar and equal skills. |
| Implementation | DevOps is focused on collaboration, so it does not have any commonly accepted framework. | Agile can implement within a range of tactical frameworks such as **safe, scrum**, and **sprint**. |
| Duration | The ideal goal is to deliver the code to production daily or every few hours. | Agile development is managed in units of sprints. So this time is much less than a month for each sprint. |
| Target areas | End to End business solution and fast delivery. | Software development. |
| Feedback | Feedback comes from the internal team. | In Agile, feedback is coming from the customer. |
| Shift left principle | It supports both variations left and right. | It supports only shift left. |
| Focus | DevOps focuses on operational and business readiness. | Agile focuses on functional and non-functional readiness. |
| Importance | In DevOps, developing, testing, and implementation all are equally important. | Developing software is inherent to Agile. |
| Quality | DevOps contributes to creating better quality with automation and early bug removal. Developers need to follow Coding and best Architectural practices to maintain quality standards. | The Agile produces better applications suites with the desired requirements. It can quickly adapt according to the changes made on time during the project life. |
| Tools | **Puppet, Chef, AWS, Ansible**, and team City OpenStack are popular DevOps tools. | **Bugzilla, Kanboard, JIRA** are some popular Agile tools. |
| Automation | Automation is the primary goal of DevOps. It works on the principle of maximizing efficiency when deploying software. | Agile does not emphasize on the automation. |
| Communication | DevOps communication involves specs and design documents. It is essential for the operational team to fully understand the software release and its network implications for the enough running the deployment process. | Scrum is the most common method of implementing Agile software development. Scrum meeting is carried out daily. |
| Documentation | In the DevOps, the process documentation is foremost because it will send the software to an operational team for deployment. Automation minimizes the impact of insufficient documentation. However, in the development of sophisticated software, it's difficult to transfer all the knowledge required. | The agile method gives priority to the working system over complete documentation. It is ideal when you are flexible and responsive. However, it can harm when you are trying to turn things over to another team for deployment. |

DevOps Interview Questions

A list of frequently asked **DevOps Interview Questions** and Answers are given below.

1) Explain DevOps?

The DevOps is a combination of two words one is software Development, and second is Operations. It allows a single team to handle the entire application lifecycle, from development to **testing, deployment,** and **operations**. DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators.

2) What are the principles of DevOps?

The principles behind DevOps are:

* Continuous deployment
* Infrastructure as code
* Automation
* Monitoring
* Security

3) How DevOps is helpful to developers?

DevOps is very helpful for developers to fix the bugs and quickly implement the new features. It also helps in more transparent communication between the team members.

4) Explain some popular tools of DevOps?

Here are some popular tools of DevOps, such as:

1. **Jenkins:** Jenkins is a DevOps tool for monitoring the execution of repeated tasks. Jenkins is a software that allows continuous integration. And it will be installed on a server where the central build will take place.
2. **Ansible:** Ansible is a leading DevOps tool. Ansible is an open-source IT engine that automates application deployment, cloud provisioning, intra service orchestration, and other IT tools.
3. **Nagios:** Nagios is one of the more useful tools for DevOps. It can determine the errors and rectify them with the help of network, infrastructure, server, and log monitoring systems.
4. **Docker:** Docker is a high-end DevOps tool that allows building, ship, and run distributed applications on multiple systems.
5. **Git:** Git is an **open-source distributed version control system** that is freely available for everyone. It is designed to handle minor to major projects with speed and efficiency.

5) What are the prerequisites for the DevOps implementation?

Following are some useful prerequisites for DevOps implementation:

* Proper communication between the team members.
* At least one version control software.
* Automated testing.
* Automated deployment.

AD

6) What are the core operation of DevOps with application development and infrastructure?

The core operation of DevOps with application development and infrastructure are:

**Application development**

* Code building
* Code coverage
* Unit testing
* Packaging
* Deployment

**Infrastructure**

* Provisioning
* Configuration
* Orchestration
* Deployment

7) What are the key components of DevOps?

The most important key components of DevOps are:

* Continuous integration
* Continuous testing
* Continuous delivery
* Continuous mongering

8) What is Vagrant?

A Vagrant is a tool of DevOps that can create and manage virtualized environments for testing and developing software.

9) What are the technical and business benefits of DevOps work culture?

Following are the technical and business benefits of DevOps work culture:

**Technical benefits**

* Less complex problems to fix.
* Continuous software delivery.
* Faster bugs resolution.

**Business benefits**

* Most stable operating environments.
* More time available to add product value.
* Faster delivery of features for customer satisfaction.

10) What is Scrum?

Scrum is used to divide a sophisticated software and product development task into smaller chunks using iterations and increasing practices. Scrum consist of three roles, such as:

1. Product owner
2. Scrum master
3. Team

11) What is version control?

It is a system that records the changes to a file or set of files over time so you can recall specific versions.

The version control system consists of a central shared repository where teammates can commit the changes to a file or set of files. It allows:

* To revert the file to the previous state.
* To compare changes over time.
* To revert the entire project to a previous state.
* To see who modified something in the files.

12) Which scripting tools are used in DevOps?

Both Python and Ruby scripting tools are used in the DevOps.

13) What is the role of AWS in DevOps?

AWS is a cloud-based service provided by Amazon that ensures scalability through unlimited computing power and storage. It empowers IT enterprises to develop and deliver experienced products and deploy applications on the cloud.

14) Why open source tools boost DevOps?

Open source tools mainly used by any organization which is adapted by DevOps pipeline because DevOps came with the focus of automation in various aspects of organization build, release, change management, and infrastructure management areas.

15) What is the need for organization playbooks as the role?

Organization playbooks as the role give more reliability and reusability to any plays while considering a task where MySQL installation should be done after the removal of Oracle DB. You need another requirement to install MySQL after java installation. In both cases, you need to install MySQL, but without roles, it needs to write playbooks separately for both use cases. Still, once using roles, the MySQL installation role is created can be utilized any number of times by invoking using logic in **site.yaml**.

16) How many cloud platform which is used for DevOps implementation?

Here are some cloud computing platform used for DevOps implementation, such as:

AD

* Google Cloud
* Amazon Web Services
* Microsoft Azure

17) Explain two-factor authentication?

The two-factor authentication is a security method in which the user provides two ways of identification from separate categories.

18) What is the use of PTR in DNS?

PTR is a pointer record that is used for a reverse DNS lookup.

19) Name some network monitoring tools?

Some most essential network monitoring tools are:

* Nagios
* OpenNMS
* Splunk
* Icinga 2
* Wireshark

20) Name three important DevOps KPIs?

Three most important DevOps KPIs are:

* Deployment frequency
* Percentage of failed deployments
* Mean time to failure recovery

21) Explain pair programming regarding DevOps?

Pair programming is an engineering practice of Extreme Programming Rules. In this case, two programmers work on the same system on the same design or algorithm.

One programmer act as a **Driver** and other as an **Observer** who continuously monitors the progress of a project to identify problems. The roles can be reversed at any point of time without any prior intimation.

22) What is CBD?

Component-Based Development (CBD) is a unique way to approach product development. In this method, developers always look for existing well defined, tested, and verified components to compose and assemble them to a product instead of developing from scratch.

23) Define DevOps Automation?

Automation is the crucial need for DevOps practices, and automate everything is the fundamental principle of DevOps. Automation kick starts from the code generation on the developer's machine, until the code is pushed to the code, to monitor the application and system in the production.

Automation in DevOps boosts speed, consistency, higher accuracy, reliability, and increases the number of deliveries.

24) What is the DevOps pipeline?

A pipeline in software engineering team is a set of automated processes which allows DevOps professionals and developer to reliably and efficiently compile, build, and deploy their code to their production compute platforms.

25) What is Azure DevOps?

Azure DevOps is also known as Microsoft visual studio team services (VSTS). It is a set of collaborative development tools built for the cloud. VSTS was commonly used as a standalone term, and Azure DevOps is a platform which is made up of a few different products, such as:

* Azure Test Plans
* Azure Boards
* Azure Repos
* Azure Pipeline
* Azure Artifacts

26) Which makes AWS DevOps highly accessible?

Here are some reasons which make AWS DevOps a highly popular, such as:

* AWS CloudFormation
* AWS EC2
* AWS CloudWatch
* AWS CodePipeline

27) What is the Build in DevOps?

The build is a method in which the source code is put together to check whether it works as a single unit. In the build creation process, the source code will undergo compilation, testing, inspection, and deployment.

28) Explain at what instance has used the SSH?

You can use SSH to log into a remote machine and work on the command line. Also, you can use it to tunnel into the system to facilitate secure encrypted communication between two hosts over an insecure network.

29) How infrastructure code is executed in AWS?

In AWS:

* The code for infrastructure will be in a simple JSON format.
* This JSON code will be organized into files called templates.
* These templates can be deployed on AWS DevOps and then managed as stacks.
* Later the CloudFormation service will do the Creating, deleting, updating, etc. operation in the stack.

30) Explain Git Bisect?

Git Bisect helps to find the commit which introduced a bug using binary search.

Ansible Tutorial



Ansible tutorial provides basic and advanced concepts of Ansible. Our Ansible tutorial is designed for beginners and professionals.

Ansible is an open-source IT engine which automates the IT tools such as intra service orchestration, application deployment, cloud provisioning, etc.

What is Ansible?

Ansible is an open-source IT engine that automates application deployment, cloud provisioning, intra service orchestration, and other IT tools.

Ansible is easy to deploy because it does not use any **agents** or **custom security** infrastructure on the client-side, and by pushing modules to the clients. These modules are executed locally on the client-side, and the output is pushed back to the Ansible server.

It can easily connect to clients using **SSH-Keys**, simplifying though the whole process. Client details, such as **hostnames** or **IP addresses** and **SSH ports**, are stored in the files, which are called inventory files. If you created an inventory file and populated it, then Ansible can use it.

Ansible uses the playbook to describe automation jobs, and playbook, which uses simple language, i.e., **YAML**. YAML is a human-readable data serialization language & commonly used for configuration files, but it can be used in many applications where data is being stored.

A significant advantage is that even the IT infrastructure support guys can read and understand the playbook and debug if needed.

Ansible is designed for multi-tier deployment. Ansible does not manage one system at a time, and it models IT infrastructure by describing all of your systems are interrelated. Ansible is entirely agentless, which means Ansible works by connecting your nodes through **SSH** (by default). Ansible gives the option to you if you want another method for the connection like **Kerberos**.

Ansible pushes small programs after connecting to your nodes which are known as "**Ansible Modules**". Ansible runs that module on your nodes and removes them when finished. Ansible manages the inventory in simple text files (These are the host's files). Ansible uses the host file where one can group the hosts and can control the actions on a specific group in the playbooks.

Why Use Ansible

Here are some important reasons for using Ansible, such as:

* Ansible is free to use by everyone.
* Ansible is very consistent and lightweight, and no constraints regarding the operating system or underlying hardware are present.
* It is very secure due to its agentless capabilities and open **SSH** security features.
* Ansible does not need any special system administrator skills to install and use it.
* Ansible has a smooth learning curve determined by the comprehensive documentation and easy to learn structure and configuration.
* Its modularity regarding **plugins, inventories, modules,** and **playbooks** make Ansible perfect companion orchestrate large environments.

Ansible History

Here are some essential points from the history of Ansible, such as:

* **Michael DeHaan** developed Ansible, and the Ansible project began in **February 2012**.
* The creator of **Cobbler** and **Func** is also the controller of the **Fedora Unified** network.
* **RedHat** acquired the Ansible tool in 2015.
* Ansible is included as part of the **Fedora distribution** of the Linux.
* Ansible is also available for **RedHat Enterprise Linux, Debian, CentOS, Oracle Linux,** and **Scientific Linux** via Extra Packages for Enterprise Linux **(EPEL)** and **Ubuntu** as well as for other operating systems.

AD

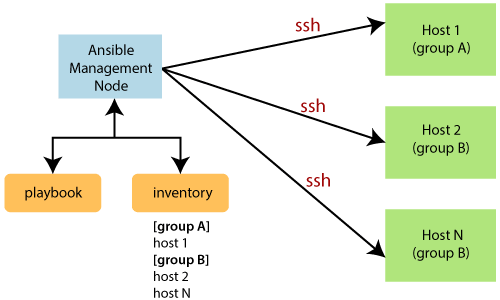
AD

Prerequisite

To learn Ansible, you have hands-on experience with running commands into a Linux shell. This will help you the Ansible tasks in a better way.

# Ansible Workflow

Ansible works by connecting to your nodes and pushing out a small program called **Ansible modules** to them. Then Ansible executed these modules and removed them after finished. The library of modules can reside on any machine, and there are no daemons, **servers,** or **databases** required.



In the above image, the **Management Node** is the controlling node that controls the entire execution of the playbook. The **inventory** file provides the list of hosts where the Ansible modules need to be run. The **Management Node** makes an **SSH** connection and executes the small modules on the host's machine and install the software.

Ansible removes the modules once those are installed so expertly. It connects to the host machine executes the instructions, and if it is successfully installed, then remove that code in which one was copied on the host machine.

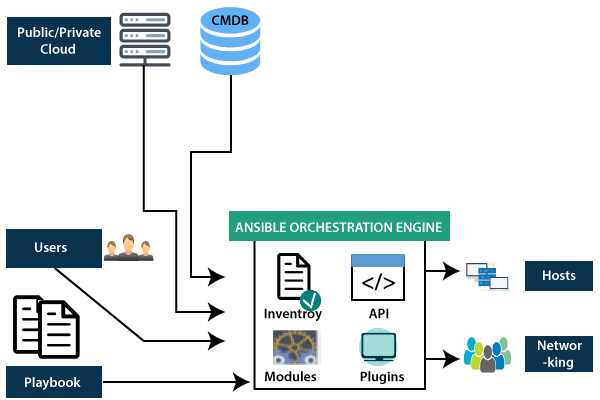
## Terms used in Ansible

Here are some important terms which are used in Ansible, such as:

|  |  |
| --- | --- |
| **Terms** | **Explanation** |
| Ansible Server | It is a machine where Ansible is installed and from which all tasks and playbooks will be executed. |
| Modules | The module is a command or set of similar commands which is executed on the client-side. |
| Task | A task is a section which consists of a single procedure to be completed. |
| Role | It is a way of organizing tasks and related files to be later called in a playbook. |
| Fact | The information fetched from the client system from the global variables with the gather facts operation. |
| Inventory | A file containing the data regarding the Ansible client-server. |
| Play | It is the execution of the playbook. |
| Handler | The task is called only if a notifier is present. |
| Notifier | The section attributed to a task which calls a handler if the output is changed. |
| Tag | It is a name set to a task that can be used later on to issue just that specific task or group of jobs. |

# Ansible Architecture

The Ansible orchestration engine interacts with a user who is writing the Ansible playbook to execute the Ansible orchestration and interact along with the services of private or public cloud and configuration management database. You can show in the below diagram, such as:



### Inventory

Inventory is lists of nodes or hosts having their IP addresses, databases, servers, etc. which are need to be managed.

### API's

The Ansible API's works as the transport for the public or private cloud services.

### Modules

Ansible connected the nodes and spread out the Ansible modules programs. Ansible executes the modules and removed after finished. These modules can reside on any machine; no database or servers are required here. You can work with the chose text editor or a terminal or version control system to keep track of the changes in the content.

### Plugins

Plugins is a piece of code that expends the core functionality of Ansible. There are many useful plugins, and you also can write your own.

### Playbooks

Playbooks consist of your written code, and they are written in YAML format, which describes the tasks and executes through the Ansible. Also, you can launch the tasks synchronously and asynchronously with playbooks.

### Hosts

In the Ansible architecture, hosts are the node systems, which are automated by Ansible, and any machine such as RedHat, Linux, Windows, etc.

### Networking

Ansible is used to automate different networks, and it uses the simple, secure, and powerful agentless automation framework for IT operations and development. It uses a type of data model which separated from the Ansible automation engine that spans the different hardware quite easily.

### Cloud

A cloud is a network of remote servers on which you can store, manage, and process the data. These servers are hosted on the internet and storing the data remotely rather than the local server. It just launches the resources and instances on the cloud, connect them to the servers, and you have good knowledge of operating your tasks remotely.

### CMDB

CMDB is a type of repository which acts as a data warehouse for the IT installations.

Ansible Installation in Linux

When you have compared and weighed your options and decided to go for Ansible. Then installed it on your system. Let's go step by step of the installation in different Linux distributions, such as:

Prerequisites

1. PyYAML: a YAML parser and emitter for the python programming language.
2. Httplib2: a comprehensive HTTP client library.
3. parmiko: native python SSHv2 protocol library.
4. Distro: RHEL/ CentOS/ Debian/ Ubuntu Linux.
5. Jinja2: a modern and designer friendly templating language for python.
6. sshpass: a non-interactive ssh password authentication.

Install Ansible on RedHat/Centos systems

**Step 1:** Install the EPEL repo

1. [root@ansible-server ~]# sudo yum install epel-release

**Step 2:** Install the Ansible package.

1. [root@ansible-server ~]# sudo yum install -y ansible

Install Ansible on Debian/Ubuntu systems

**Step 1:** First perform an update to the packages

1. $ sudo apt update

**Step 2:** Then install the software properties common package.

1. $ sudo apt install software-properties-common

**Step 3:** And install the Ansible personal package archive.

1. $ sudo apt-add-repository ppa:ansible/ansible

**Step 4:** Install the Ansible.

1. $ sudo apt update
2. $ sudo apt install ansible

AD

Install Ansible using pip

The pip command is a tool for installing and managing python packages.

**Step 1:** This given method works on the Linux and UNIX like systems.

1. $ sudo pip install ansible

Install the latest version of Ansible

**Using Portage (Gentoo)**

1. $ emerge -av app-admin/ansible

To install the latest version, you need to unmask the Ansible package prior to emerging:

1. $ echo 'app-admin/ansible' >> /etc/portage/**package**.accept\_keywords

**Using pkg (FreeBSD)**

Ansible works with both python 2 and 3 version, FreeBSD has different packages for each python version. To install you can use:

1. $ sudo pkg install py27-ansible

**Or**

1. $ sudo pkg install py36-ansible

You can also to install from ports:

1. $ sudo make -C /usr/ports/sysutils/ansible install

**Using OpenCSW (Solaris)**

The Ansible is available for Solaris as SysV package from openCSW:

1. # pkgadd -d http://get.opencsw.org/now
2. # /opt/csw/bin/pkgutil -i ansible

**Using Pacman (Arch Linux)**

Ansible is available in the community repository:

1. $ pacman -S ansible

**Using sbopkg (Slackware Linux)**

Ansible build script is available in the SlackBuilds.org repository. Also can be built and installed using sbopkg.

**Step 1:** Create queue with Ansible and all dependencies.

AD

1. # sqg -p ansible

**Step 2:** Build and install packages from created queue file.

1. # sbopkg -k -i ansible

**Using swupd (Clear Linux)**

**Step 1:** Ansible and its dependencies are available as part of the sysadmin host management bundle.

1. $ sudo swupd bundle-add sysadmin-hostmgmt

**Step 2:** And update of the software will be managed by the swupd tool.

1. $ sudo swupd update

Installing

**By using yum**

On Fedora:

1. $ sudo dnf install python-argcomplete

On RHEL and CentOS:

1. $ sudo yum install epel-release
2. $ sudo yum install python-argcomplete

**By using apt**

1. $ sudo apt install python-argcomplete

**By using pip**

1. $ pip install argcomplete

Ansible ad-hoc Commands

Ad-hoc commands are one of the simplest ways of using Ansible. These are used when you want to issue some commands on a server or bunch of servers. The ad-hoc commands are not stored for future use, but it represents a fast way to interact with the desired servers.

The Ansible ad-hoc command uses the **/usr/bin/ansible** command-line tool to automate a single task on one or more managed nodes. The Ad-hoc commands are quick and easy, but they are not re-usable. The Ad-hoc commands demonstrate the simplicity and power of Ansible.

Syntax

1. ansible <hosts> [-m <module\_name>] -a <"arguments"> -u <username> [--become]

Explanation

**Hosts:** It can be an entry in the inventory file. For specifying all hosts in the inventory, use all or "\*".

**module\_name:** It is an optional parameter. There are hundreds of modules available in the Ansible, such as **shell, yum, apt, file,** and **copy**. By default, it is the **command**.

**Arguments:** We should pass values that are required by the module. It can change according to the module used.

**Username:** It specifies the user account in which Ansible can execute commands.

**Become:** It's an optional parameter specified when we want to run operations that need sudo privilege. By default, it becomes false.

1. Parallelism and shell commands

You can reboot your company server in 12 parallel forks at the same time. For this, you need to set up the SSHagent for connection.

1. $ ssh-agent bash
2. $ ssh-add ~/.ssh/id\_rsa

To run reboot for all your company servers in the group, 'abc', in 12 parallel forks:

1. $ ansible abc -a "/sbin/reboot" -f 12

By default, Ansible will run the above ad-hoc commands from the current user account. If you want to change then pass the username in ad-hoc command as follows:

1. $ ansible abc -a "/sbin/reboot" -f 12 -u username

2. File Transfer

You can use ad-hoc commands for doing SCP (secure copy protocol) which means lots of files in parallel on multiple machines or servers.

**Transferring file on many machines or servers**

1. $ ansible abc -m copy -a "src = /etc/yum.conf dest = /tmp/yum.conf"

**Creating new directory**

1. $ ansible abc -m file -a "dest = /path/user1/new mode = 888 owner = user1 group = user1 state = directory"

**Deleting all directory and files**

1. $ ansible abc -m file -a "dest = /path/user1/new state = absent"

3. Managing Packages

Ad-hoc commands are available for apt and yum module. Here are the following ad-hoc commands using yum.

Below command checks, if the yum package is installed or not, but not update it.

1. $ ansible abc -m yum -a "name = demo-tomcat-1 state = present"

Below command checks the package is not installed.

1. $ ansible abc -m yum -a "name = demo-tomcat-1 state = absent"

And below command checks the latest version of package is installed.

1. $ ansible abc -m yum -a "name = demo-tomcat-1 state = latest"

4. anaging Users and Groups

You can manage, create, and remove a user account on your managed nodes with ad-hoc commands.

1. $ ansible all -m user -a "name=foo password=<crypted password here>"
3. $ ansible all -m user -a "name=foo state=absent"

5. Managing Services

Ensure a service is started on all the webservers.

1. $ ansible webservers -m service -a "name=httpd state=started"

Alternatively, restart a service on all webservers:

1. $ ansible webservers -m service -a "name=httpd state=restarted"

Ensure a service is stopped:

AD

1. $ ansible webservers -m service -a "name=httpd state=stopped"

6. Gathering Facts

Fact represents the discovered variables about a system. You can use the facts to implement conditional execution of tasks, and also used to get ad-hoc information about your systems. To see all the facts:

1. $ ansible all -m setup

Ansible Playbooks

Playbooks are the files where the Ansible code is written. Playbooks are written in YAML format. **YAML** means "Yet Another Markup Language," so there is not much syntax needed. **Playbooks** are one of the core features of Ansible and tell Ansible what to execute, and it is used in complex scenarios. They offer increased flexibility.

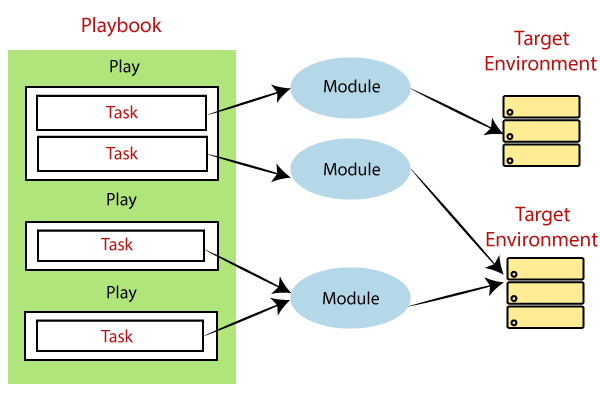
Playbooks contain the steps which the user wants to execute on a particular machine. And playbooks are run sequentially. Playbooks are the building blocks for all the use cases of Ansible.

Ansible playbooks tend to be more configuration language than a programming language.

Through a playbook, you can designate specific roles to some of the hosts and other roles to other hosts. By doing this, you can orchestrate multiple servers in very different scenarios, all in one playbook.

Playbook Structure

Each playbook is a collection of one or more plays. Playbooks are structured by using Plays. There can be more than one play inside a playbook.



The function of the play is to map a set of instructions which is defined against a particular host.

There are different YAML editors, but prefer to use a simple editor such as notepad++. First, open the notepad++ and copy-paste the below YAML and change the language to YAML (Language → YAML).

A YAML starts with --- (3 hyphens) always.

Create a Playbook

Let's start by writing an example YAML file. First, we must define a task. These are the interface to ansible modules for roles and playbooks.

One playbook with one play, containing multiple tasks looks like the below example.

---

   name: install and configure DB

   hosts: testServer

   become: yes

   vars:

      oracle\_db\_port\_value : 1521

   tasks:

   -name: Install the Oracle DB

      yum: <code to install the DB>

   -name: Ensure the installed service is enabled and running

   service:

      name: <your service name>

Above is a basic syntax of a playbook. Save it in a file as **test.yml**. A YAML syntax needs to follow the correct indentation.

YAML Tags

Here are some YAML tags are given below, such as:

|  |  |
| --- | --- |
| **Tags** | **Explanation** |
| Name | It specifies the name of the Ansible Playbooks. |
| Hosts | It specifies the lists of the hosts against which you want to run the task. And the host's Tag is mandatory. It tells Ansible that on which hosts to run the listed tasks. These tasks can be run on the same machine or the remote machine. One can run the tasks on the multiple machines, and the host's tag can have a group of host's entry as well. |
| Vars | Vars tag defines the variables which you can use in your playbook. Its usage is similar to the variables in any programming language. |
| Tasks | Tasks are the lists of the actions which need to perform in the playbooks. All the playbooks should contain the tasks to be executed. A task field includes the name of the task. It is not mandatory but useful for debugging the playbook. Internally each task links to a piece of code called a module. A module should be executed, and arguments that are required for the module you want to run. |

Ansible Tower

Ansible Tower is like Ansible at a more enterprise level. It is a web-based solution for managing your organization with an easy user interface that provides a dashboard with all of the state summaries of all the hosts. And allows quick deployments, and monitors all configurations.

The tower allows us to share the **SSH** credentials without exposing them, logs all the jobs, manage inventories graphically, and syncs them with a wide variety of cloud providers.

Previously, Ansible Tower called the **AWX** project, is the fix to this problem. Especially those that render better as graphical rather than text-based output, such as real-time node monitoring.

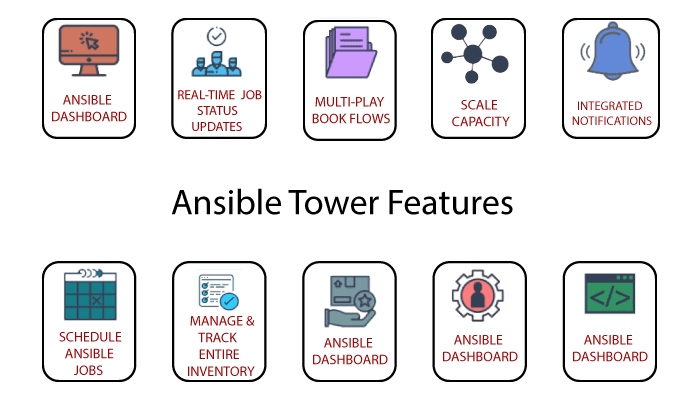
Prerequisites to Install Ansible Tower

There is the following prerequisite to install the Ansible Tower, such as:

* The following operating systems support Ansible Tower
  + RedHat Enterprise Linux 6 64-bit
  + RedHat Enterprise Linux 7 64-bit
  + CentOS 6 64-bit
  + CentOS 7 64-bit
  + Ubuntu 12.04 LTS 64-bit
  + Ubuntu 14.04 LTS 64-bit
  + Ubuntu 16.04 LTS 64 bit
* You should have the latest stable release of Ansible.
* It required a 64-bit support kernel, runtime, and 20 GB hard disk.
* Minimum 2 GB RAM (4 GB RAM recommended) is required.
  + Minimum 2 GB RAM is recommended for Vagrant trial installations
  + And 4 GB RAM is recommended /100 forks

Ansible Tower Features

Here are some features of the Ansible Tower, such as:



**1. Ansible Tower Dashboard:** It displays everything which is going on in your Ansible environment, such as the inventory status, the recent job activity, the hosts, and so on.

**2. Multi-Playbook Workflows:** It allows to chain any numbers of playbooks, any way of the usage of different inventories, runs different users, or utilizes various credentials.

**3. Real-Time Job Updates:** Ansible can automate the complete infrastructure. Also, you can see real-time job updates such as plays and tasks broken down by each machine either been successful or failure. Therefore you can see the status of your automation and know what's next in the queue.

**4. Scale Capacity with Cluster:** You can connect multiple Ansible Tower nodes into an Ansible Tower cluster as the clusters add redundancy and capacity, which allows scaling Ansible automation across the enterprise.

**5. Self-Service:** You can launch playbooks with just a single click through this feature.

**6. Remote Command Execution:** With this command, you can run simple tasks such as restart any malfunctioning service, add users, reset passwords on any host or group of hosts in the inventory.

**7. Manage and Track Inventory:** It manages your entire infrastructure by pulling inventory from public cloud providers such as Microsoft Azure, amazon web services, etc.

**8. Integrated Notification:** This notifies you when a job succeeds or fails across the entire organization at once, or customize on a pre-job basis.

**9. Schedule Ansible Jobs:** It schedule different kinds of jobs such as playbook runs, cloud inventory updates, and source control updates to run according to the need.

**10. REST API and Tower CLI Tool:** Every feature present in Ansible Tower is available through the Ansible Tower's REST API, which provides the ideal API for the systems management infrastructure. The Ansible Tower's CLI tool is available for launching jobs from CI systems such as Jenkins, or when you need to integrate with other command-line tools.

Ansible Roles

Roles provide a framework for fully independent or interdependent collections of files, tasks, templates, variables, and modules.

The role is the primary mechanism for breaking a playbook into multiple files. This simplifies writing **complex playbooks** and makes them easier to reuse. The breaking of the playbook allows you to break the playbook into reusable components.

Each role is limited to a particular functionality or desired output, with all the necessary steps to provide that result either within the same role itself or in other roles listed as dependencies.

Roles are not playbooks. Roles are small functionality that can be used within the playbooks independently. Roles have no specific setting for which hosts the role will apply.

Top-level playbooks are the bridge holding the hosts from your inventory file to roles that should be applied to those hosts.

Creating a Role

The directory structure for roles is essential to creating a new role, such as:

**Role Structure**

The roles have a structured layout on the file system. You can change the default structured of the roles as well.

**For example,** let us stick to the default structure of the roles. Each role is a directory tree in itself. So the role name is the directory name within the /roles directory.

1. $ ansible-galaxy -h

Usage

1. ansible-galaxy [delete|**import**|info|init|install|list|login|remove|search|setup] [--help] [options] ...

Options

* -h: (help) it shows this help message and exit.
* -v: (verbose) Verbose mode (-vvv for more, -vvvv to enable connection debugging).
* --version: it shows program version number and exit.

Roles are stored in separate directories and have a particular directory structure

[root@ansible-server test2]# tree

.

`-- role1

    |-- defaults

    |   `-- main.yml

    |-- handlers

    |   `-- main.yml

    |-- meta

    |   `-- main.yml

    |-- README.md

    |-- tasks

    |   `-- main.yml

    |-- tests

    |   |-- inventory

    |   `-- test.yml

    `-- vars

        `-- main.yml

AD

Explanation

* The YAML file in the default directory contains a list of default variables that are to be used along with the playbook.
* The handler's directory is used to store handlers.
* The meta-directory is supposed to have information about the author and role dependencies.
* The tasks directory is the main YAML file for the role.
* The tests directory contains a sample YAML playbook file and a sample inventory file and is mostly used for testing purposes before creating the actual role.
* The vars directory contains the YAML file in which all the variables used by the role will be defined. The directory templates and the directory files should contain files and templates that will be used by the tasks in the role

Ansible Variables

In playbooks, the variable is very similar to using the variables in a programming language. It helps you to assign a value to a variable and use it anywhere in the playbook. You can put the conditions around the value of the variables and use them in the playbook accordingly.

Creating Valid Variable Names

Before start using variables, it's important to know what valid variable names are.

Variable names should be letters, numbers, and underscores. The variable should always start with a letter.

foo\_port and foo2 both are the correct or valid variable names.

Foo-port, foo port, foo.port, and 10foo all are invalid variable names.

YAML supports dictionaries that map keys to values. For instance:

foo:

  field1: one

  field2: two

Then you can reference a specific field in the dictionary using either bracket notation or dot notation:

foo['field1']

foo.field1

Both will reference the same value "one". But, if you choose to use dot notation, be aware that some keys can cause problems because they collide with the attributes and methods of python dictionaries. You should use bracket notation instead of dot notation if you use keys which start and end with two underscores or any of the known public attributes:

Example

- hosts : <your hosts>

vars:

tomcat\_port : 8080

In the above example, defined a variable name **tomcat\_port** and assigned the value 8080 to the variable and can use it in your playbook wherever required.

The below code is from one of the roles (install-tomcat), such as:

block:

   - name: Install Tomcat artifacts

      action: >

      yum name = "demo-tomcat-1" state = present

      register: Output

   always:

      - debug:

         msg:

            - "Install Tomcat artifacts task ended with message: {{Output}}"

            - "Installed Tomcat artifacts - {{Output.changed}}"

Explanation

* **block:** The Ansible syntax to execute a given block.
* **name:** It is used in logging and helps in debugging which all blocks were successfully executed.
* **action:** The action is an Ansible keyword used in YAML.
* **register:** The output of the action tag is registered by using the register keyword.
* **always:** It is also an Ansible keyword; it says that below will still be executed.
* **msg:** It displays the message.

Ansible Tags

If you have a large playbook, it becomes useful to be able to run only a specific part of it rather than running everything in the playbook. Ansible supports a tag attribute for this reason.

When you apply tags on things, then you can control whether they are executed by adding command-line options.

When you execute a playbook, you can filter tasks based on the tags in two ways, such as:

1. On the command line, with the **-tags** or **-skip-tags** options.
2. In Ansible configuration settings, with the **TAGS\_RUN** and **TAGS\_SKIP** options.

In Ansible, tags can be applied to many structures, but its simplest use is with individual tasks. Let's see an example that tags two tasks with different tags, such as:

tasks:

- yum:

    name: "{{ item }}"

    state: present

  loop:

  - httpd

  - memcached

  tags:

  - packages

- template:

    src: templates/src.j2

    dest: /etc/foo.conf

  tags:

  - configuration

If you want to run the **configuration** and **packages** part of a very long playbook, then you can use the -tags option on the command line.

1. ansible-playbook example.yml --tags "configuration,packages"

And if you want to run a playbook without certain tagged tasks, then you can use the **-skip-tags** command-line option.

1. ansible-playbook example.yml --skip-tags "packages"

Tag Reuse

We can apply the same tag to more than one task. By using the "**--tags**" command line options, all tasks with that tag name will be run.

**For example:** In below example, we use one tag "**ntp**" for several tasks, such as:

---

# file: roles/common/tasks/main.yml

- name: be sure ntp is installed

  yum:

    name: ntp

    state: present

  tags: ntp

- name: be sure ntp is configured

  template:

    src: ntp.conf.j2

    dest: /etc/ntp.conf

  notify:

  - restart ntpd

  tags: ntp

- name: be sure ntpd is running and enabled

  service:

    name: ntpd

    state: started

    enabled: yes

  tags: ntp

Special Tags

"**always**" is a unique tag that will always run a task, unless specifically skipped (**--skip-tags always**)

**For example:**

tasks:

- debug:

    msg: "Always runs"

  tags:

  - always

- debug:

    msg: "runs when you use tag1"

  tags:

  - tag1

New in version 2.5

Here is another unique tag that is "**never**" which prevents a task from running unless a tag is specifically requested.

**For example:**

1. tasks:
2. - debug: msg="{{ showmevar }}"
3. tags: [ never, debug ]

In the above example, the task will only run when the "**never**" or "**debug**" tag is explicitly requested.

Here are another three special keywords for tags:

"**tagged**" which run only tagged,

"**untagged**" which run only untagged, and

"**all**" which run all tasks respectively.

By default, Ansible runs as if "**--tags**" all had been specified.

Ansible Galaxy

Ansible Galaxy is a galaxy website where users can share roles and to a command-line tool for **installing, creating,** and **managing** roles.

Ansible Galaxy gives greater visibility to one of Ansible's most exciting features, such as application installation or reusable roles for server configuration. Lots of people share roles in the Ansible Galaxy.

Ansible roles consist of many playbooks, which is a way to group multiple tasks into one container to do the automation in a very effective manner with clean, directory structures.

Ansible Galaxy Commands

Here are some helpful Ansible Galaxy commands, such as:

* To display the list of installed roles, with version numbers.

1. ansible-galaxy list

* To remove an installed role.

AD

1. ansible-galaxy remove [role]

* To create a role template suitable for submission to Ansible Galaxy.

1. ansible-galaxy init

Create Roles with Ansible Galaxy

The Ansible Galaxy is essentially a large public repository of Ansible roles. Roles ship with READMEs detailing the roles use and variables. Ansible Galaxy contains a large number of roles that are continually evolving and increasing.

The Galaxy can use Git to add other role sources like GitHub. You can initialize a new galaxy role using **the ansible-galaxy init** or install a role directly from the Ansible galaxy role store by executing the **ansible-galaxy install <name of role>**command.

To create an Ansible role using the Ansible Galaxy, you need to use the **ansible-galaxy** command and its templates. Roles must be downloaded before they used in the playbooks. They are placed into the default directory **that is /etc/ansible/roles**.

Create Collections

Ansible Galaxy has been a tool for constructing and managing roles with new iterations of the Ansible, and you are bound to see changes or additions. On Ansible version 2.8, you get the unique feature of the collections.

Collections are the distribution format for the Ansible content. They can be used to package and distribute roles, modules, playbooks, and plugins.

Collections follow the following simple structure:

collection/

├── docs/

├── galaxy.yml

├── plugins/

│ ├── modules/

│ │ └── module1.py

│ ├── inventory/

│ └── .../

├── README.md

├── roles/

│ ├── role1/

│ ├── role2/

│ └── .../

├── playbooks/

│ ├── files/

│ ├── vars/

│ ├── templates/

│ └── tasks/

└── tests/

The ansible-galaxy-collection command implements the following commands. Some commands are the same as used with ansible-galaxy, such as:

* **init:** It creates a basic collection Skeleton based on the default template included with Ansible or your own template.
* **build:** It creates a collection artifact that can be uploaded to the galaxy or your own repository.
* **publish:** It publishes a built connection artifact to the galaxy.
* **install:** It installs one or more connections.

Ansible Commands Cheat Sheets

Here are some commands which are used in Ansible, such as:

* To install EPEL repo on Centos/RHEL systems.

1. [root@ansible-server ~]# sudo yum install epel-release

* To install Ansible package on Centos/RHEL systems.

AD

1. [root@ansible-server ~]# sudo  yum install -y ansible

* To perform an update to the packages on Debian/Ubuntu systems.

1. $ sudo apt update

* To install the software properties-common-package on Debian/Ubuntu systems.

1. $ sudo apt install software-properties-common

* To install Ansible personal package archive on Debian/Ubuntu systems.

1. $ sudo apt-add-repository ppa:ansible/ansible

* To install Ansible on Debian/Ubuntu systems.

1. $ sudo apt update
2. $ sudo apt install ansible

* To issue a ping command on all servers defined in the inventory file named hosts.

1. [root@ansible-server test\_ansible]# ansible -i hosts all -m ping

* To issue a ping command only on hosts2.

1. [root@ansible-server test\_ansible]# ansible -i hosts all -m ping --limit host2

* To copy the file "testfile" on all hosts in the inventory file.

1. [root@ansible-server test\_ansible]# ansible -i hosts all -m copy -a "src=/root/test\_ansible/testfile dest=/tmp/testfile"

* To install ncdu package on all hosts.

1. [root@ansible-server test\_ansible]# ansible -i hosts all -m yum -a 'name=ncdu state=present'

* To remove ncdu package on all hosts.

1. [root@ansible-server test\_ansible]# ansible -i hosts all -m yum -a 'name=ncdu state=absent'

* To build the directory structure for the role named role1.

1. [root@ansible-server test2]# ansible-galaxy init role1

* To dry-run p4.yml playbook.

1. [root@ansible-server test\_ansible]# ansible-playbook -i hosts p4.yml --check

* To run a p4.yml playbook with password authentication for all hosts.

1. [root@ansible-server test\_ansible]# ansible-playbook -i hosts p4.yml -k

Ansible Modules

Ansible modules are discrete units of code which can be used from the command line or in a playbook task.

The modules also referred to as task plugins or library plugins in the Ansible.

Ansible ships with several modules that are called **module library**, which can be executed directly or remote hosts through the playbook.

Users can also write their modules. These modules can control like **services, system resources, files,** or **packages,** etc. and handle executing system commands.

Let's see how to execute three different modules from the command line.

ansible webservers -m service -a "name=httpd state=started"

ansible webservers -m ping

ansible webservers -m command -a "/sbin/reboot -t now"

Each module supports taking arguments. Mainly all modules take **key=value** arguments, space delimited.

Some module takes no arguments, and the shell/command modules take the string of the command which you want to execute.

From playbook, Ansible modules execute in a very similar way, such as:

- name: reboot the servers

  command: /sbin/reboot -t now

Here is another way to pass arguments to a module that is using **YAML syntax**, and it is also called complex args.

- name: restart webserver

  service:

    name: httpd

    state: restarted

Technically, all modules return **JSON** format data, though command line or playbooks, you don't need to know much about that. If you're writing your module, it means you do not have to write modules in any particular language which you get to choose.

Modules should be idempotent and avoid making any changes if they detect that the current state matches the desired final state. When using Ansible playbooks, these modules can trigger "**change events**" in the form of notifying "**handlers**" to run additional tasks.

Documentation for each module can be accessed from the command line with the Ansible-doc tool:

1. ansible-doc yum

Ansible Shell

Ansible shell module is designed to execute the shell commands against the target UNIX based hosts. Ansible can run except any high complexes commands with pipes, redirection. And you can also perform the shell scripts using the Ansible shell module.

The main advantage of the Ansible shell is except any high complexes commands with pipes and semicolons can be a disadvantage from the security perspective as a single mistake could cost a lot and break the system integrity.

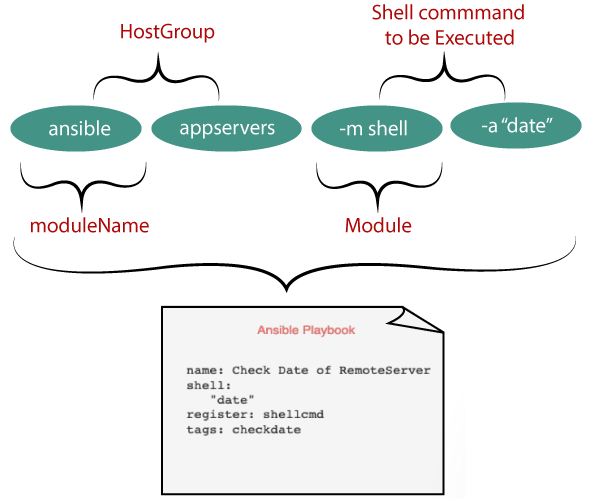
* The Ansible shell module is designed to work only with LINUX based machines and not for the windows. For windows, you should use the **win\_shell**
* Ansible shell module can be used to execute shell scripts. Ansible has a dedicated module named script, which is used to copy the shell script from the control machine to the remote server.

Let see the syntax of how to use the Ansible shell module in the playbook and Adhoc:

Syntax of Ansible shell module in a playbook

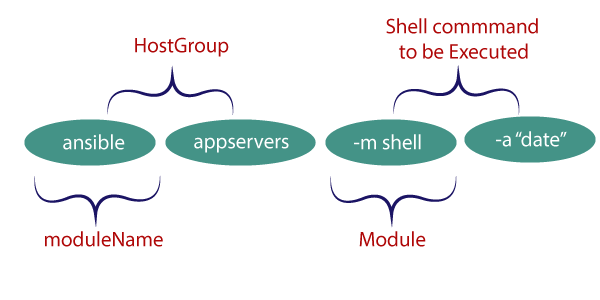
The beauty of the playbook is the way it looks and written. A playbook is written in YAML so it can be easily understood.

The below image demonstrates how an Adhoc command would be transformed as a play of an Ansible playbook.



Syntax of Ansible shell module in Adhoc

The below image shows a quick syntax of the Ansible shell module in Adhoc manner.



Example

To execute a single command in a single task using a Shell or command module. Suppose you want to get the date of the remote server. And the remote server is under the hostgroup which name is testservers.

**Step 1:** Login to the Ansible server.

**Step 2:** Below is an example that executes a single command using the Shell module in a remote host.

---

-name: Shell command example

Hosts: testservers

tasks:

-name: check date with the shell command

shell:

"date"

register: datecmd

tags: datecmd

-debug: msg= "{{datecmd.stdout}}"

In the above example, we are running our playbook against a hostgroup named **testservers** and executing a simple date command and saving the output of that command into a **Register** variable named **datecmd**.

At the last line, we retrieve the registered variable and printing only the date command output stored in the **stdout** property of **datecmd**.

Example 2: Execute multiple commands in a single shell:

The Shell can accept various commands together in a single shell play. Also, you can write your shell script with the Ansible shell module.

In the below example, we grouped some shell commands to execute a controlled and clean tomcat restart.

The playbook is designed to execute the following steps in order, such as:

* Stop the tomcatServer
* Clear the cache
* Truncate the log file
* Start the instance

AD

---

  - name: Shell Examples

    hosts: testservers

    tasks:

    - name: Clear Cache and Restart tomcat

      become: yes

      delay: 10

      async: 10

      poll: 50

      shell: |

        echo -e "\n Change directory to the Tomcat"

        cd tomcat8/

        echo -e "\n Present working directory is" `pwd`

        echo -e "\n Stopping the tomcat instance"

        bin/shutdown.sh

        echo -e "\n Clearning the tmp and work directory of tomcat"

        rm -rfv tmp/\*

        rm -rfv work/\*

        echo -e "\nTruncate the log file"

        > logs/catalina.out

        echo -e "\nDirectory listing"

        ls -lrtd logs/catalina.out

        echo -e "\nStarting the instance"

        bin/startup.sh

      args:

        chdir: "/apps/tomcat/"

      register: fileout

      tags: fileout

    - debug: msg="{{ fileout.stdout\_lines }}"

Ansible Templates

Ansible is used to manage configurations of multiple servers and environments. But these configuration files can vary for each cluster or remote server. But apart from a few parameters, all other settings will be the same.

Creating static files for each of these configurations is not an efficient solution. It will take a lot of time, and every time a new cluster is added, then you have to add more files. If there is an efficient way to manage these dynamic values, it would be beneficial. This is where Ansible template modules come into play.

A template is a file that contains all your configuration parameters, but the dynamic values are given as variables in the Ansible. During the playbook execution, it depends on the conditions such as which cluster you are using, and the variables will be replaced with the relevant values.

You can do more than replacing the variables with the help of the Jinj2 templating engine. You can have loops, conditional statements, write macros, filters for transforming the data, do arithmetic calculations, etc.

Usually, the template files will have the .j2 extension, which denotes the Jinja2 templating engine used.

The double curly braces will denote the variables in a template file, **'{{variables}}'**.

We need to have two parameters when using the Ansible Template module, such as:

* **src:** The source of the template file. It can be a relative and absolute path.
* **dest:** Dest is the destination path on the remote server.

Template Module Attributes

Here are some other parameters which can be used to change some default behavior of the template module:

* **Force:** If the destination file already exists, then the Force parameter will decide whether it should be replaced or not. By default, the value is yes.
* **Mode:** This parameter is used to set the permissions for the destination file explicitly.
* **Backup:** If you want a backup file to be created in the destination directory, you should set the value of the backup parameter to yes. By default, the value is no. and the backup file will be created every time there is a change in the destination directory.
* **Group:** Name of the group that should own the directory. It is similar to executing chown command for a file in Linux systems.

AD

Example

In the below example, we are using the template module on the example1.j2 file that replaces the default variables with values given in the playbook.

**File: Playbook.yml**

---

- hosts: all

  vars:

    variable1: 'Hello'

    variable2: 'My first playbook using template'

  tasks:

    - name: Basic Template Example

      template:

        src: example1.j2

        dest: /home/knoldus/Documents/Ansible/output.txt

**File: example1.j2**

{{variable1}}

No change in **this** line

{{variable2}}

**File: output.txt**

Hello

No change in this line

My first playbook using the template

You can see, their values replace both variables in the example1.j2 in the above example.

# Ansible YAML

YAML is used to describe configuration that has been increasing in the past few years with the help of **Ansible** and **SaltStack**.

YAML is more comfortable for humans to read and write in comparison to other standard data formats such as XML or JSON. There are libraries available in most programming languages for working with YAML.

For Ansible, every YAML file starts with a list. Each item in the list is a list of key-value pairs, commonly called a **"hash"** or **"dictionary"**. So, we need to know how to write lists and dictionaries in YAML.

There’s another small quirk to YAML. All YAML files (regardless of their association with Ansible or not) can optionally begin with **---** and end with **---**. This is part of the YAML format and indicates the start and end of a document.

All members of a list are lines beginning at the same indentation level starting with a **"-"** (a dash and space):

---

# A list of colors

- White

- Orange

- Red

- Black

---

We have different ways in which the YAML data is represented, such as:

**Key-value Pair**

YAML uses the Key-Value pair to represent the data. And the dictionary is described in the key: value pair.

#### **NOTE: There should be space between: and value.**

**For example,** a student record

---

# A student record

Martin:

name: Martin

roll no: 10

**class**: 12th

div: A

---

**Abbreviation**

We can also use the abbreviation to represent the directories:

1. Martin: [name: martin, roll no: 10, **class**: 12th, div: A]

### Representing List

We can also represent List in YAML. Every element (member) of the list should be written in a new line with the same indentation starting with "-" (- and space).

**For example:** Name of the countries

---

#Name of country

Countries:

   - India

   - China

   - USA

   - Iceland

---

**Abbreviation**

To represent the list, we can also use the abbreviation method:

1. Countries: ['India', 'China', 'USA', 'Iceland']

**List inside Dictionaries**

We can use the list inside dictionaries, i.e., the value of a key is a list.

**For example,** a student record

---

# A student record

Martin:

name: Martin

roll no: 10

**class**: 12th

div: A

likes:

- Physics

- Chemistry

- Math

---

**List of Directories**

We can also make a list of directories:

**For example:**

---

# A student record

- Martin:

name: Martin

roll no: 10

**class**: 12th

div: A

likes:

- Physics

- Chemistry

- Math

- Edward:

 name: Edward

 roll no: 11

**class**: 12th

div: A

likes:

- Biology

- English

---

YAML uses **"|"** to include newlines while showing multiple lines and **">"** to suppress newlines while showing various lines. Due to this, we can read and edit long lines. In both cases, the indentation will be ignored.

AD

We can also represent **Boolean** (True/false) values in YAML, where **Boolean** values can be case insensitive.

**For example,** a student result

---

#a student result

- Martin:

name: Martin

roll no: 10

**class**: 12th

div: A

likes:

- Physics

- Chemistry

- Math

   result:

      Physics: 70

     Chemistry: 45

Math: 85

Biology: 65

      English: 80

   passed: TRUE

   messageIncludeNewLines: |

      Congratulation!!

      You passed with 79%

   messageExcludeNewLines: >

      Congratulation!!

      You passed with 79%

---

# Ansible Inventory

Ansible works against multiple managed hosts in your infrastructure at the same time, using a list or group of lists is known as the inventory.

Once an inventory is defined, you use patterns to select the hosts or groups you want to run against to Ansible.

The default location for inventory is a file called **/etc/ansible/hosts**. You can also specify a different inventory file at the command line using the **-i <path>** option. You can pull the inventory file from dynamic or cloud sources or different formats (YAML, ini). Ansible has inventory plugins to make it flexible and customize.

### Hosts and group

The format is **/etc/ansible/** hosts are in **INI** like format, such as:

mail.example.com

[webservers]

foo.example.com

bar.example.com

[dbservers]

one.example.com

two.example.com

three.example.com

Heading in the brackets is a group name, which is used in classifying the systems. And deciding what policy you are controlling at what time and for what purpose. You can put the systems in more than one group.

For example, a server could be both a **dbserver** and a **webserver**.

If you have hosts that run on a non-standard SSH port, then you can put the port number after the hostname with the colon. The Ports listed in the SSH configuration file that can be used with the **OpenSSH** connection but not use with the **paramiko** connection.

To makes things explicit, it is suggested that you set them if items are not running on the default ports:

1. badwolf.example.com:5309

Suppose you have static IPs and want to set up some aliases that live in your host file, or you can connect through tunnels. Also, you can describe the hosts like the below example:

1. Jumper ansible\_port=5555 ansible\_host=192.0.2.50

In the above example, trying to Ansible against the host alias "**jumper**" will connect 192.0.2.50 on port 5555. It is using features of the inventory file to define the special variables.

### Hosts Variables

You can assign the variables to the hosts that will be used in playbooks, such as:

[atlanta]

host1 http\_port=80 maxRequestsPerChild=808

host2 http\_port=303 maxRequestsPerChild=909

### Group Variables

The variables can be applied to an entire group at once, such as:

[atlanta]

host1

host2

[atlanta:vars]

ntp\_server=ntp.atlanta.example.com

proxy=proxy.atlanta.example.com

### Groups of Groups and Group Variables

It is possible to make groups of the group using the **:children's** suffix. And you can apply variables using **:vars**.

[atlanta]

host1

host2

[raleigh]

host2

host3

[southeast: children]

Atlanta

Raleigh

[southeast:vars]

some\_server=foo.southeast.example.com

halon\_system\_timeout=30

self\_destruct\_countdown=60

escape\_pods=2

[usa: children]

southeast

northeast

southwest

northwest

# Ansible Debug

Ansible provides a debug module option that makes the tasks more manageable. It is a handy tool to figure out any problem areas.

Ansible version 2.1 extended the debug module with a verbosity parameter that transforms it from a print line.

**For example:** Let's create the playbook **1\_debug\_example.yml**, such as:

---

- name: Debug Example Uptime

hosts: localhost

connection: local

tasks:

- name: Find Uptime

shell: /usr/bin/uptime

register: result

- name: Print debug message

debug:

var: result

verbosity: 2

During the Ansible playbook debugging, it is useful to know how to display the registered variables or host facts.

To print a message from the Ansible playbook, as well as a value of a variable, we can use the Ansible debug module. Ansible debug module is easy to use.

**For example:** Let's execute a simple hello world playbook **2\_debug\_example.yml**, such as:

---

- name: Debug Example - Hello World

hosts: localhost

tasks:

- name: Print debug message

debug:

the Ansible includes a debugger as a part of the strategy plugins. This debugger enables you to debug as a task. You have access to all the features of the debugger in the context of the task. You can check or set the value of variables, update module arguments, and re-run the task with the new variables and arguments to resolve the cause of the failure.

There are many ways to invoke the debugger, such as:

**Using the debugger Keyword**

The debugger keyword can be used on any block where you provide a name attribute such as a role, block, task, or play.

The debugger keyword accepts several values, such as:

**Always:** Always invokes the debugger, regardless of the outcome.

**Never:** Never invokes the debugger, regardless of the outcome.

**On\_failed:** It only invokes the debugger if a task fails.

**On\_unreachable:** It only invokes the debugger if the host was unreachable.

**On\_skipped:** It only invokes the debugger if the task is skipped.

#### **NOTE: These options override any global configuration to enable or disable the debugger.**

**On a Task**

- name: Execute a command

  command: **false**

  debugger: on\_failed

**On a play**

- name: Play

  hosts: all

  debugger: on\_skipped

  tasks:

    - name: Execute a command

      command: **true**

      when: False

When a provided at the general level, and a more specific level, the more particular wins:

- name: Play

  hosts: all

  debugger: never

  tasks:

    - name: Execute a command

      command: **false**

      debugger: on\_failed

# Ansible Apt

APT stands for "Advanced Packaging Tool" is the preferred package management toolset in Ubuntu. It allows us to install new packages, update them, and remove the packages from Ubuntu or Debian systems. Here are 3 APT related command-line tools, such as:

**Apt-get:** All the basic package management operations can be done by using this tool. Ansible apt-get module provides this functionality.

**Apt-add-repository:** It is used for adding a new repository to the repository list. The default repository may not have the latest version of all the packages. So you need to add additional repositories for some software maintainers. Ansible apt\_repository module provides the functionality for adding a new repository.

**Apt-key:** It is used to manage the list of keys for authenticating apt packages. Ansible apt\_key module is used to manage the keys.

### Installing new Apt Packages

To install the new packages, you have to give the name of the package in the name parameter and the desired state of the package.

The default state of the package is "present". Also, it is better to set the update\_cache to true. Thus you can ensure the indexes are synchronized with the sources list. It is the same as running the apt-get update command before installing a package.

The below example will do a cache update to synchronize the index. Check if the 'zip' package is installed on the target server. And if it is not installed, the package will be installed. If the package is already installed, then it won't be upgraded.

-hosts: loc

tasks:

-name: Ansible apt install packages

apt:

name: zip

state: present

update\_cache: **true**

**1. Installing the latest version of a package**

If you set the state of the packages to "present", then Ansible will only check if the package is present. So if the new package is available, it will not be able to install.

If you want to install the latest apt packages, then you have to set the state parameter to the latest.

This will ensure the package with the latest version is installed. The below example will update the cache first, then install the latest package of zip, such as:

-hosts: loc

tasks:

-name: ansible apt install latest version

apt:

name: zip

state: latest

update\_cache: **true**

**2. Ansible install multiple packages**

Instead of writing multiple tasks to install packages, you can use **with\_items** and combine those tasks.

In the below example, we are going to install 3 packages: docker-ce, Nginx, and git.

-hosts: loc

tasks:

-name: ansible apt with\_items

apt:

name: "{{item}}"

update\_cache: **true**

state: present

with\_items:

-'docker-ce'

-'nginx'

-'git'

**3. Ansible Apt ad-hoc**

You can also use the ad-hoc method to install new packages using the apt module, such as:

ansible all -m apt -a "name=nginx state=absent" -i inventory.ini

### Removing Apt Packages

You can also remove the packages using apt module by setting the state parameter to absent.

The below example will remove the zip package. Since the module is idempotent, it will not go through an error if the package is not present.

-hosts: loc

tasks:

-name: ansible apt remove **package**

apt:

name: zip

state: absent

# Ansible Lineinfile

The lineinfile is one of the most powerful modules in the Ansible toolbox. Ansible lineinfile module is used to insert a line, modify, remove, and replace an existing line.

Ansible lineinfile module saves your time when you work with the file and modify their content on the run, such as adding a new line in the file or updating, replace a line in the file when specific text is found, and much more.

Ansible lineinfile provides many parameters to do the job quickly. You can also use the condition to match the line before modifying, removing using the regular expressions. You can reuse and modify the matched line using the backreference parameter.

#### **NOTE: Ansible lineinfile can be used only for working a single line in a file. If you want to replace multiple lines, replace the module, and if you're going to insert, update, remove a block of lines in a file use blockinfile module.**

### Insert a Line

Let's see how to write a line to a file if it is not present. You can set the path of the file to be modified using the path (>Ansible 2.3)/ dest parameter. And set the line to be inserted using the line parameter.

The below example will write the line "Inserting a line in a file" to the file "remote\_server.txt". The new line will be added to the EOF. If the line is already existing, then it will not be combined.

You can also set the **create** parameter, which says if the file is not present, then create a new file. The default value for the state is **present**.

- hosts: loc

  tasks:

    - name: Ansible insert lineinfile

      lineinfile:

        dest: /home/javaTpoint/remote\_server.txt

        line: Inserting a line in a file.

        state: present

        create: yes

### Removing a Line

Set the state parameter to absent or remove the line specified. All the occurrence of that line will be removed.

- hosts: loc

  tasks:

    - name: Ansible lineinfile remove the line

      lineinfile:

        dest: /home/javaTpoint/remote\_server.txt

        line: Removed lines.

        state: absent

### Replacing or Modifying a Line

To modify a line, you need to use the Ansible backrefs parameter along with the regexp parameter. This should be used with state=present.

If the regexp does not match any line, then the file is not changed. If the regexp matches a line or multiple lines, then the last matched line will be replaced. The grouped elements in regexp are populated and can be used for modification.

In the below example, we are commenting on a line. The full line is captured line by placing them inside the parenthesis to '\1'. The '#\1' replaces the line with '#' followed by what was captured.

You can have multiple captures and call them by using '\1', '\2', '\3' etc.

**Commenting a line with Ansible lineinfile backrefs**

- name: Ansible lineinfile regexp replace the example

  lineinfile:

    dest: /etc/ansible/ansible.cfg

    regexp: '(inventory = /home/fedora/inventory.ini.\*)'

    line: '#\1'

    backrefs: yes

**Uncommenting the line with lineinfile regexp**

- name: Ansible lineinfile backrefs example

  lineinfile:

    dest: /etc/ansible/ansible.cfg

    regexp: '#(inventory = /home/fedora/inventory.ini.\*)'

    line: '\1'

    backrefs: yes

# Ansible Copy

Ansible provides the functionality of copying the files and directories with the help of copy and fetch modules. The copy module is versatile.

The copy module is used to copy files and folders from the local machine to the remote servers. And the fetch module to copy data from the remote machine to the local machine.

If you want to copy files after substituting with variables, such as config files with IP changes, or you can use the template module also. You can perform a lot of complicated tasks with this module.

## Copying Files from Local to Remote

The copy module is used to check the file set in the **src** parameter, on the local machine. And then, it will copy the data to the remote machine path specified in the **destpath**.

In the below example, we will copy the **sample.txt** file in the home directory of the local machine, to the destination is the **/tmp** directory on the remote server. as long as we are not specifying any permission for the file, the default permission for the remote file is set as -**rw-rw-r-(0664)**.

- hosts: blocks

  tasks:

  - name: Ansible copy file to a remote server

    copy:

      src: ~/sample.txt

      dest: /tmp

**Case 1:** If the file is already present on the remote server, but the source file's content is different, then the destination file will be modified. You can control this by setting the force parameter. The default is set to yes. So it modifies the file by default.

If you don't want the file to be modified, if the source file is different, then you can set it No. The following task will copy the file if the file does not exist on the remote server.

- hosts: blocks

  tasks:

  - name: Ansible copy file force

    copy:

      src: ~/sample.txt

      dest: /tmp

      force: no

**Case 2:** If the file did not found on the local machine, the Ansible throw an error.

**For example:** fatal: [remote-machine-1]: FAILED!=> {"changed": false, "failed": true, "msg": "unable to find '~/sample.txt' in expected paths."}

## Copying Directories from Local to Remote

You can also copy folders or directories using the Ansible copy module. If the '**src**' path is a directory, then it will be copied recursively. Or the entire directory will be copied.

There are two different variations for this task. Depending on whether you have the '/' character at the endpoint of the 'src' path or not.

The first method will create a directory on the remote server, with the name set in the src parameter. Then, it will copy and paste the content of the source folder into that directory.

If you want this behavior, then it doesn't give the '/' after the path in the src parameter.

in the below example, it will first create a directory named **copy\_dir\_ex** in the **/tmp** of the remote server.

- hosts: blocks

  tasks:

  - name: Ansible copy the directory to the remote server

    copy:

      src:/Users/mdtutorials2/Documents/Ansible/copy\_dir\_ex

      dest:/Users/mdtutorials2/Documents/Ansible/tmp

## Copying Files between Directories on Remote Machine

Ansible copy allows you to copy the files from one directory to another on the same remote machine. But this is only for files, not for the directories. You can use the **remote\_src parameter** to let Ansible know your intentions.

The below code will copy **/tmp/test.txt** to the home directory of the user **(/home/[username]/)**.

---

-hosts: webservers

tasks:

-name: copy the file between directories on a remote server

copy:

src: /tmp/test.txt

dest: ~/test.txt

remote\_src: yes

# Ansible Command

Ansible command module is used to run any commands or run any scripts in the remote target machine. Or used to execute commands on a remote node.

The command module is used to run simple Linux commands on a remote node or server, which is a part of the host group or standalone server mentioned in the host group.

## Ansible Command Module and Shell Module

The shell module is used when we need to execute a command in remote servers, in the shell of your choice. By default, the commands are run on the **/bin/sh** shell. You can make use of the various operations such as '|', '<', '>' etc. and environmental variables such as **$HOME**.

The command module does not process the commands through a shell. So it does not support the above operations.

You give the command you want to execute the same way you provide it on a UNIX shell, command name followed by the arguments.

- name: Executing a command using the shell module

  shell: ls -lrt > temp.txt

The first command lists all the files in the current folders and writes that to the file, temp.txt.

- name: Executing a command using the command module

  command: hello.txt

The above example displays the content of the hello.txt file.

## Changing the Default Directory

The command will always execute in the default directory. You can change and specify the directory path where you want to run the command using the **chdir** parameter. This parameter is available for both command and shell module.

You can also change the default shell by specifying the absolute path of the require shell in the executable parameter.

- - hosts: loc

  tasks:

  - name: ansible command with chdir and executable parameters

    command: ls -lrt

    args:

      chdir: /home/Ansible/command\_chdir\_example

      executable: /bin/bash

In the above example, using the "Bourne Again Shell" by giving the obsolete the path **/bin/bash**. And changed the directory to /home/Ansible/command\_chdir\_example.

## Executing Multiple Commands

If you need to run multiple commands, then you can give them to both shell and command modules using the "**with\_items**".

**Example 1:**

- hosts: loc

  tasks:

  - name: Ansible command module multiple commands

    command: "touch {{ item }}"

    with\_items:

      - hello.txt

      - hello1.txt

      - hello2.txt

    args: chdir: /root/ansible

**Example 2:**

- hosts: loc

  tasks:

  - name: Ansible shell module multiple commands

    shell: "cat {{ item }} | grep ansible"

    with\_items:

      - hello.txt

      - hello1.txt

      - hello2.txt

    args:

      chdir: /root/ansible

In the above examples, we want to execute three files; hello.txt, hello1.txt, and hello2.txt. Since I give the {{item}} keyword in the command, it will be replaced with an element of the list in each iteration. Ensure that the level of indentation of "with\_item" is on the same level as the module name.

# Ansible File

Ansible file module is used to creating and deleting the file or multiple files in the remote server. You can also create and delete the directories and change the permissions of the data.

You can also create and delete the soft links (symlinks) as well as hard links. With the help of the Ansible file module, you can set the permission of the files.

## Creating a File in Remote Server

In the Ansible file module, we have different parameters. We are using **path** and **state** parameters that are must in every file module. In the file parameter, we will mention the path of the file in the remote server. On this path, only the file will be created.

**At path:** It mentions the path of the file in the remote server.

**At state:** It mentions touch, and touch will create file exact like Linux command.

Then, it will create a new empty file with the name devops.txt. So mention filename in the path. So in the state: we will mention touch to create the file.

- name: create the file in a remote server

  file:

   path: /path/to/file/in/remote/server/devops.txt

   state: touch

## Deleting a File in Remote Server

If you want to delete any command in the remote server. So at path parameter, mention the path of the file which you want to delete.

**At path:** Mention the path of the file in the remote server.

**At state:** Mention absent to delete the file.

So in the state: we will use touch to create the file, absent to delete the e file.

- name:  delete the file in a remote server

  file:

   path: /etc/abcd.conf

   state: absent

## Creating a File with Permissions

We can also create the file with permission by using the file module.

At the mode parameter: we have 4 digits. Always mention zero at the starting, and remaining digits will be your file permissions.

At owner parameter: mention the owner of the file.

tasks:

  - name: Ansible file module to create a **new** file with permissions.

    file:

    path: /path/to/cretae/file/devops.txt

    state: touch

    mode: 0421

    owner: devops

This permission will be set to that newly created file.

file:

 path: /path/to/cretae/file/devops.txt

 state: touch

 mode: "u=rw,g=w,o=e"

 owner: devops

Both the codes work the same, but in the other code, we are using the symbolic mode, which is equivalent to 0421.

AD

## Creating Multiple Files

A path parameter: we can create a loop to create multiple files by using "{{item}}".

At with\_items parameter: mention file names which you want to create.

By using "{{item}}" and with\_items parameter, we can create loop or multiple files.

tasks:

- name: Ansible file module to create multiple files

  file:

   path: "{{ item }}"

   state: touch

   mode: 0421

  with\_items:

  - devops1.txt

  - devops2.txt

  - devops3.txt

## Deleting Multiple Files

The code will be the same to create multiple files and to delete files but a small change in the state parameter.

State parameter: Touch the create files and absent to delete files.

- name: Ansible file module to delete multiple files

  file:

   path: "{{ item }}"

   state: absent

  with\_items:

  - devops1.txt

  - devops2.txt

  - devops3.txt

# Ansible Vault

Ansible Vault is a feature which allows user to encrypt values and data structures within Ansible projects. This provides the ability to secure any secrets or sensitive data that is necessary to run Ansible plays successfully but should not be publicly visible, such as private keys or passwords. Ansible automatically decrypts the vault-encrypted content at runtime when the key is provided.

To integrate these secrets with regular Ansible data, both the Ansible and Ansible-playbook commands, for executing ad hoc tasks and structured playbook respectively, have support for decrypting vault-encrypted content at runtime.

Ansible Vault is implemented with file-level granularity; it means files are either entirely encrypted or unencrypted. It uses the AES256 algorithm to provide symmetric encryption keyed to a user-supplied password.

This means the same password is used to encrypt and decrypt the content, which is helpful from a usability standpoint. Ansible can identify and decrypt any vault-encrypted files it finds while executing a task or playbook.

Though there is a proposal to change this, at the time of writing this, users can only pass in a single password to Ansible. It means that each of the encrypted files involved must share a password.

## Using Ansible Vault

The simple use of the Ansible vault is to encrypt variables files. It can encrypt any YAML file, but the most common files to encrypt are:

* A role's defaults/ main.yml file
* A role's vars/main.yml file
* Files within the group\_vars directory
* Any other file used to store variables

## Encrypting an Existing File

You can encrypt a regular plaintext variable file by using the ansible vault and define the password that needed later to decrypt it.

#encrypt a role's defaults/main.yml file

ansible-vault encrypt defaults/main.yml

>New vault password:

>Confirm **new** vault password:

>Encryption successful

The ansible-vault command will prompt you a password twice. After that, the file will be encrypted.

## Creating an Encrypted File

To create an encrypted data file, use the ansible-vault to create command, and pass the filename.

1. $ansible-vault create <file name>

You will be prompted to create a password and then confirm it by re-typing it.

Once your password is confirmed, a new file will be created and will open an editing a window. By default, the editor for Ansible vault is VI. You can add data, save it, and exit from it.

AD

## Editing Encrypted Files

If you want to edit the encrypted file, you can edit it using ansible-vault edit command. This command will decrypt the file to a temporary file and allow you to edit the file.

1. $ansible-vault edit <file name>

You will be prompted to insert the vault password. The decrypted file will open in a VI editor, and then you can make the required changes. Save the changes and removing the temporary file.

## Rekeying Encrypted Files

If you want to change your password on a vault on a vault-encrypted file, you can do it by using the rekey command.

1. $ansible-vault rekey <file1> <file2> <file3>

The above command can rekey multiple data files at once and ask for the original password and the new password.

## Encrypting Unencrypted Files

If you have existing files which you want to encrypt, use the ansible-vault encrypt command. This command can operate on multiple files at once.

1. $ansible-vault encrypt <file1> <file2> <file3>

## Decrypting Encrypted Files

If you have existing files that you no longer want to keep encrypted, you can decrypt them permanently by running the ansible-vault decrypt command. This command will save them unencrypted to the disk.

1. $ansible-vault decrypt <file1> <file2> <file3>

## Viewing Encrypted Files

If you want to view the contents of an encrypted file without editing it, then you can use the ansible-vault view command.

1. $ansible-vault view <file1> <file2> <file3>

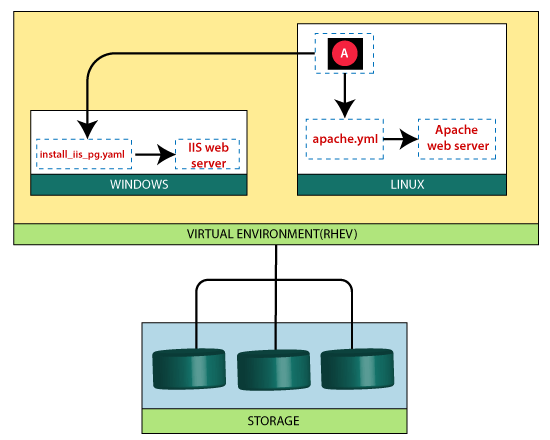
# Ansible Windows

Using Ansible to manage Windows, many of the syntax and rules that apply for Unix/Linux hosts also apply to Windows. Still, there are some differences when it comes to components like path separators and OS-specific tasks. This document covers details specific to using Ansible for Windows.

## How does it Work

Ansible manages Linux/Unix machines using SSH by default.

Starting in version 1.7, Ansible contains support for managing Windows machines. This uses native PowerShell remoting, rather than SSH.



Ansible will still be run from a Linux control machine and uses the "winrm" Python module to talk to remote hosts. While not supported by Microsoft or Ansible, this Linux control machine can be a Windows Subsystem for Linux (WSL) bash shell.

No additional software needs to be installed on the remote machines for Ansible to manage them, and it still maintains the agentless properties that make it popular on Linux/Unix.

Note that it is important you have a basic understanding of Ansible, so if you haven't written a Linux playbook first, it might be difficult to you.

## Linux like Linux, Windows like Windows

Users prefer Ansible as a simple, agentless, powerful automation tool that allows them to model and deliver any configuration, deployment, and environment across their IT infrastructure.

Ansible knew that the key was to bring the same simple, agentless paradigm to managing Windows, while still feeling native to Windows administrators. Ansible native Windows support uses Windows PowerShell remoting to manage Windows like Windows in the same Ansible agentless way that Ansible manages Linux like Linux.

With Ansible's native Windows supports:

* Gather facts on Windows hosts
* Install and uninstall MSIs
* Enable and disable Windows Features
* Start, stop, and manage Windows services
* Create and manage local users and groups
* Manage Windows packages via the chocolatey package manager
* Manage and install Windows updates
* Fetch files from remote sites
* Push and execute any PowerShell scripts you write

You can connect and automate Windows using local or domain users, and you will be able to use Windows 'runas' support to execute actions as the Administrator, just like 'sudo' or 'su' on Linux.

You can write your modules in PowerShell with easy extensibility of Ansible. And extend Ansible for whatever other functionality you need. Ansible users have written modules for managing to file system ACLs, managing Windows Firewall, managing hostname, domain membership, and more.

# Ansible Yum

Ansible has a specific module for managing the Yum packages. You can install, remove, upgrade or downgrade versions and many more by using this module.

The Yum module also requires two parameters for the primary command, like other package management modules in Ansible.

* Name: provides the name of the package which you want to install.
* State: maintains the state of the packages, like what should be the state of the package after the task is completed (present or absent). By default, the value of the parameter is "present".

## Installing a Package

Let's install the git package using the Yum module. Set the name parameter to "git" and the state parameter to "present".

- hosts: all

  tasks:

  - name: Install yum **package** in Ansible example

    yum:

      name: git

      state: present

If the package was not on the remote server, then the latest version will be installed.

And if the package was already installed on the remote server, then it will not be updated to a new version because the "state" is already "present".

#### **NOTE: Both the "present" and "installed" parameters have the same behavior.**

## Installing the latest Version

If you want to install the newest version, then you can set the state parameter to "latest". It will install the newest package, whether the package is present or not.

- hosts: all

  tasks:

  - name: Install the latest yum **package** example.

    yum:

      name: git

      state: latest

## Installing a Specific Version

Sometimes you want to install a particular version of the packages. You can do this by appending the version with the package name.

1. <packagename>-<**package** version>

**For example** git-1.8.3.1-6.el7

Let's install the git package with version and release, 1.8.3.1-6.el7, on the remote server.

- hosts: all

  tasks:

  - name: Install a specific version of a **package** in Ansible.

    yum:

      name: git-1.8.3.1-6.el7

      state: present

**Output**

------

[root@rpm ~]# yum info git

Installed Packages

Name: git

Arch: x86\_64

Version: 1.8.3.1

Release: 6.el7

AD

## Installing Multiple Packages

If you want to install various packages, you can do this by using the "with\_items" statement to loop through a list of the packages.

Let's execute three Yum packages such as MySQL, git, and httpd.

- hosts: all

  tasks:

  - name: yum

    yum:

      name: "{{ item }}"

      state: present

    with\_items:

    - git

    - httpd

    - mysql

#### **NOTE: The above code will not be executed as a single package installation in each loop instance. Instead, all the modules are installed in one go. This optimized form is the behavior since 1.9.2.**

## Update all Packages

You can update all the yum packages, like giving the command yum -y update. You can use the wildcard "\*" in the name.

- hosts: all

  tasks:

  - name: Upgrade all yum packages ansible.

    yum:

      name: "\*"

      state: latest

Also, you can use the exclude parameter so that some packages should not be upgraded. The following task will not update the git package.

- hosts: all

  tasks:

  - name: Exclude some packages from an upgrade in Ansible.

    yum:

      name: "\*"

      state: latest

      exclude: git\*

# Ansible AWX

Ansible AWX is the open-source freely available project for testing and you can use Ansible AWX in a lab, development, or other POC environment. It was the foundation on which Ansible Tower was created.

Ansible AWX is a development branch of code that only undergoes minimal testing and quality engineering testing. There are some drawbacks to enterprise adoption, which is the lack of signed images for using it in the enterprise, lack of real versioning being present with upgrades, and support being offered.

AWX is a web-based solution that makes Ansible more comfortable to use for IT teams. It is designed to be the hub for all automation tasks. It has a fantastic browser REST API and allows you to control access, graphically manage or sync inventory with a wide variety of cloud sources, log all your jobs, and integrate well with Lightweight Directory Access Protocol (LDAP).

Ansible Tower is a commercial version based on AWX by Red Hat. Ansible AWX and Ansible Tower both have similar features.

Ansible is planning on continuing to mature the product lifecycle of Ansible AWX so that it is in tighter lock-step with Ansible Tower releases.

The most significant benefit of AWX over Ansible Tower is that you have all of the enterprise features for an unlimited number of nodes. And one of the significant differences you will see between AWX and Ansible Tower is a different logo for the products.

## Ansible AWX Prerequisites

* 2 or 3 servers
* Ansible version 2.4+
* 10.5.5.20 Ansible AWX
* 10.5.5.21 lemp
* Docker
* Docker python module
* Git version 1.8.4+
* Node 10.x LTS version
* NPM 6.x LTS
* Root privileges

## System Requirements for AWX

* 4 GB of RAM
* 3.4 GHz CPU with 2 Cores
* 20 GB of hard disk space
* Running Docker, Openshift, or Kubernetes

AD

## AWX Installation

Here are some steps to deploy and run the Ansible playbook for the basic LEMP installation through the Ansible AWX dashboard, such as:

**Step 1:** Install EPEL on CentOS 8

It provides some of the quintessential packages required by AWX. Therefore, login as a root user to your CentOS 8 node and run the command:

1. [root@awx-ansible~] #dnf install epel-release-y

**Step 2:** Install additional packages and dependencies

We need to install essential packages that will be required as we get along with the installation of AWX

1. [root@awx-ansible~] #dnf install git gcc gcc-c++ nodejs gettext device-mapper-persistent-data lvm2 bzip2 python3-pip

**Step 3:** Install Docker CE on CentOS 8

To install Docker on CentOS 8, we need to append the Docker repository to the system using the dnf config-manager tool.

1. [root@awx-ansible~] #dnf config-manager --add-repo=https://download.docker.com/linux/centos/docker-ce.repo

To install the Docker, run the below command:

1. [root@awx-ansible~] #dnf install docker-ce-3:18.09.1-3.e17

Run the below command to know what exact version of Docker installed.

1. [root@awx-ansible~] #rpm-qa|grep docker

Or

1. [root@awx-ansible~] #docker --version

Proceed to start and enable Docker using the below commands:

1. [root@awx-ansible~] #systemct1 start docker
2. [root@awx-ansible~] #systemct1 enable --now docker.service

**Step 4:** Install Docker-Compose on CentOS 8

To installation of docker-compose, we will use the pip3 since the system is running using python3.

1. [root@awx-ansible~] #pip3 install docker-compose

**Step 5:** Install Ansible AWX

To install Ansible AWX, first clone the repo as shown below:

1. [root@awx-ansible~] #git clone https://github.com/ansible/awx.git

We need to configure the admin & Postgres password

AD

1. pg\_admin\_password=postgrespass@789
2. admin\_password=Linuxtechi@789

Therefore, sure to generate a cryptographic key for encryption of the inventory file

1. [root@awx-ansible~] #openssl rand -base64 30

Copy the secret key and append it to the secret\_key entry in the inventory file. Save and exit from the inventory file.

To install the Ansible AWX run the following command:

1. [root@awx-ansible installer] # ansible-playbook -i inventory install.yml

After installation is done, you can check the containers which are launched via docker-compose.

1. [root@awx-ansible installer] #docker ps

**Step 6:** Accessing AWX GUI portal

To access AWX web console, open the browser and type in your Ansible AWX server IP and press ENTER.

Provide the username and password for admin and press ENTER. It displays an AWX dashboard to you.

# Ansible Unarchive

Ansible unarchive module is used to unpack or uncompressed the files from an archive file such as zip, tar, tar.gz. It can optionally copy the files to the remote server before uncompressing them.

The unarchive module uses the basic unzip and tar **-xvf** command-line tools to operate. So the target server must have installed these commands. Since most of the **\*nix** distributions are in-built these tools. Here are some essential points of Ansible unarchive, such as:

* It requires zipinfo and gtar/unzip command on the target remote host.
* You can handle **.zip** files using unzip and .tar, .tar.gz, .tar.bz2, .tar.xz files using gtar.
* It does not handle .gz, .bz2, .xz files that not contain a .tar archive.
* It uses gtar **--diff** arg to calculate if changed or not. If --diff arg is not supported, it will always unpack the archive.
* By default, it will copy the source file from the local system to the target before unpacking.
* Set **remote\_src=yes** to unpack an archive which is already existing on the target.
* If checksum validation is desired, use **get\_url** or Uri instead to fetch the file and set **remote\_src=yes**.
* Use the **win\_unzip** module for windows targets.
* We can not touch the Existing files or directories in the destination, which are not in the archive.
* Existing files or directories in the destination which are not in the archive are ignored to decide if the archive should be unpacked or not.

## Why Use Unarchive Module

The Ansible Unarchive module is useful for moving large files and folders across host machines.

For example, if you have a bunch of NGINX configuration files, you can use the unarchive command to download a zipped folder from a URL and unzip it.

### Examples

1. Extract foo.tgz into /var/lib/foo as shown in the following code, such as:

- name: Extract foo.tgz into /var/lib/foo

  unarchive:

    src: foo.tgz

    dest: /var/lib/foo

2. To unarchive a file that is already on the remote machine:

- name: Unarchive a file that is already on the remote machine

  unarchive:

    src: /tmp/foo.zip

    dest: /usr/local/bin

    remote\_src: yes

3. To unarchive a file that needs to be downloaded:

- name: Unarchive a file that needs to be downloaded

  unarchive:

    src: https://example.com/example.zip

    dest: /usr/local/bin

    remote\_src: yes

4. To unarchive a file with extra options

- name: Unarchive a file with extra options

  unarchive:

    src: /tmp/foo.zip

    dest: /usr/local/bin

    extra\_opts:

    - --transform

    - s/^xxx/yyy/

# Ansible Pip

Ansible pip module is used when you need to manage python libraries on the remote servers.

There are two prerequisites if you need to use all the features in the pip module.

* The pip package should already be installed on the remote server.
* Virtualenv package should be installed on the remote server already if you need to manage the packages in the python virtual environment.

#### **NOTE: If you get the error "unable to find any of pip2, pip to use. Pip needs to be installed". The pip module is not available on the remote server during the execution.**

## Installing a Pip Module

To install a new python library, you need to set the name of the package against the "name" parameter. By default, the "state" parameter is "present", the module will try to install the library.

If the library is already installed, then nothing will be done. And if a new version of the library exists, it will not be upgraded.

- hosts: all

  tasks:

  - name: Installing NumPy python library using pip module

    pip:

      name: NumPy

## Installing Using a Requirement File

Another way to install the libraries is via the requirements file. If you have any requirements file with all the libraries in the remote servers, give it as input to the "requirements" parameters.

Also, you can use the copy module beforehand to copy the requirements file to every remote server. In the following code, install the requirements file in the location/tmp/req.txt.

- hosts: all

  tasks:

  - name: Installing python libraries using requirements file

    pip:

      requirements: req.txt

      chdir: /tmp

req.txt

-------

nltk==3.0.0

numpy<2.0.0 scipy>=1.0.0

## Installing Multiple Python Libraries

To install the multiple packages, set all the libraries against the "name" parameter, separated by a comma.

- hosts: all

  tasks:

  - name: Installing multiple python packages

    pip:

      name: NumPy,SciPy

AD

## Installing a Particular Version of Pip Library

There is a "version" parameter, which can be used to install only the mentioned version of a library. In the following code, install the version of nltk library.

- hosts: all

  tasks:

  - name: Installing a required version of python library

    pip:

      name: nltk

      version: '3.0.0'

## Reinstall a Python Library

You can reinstall the python library by using the "forcereinstall" value for the "state" parameter.

This will reinstall the latest version of the library. You can use the "version" parameter along with it. The following code will install the version 3.0.0 of the nltk library.

- hosts: all

  tasks:

  - name: Reinstalling a python library

    pip:

      name: nltk

      version: 3.0.0

      state: forcereinstall

## Removing a Python Library

You can delete a python library by changing the state to "absent". In the following code, we will remove the "NumPy" and "SciPy" python libraries from the remote servers.

- hosts: all

  tasks:

  - name: Removing Python libraries

    pip:

      name: NumPy,SciPy

      state: absent

Ansible vs Chef

Ansible and Chef both are the most popular configuration management tools. Both tools can accomplish many of the same tasks, they each have different strengths, and they perform their tasks in different ways.

This tutorial will explore the strengths and differences of these tools. Before move further, take a glance at Ansible and Chef.

Ansible

Ansible is an open-source IT engine that automates application deployment, cloud provisioning, intra service orchestration, and other IT tools.

Ansible is easy to deploy because it does not use any **agents** or **custom security** infrastructure on the client-side, and by pushing modules to the clients. These modules are executed locally on the client-side, and the output is pushed back to the Ansible server.

It can easily connect to clients using **SSH-Keys**, simplifying though the whole process. Client details, such as **hostnames** or **IP addresses** and **SSH port**s, are stored in the files, which are called inventory files. If you created an inventory file and populated it, then Ansible can use it.

Ansible uses the playbook to describe automation jobs, and playbook, which uses simple language, i.e., **YAML**. YAML is a human-readable data serialization language & commonly used for configuration files, but it can be used in many applications where data is being stored.

A significant advantage is that even the IT infrastructure support guys can read and understand the playbook and debug if needed.

Ansible is designed for multi-tier deployment. Ansible does not manage one system at a time, and it models IT infrastructure by describing all of your systems are interrelated. Ansible is entirely agentless, which means Ansible works by connecting your nodes through **SSH** (by default). Ansible gives the option to you if you want another method for the connection like **Kerberos**.

Chef

The chef is a powerful automation platform that transforms infrastructure into the code. Whether you are operating in the on-premises, cloud, or a hybrid environment.

Chef automates how the infrastructure is **deployed, configured,** and **managed** across your network. A chef is an open-source cloud configuration that translates system administration tasks into reusable definitions, otherwise known as recipes and cookbooks.

Chef runs on different platforms such as Windows, AIX, Enterprise Linux distributions, Solaris, FreeBSD, Cisco IO, and Nexus.

It also supports cloud platforms such as Amazon Web Services (AWS), Google Cloud Platform, OpenStack, IBM Bluemix, HPE Cloud, Microsoft Azure, VMware vRealize Automation, and Rackspace.

**Below are some main differences between the Ansible and Chef:**

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Ansible** | **Chef** |
| **Availability** | Ansible runs with a single active node, called the Primary instance. If the primary goes down, there is a Secondary instance to take its place. | When there is a failure on the primary server, which is a chef server, it has a backup server to take the place of the primary server. |
| **Easy to setup** | Ansible has only a master running on the server machine, but no agents running on the client machine. It uses an SSH connection to log in to client systems or the nodes you want to configure. Client machine VM requires no unique setup. That's why it is faster to setup! | Chef has a master-agent architecture. Chef server runs on the master machine, and Chef client runs as an agent on each client machine. And also, there is an extra component called workstation, which contains all the tested configurations and then pushed to the central chef server. That's why it is not that easy. |
| **Management** | Easy to manage the configurations as it uses YAML (Yet Another Markup Language). The server pushes configurations to all the nodes. Suitable for real-time application, and there is immediate remote execution. | You need to be a programmer to manage the configurations as it offers configurations in Ruby DSL. The client pulls the configurations from the Server. |
| **Configuration language** | Ansible uses YAML (Python). It is quite easy to learn and its administrator oriented. Python is inbuilt into most Unix and Linux deployments, so setting the tool up and running is quicker. | Chef uses Ruby Domain Specific Language (Ruby DSL). It has a Steep Learning Curve and its developer-oriented. |
| **Interoperability** | The Ansible server has to be on Linux/Unix machine. As well as Ansible supports windows machines. | Chef Server works only on Linux/Unix, but Chef Client and Workstation can be on windows as well. |
| **Pricing** | The pricing for Ansible Tower for standard IT operations up to 100 nodes is $10,000 per year. This includes 8\*5 support, whereas premium offers 24\*7 support for $14000 per year. | Chef Automate gives you everything you need to build, deploy in $137 node per year. |
| **Authoritative configuration** | Ansible's authoritative configuration comes from its deployed playbooks, which are perfect as source control systems. Or the Ansible method is more accessible and makes more sense. | The chef relies on its server as the authoritative configuration, and those servers require uploaded cookbooks, which means making sure the latter are consistent and identical. |

# Ansible vs Puppet

Ansible and Puppet are fast becoming essential components for managing a large number of servers. They are commonly called configuration management and remote execution tools.

These mega-useful apps allow the admin, for instance, to execute an action on several servers simultaneously, and deploy multiple apps with a single click. It makes it much easier to configure and maintain thousands of servers. Before move further, take a glance at Ansible and puppet.

## Ansible

Ansible is an open-source IT engine that automates application deployment, cloud provisioning, intra service orchestration, and other IT tools.

Ansible is easy to deploy because it does not use any **agents** or **custom security** infrastructure on the client-side, and by pushing modules to the clients. These modules are executed locally on the client-side, and the output is pushed back to the Ansible server.

It can easily connect to clients using **SSH-Keys**, simplifying though the whole process. Client details, such as **hostnames** or **IP addresses** and **SSH ports**, are stored in the files, which are called inventory files. If you created an inventory file and populated it, then Ansible can use it.

Ansible uses the playbook to describe automation jobs, and playbook, which uses simple language, i.e., **YAML**. YAML is a human-readable data serialization language & commonly used for configuration files, but it can be used in many applications where data is being stored.

A significant advantage is that even the IT infrastructure support guys can read and understand the playbook and debug if needed.

Ansible is designed for multi-tier deployment. Ansible does not manage one system at a time, and it models IT infrastructure by describing all of your systems are interrelated. Ansible is entirely agentless, which means Ansible works by connecting your nodes through **SSH** (by default). Ansible gives the option to you if you want another method for the connection like **Kerberos**.

## Puppet

Puppet is a configuration management tool that is used for configuring, deploying, and managing servers. It also performs the following functions, such as:

* It defines the distinct configuration for every host, and also it continuously checks and confirms the required configuration, such as is it in place and not altered on the host. (if altered, they will revert to the required configuration).
* Dynamic scaling up and scaling down of machines.
* It provides control over all your configured machines, so a centralized change gets propagated to all automatically.

Puppet uses a master-slave architecture in which the master and the slave communicate through a secure encrypted channel with the help of SSL.

**Below are some main differences between the Ansible and Puppet:**

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Ansible** | **Puppet** |
| Availability | Ansible runs with a single active node, called the Primary instance. If the primary goes down, there is a Secondary instance to take its place. | Puppet has multi-master architecture. If the active master goes down, then the other master takes the active master place. |  |
| Easy to setup | Ansible has only a master running on the server machine, but no agents running on the client machine. It uses an SSH connection to log in to client systems or the nodes you want to configure. Client machine VM requires no unique setup. That's why it is faster to setup! | Puppet also has a master-agent architecture. Puppet server runs on the master machine, and Puppet clients run as an agent on the client machine. After that, there is a certificate signing between the agent and the master. That's why it is not that easy to setup. |  |
| Management | Easy to manage the configurations as it uses YAML (Yet Another Markup Language). The server pushes configurations to all the nodes. Suitable for real-time application, and there is immediate remote execution. | Puppet is not easy to manage the configurations as it uses its language called Puppet DSL. The client pulls the configurations from the Server. It is entirely system-administrator oriented, and there is non-immediate remote execution. |  |
| Configuration language | Ansible uses YAML (Python). It is quite easy to learn, and it is administrator oriented. Python is inbuilt into most Unix and Linux deployments, so setting the tool up and running is quicker. | Puppet uses its puppet Domain Specific Language (Puppet DSL). It is not easy to learn, and it is system administrator oriented. |  |
| Interoperability | The Ansible server has to be on Linux/Unix machine. As well as Ansible supports windows machines. | Puppet Master works only on Linux/Unix, but Puppet Agent also works on windows. |  |
| Pricing | The pricing for Ansible Tower for standard IT operations up to 100 nodes is $10,000 per year. This includes 8\*5 support, whereas premium offers 24\*7 support for $14000 per year. | The pricing for puppet ranges from $112 node per year with a standard support plan to $199 node per year with the premium plan. |  |
| GUI | Ansible was the command-line tool only at the time of its inception. Now it has the UI in the enterprise version, but it is not perfect. Sometimes, GUI is not in perfect sync with the Command line and not able to perform the same things like the command-line interface. | Puppet's Graphical User Interface is more interactive than Ansible. It is used to manage, view, and monitor more complex tasks. Otherwise, there is an option of using a command-line interface too when need which is written in Ruby. |  |

# Ansible Interview Questions



A list of frequently asked **Ansible Interview Questions** and Answers are given below.

### 1) What is Ansible?

Ansible is an open-source IT engine that automates application deployment, cloud provisioning, intra service orchestration, and other IT tools.

Ansible is very easy to deploy because it does not use any **agents** or **custom security** infrastructure on the client-side, and by pushing modules to the clients. These modules are executed locally on the client-side, and the output is pushed back to the Ansible server. It can easily connect to clients using **SSH-Keys**.

### 2) What are the uses of Ansible?

* Ansible is free to use by everyone.
* Ansible is very consistent and lightweight, and no constraints regarding the operating system or underlying hardware are present.
* It is very secure due to its agentless capabilities and open **SSH** security features.
* Ansible does not need any special system administrator skills to install and use it.
* Ansible has a smooth learning curve determined by the comprehensive documentation and easy to learn structure and configuration.
* Its modularity regarding **plugins, inventories, modules,** and **playbooks** make Ansible perfect companion orchestrate large environments.

### 3) What is Ansible Tower?

Ansible Tower is like Ansible at a more enterprise level. It is the web-based solution for managing your organization with an easy user interface that provides a dashboard with all of the state summaries of all the hosts. And allows quick deployments, and monitors all configurations.

### 4) What is Ansible Galaxy?

Ansible Galaxy is a galaxy website where users can share roles and to a command-line tool for **installing, creating,** and **managing** roles.

Ansible Galaxy gives greater visibility to one of Ansible's most exciting features, such as application installation or reusable roles for server configuration. Lots of people share roles in the Ansible Galaxy.

### 5) What is Ansible Modules?

Ansible modules are discrete units of code which can be used from the command line or in a playbook task.

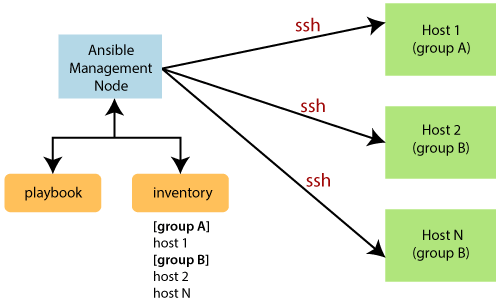
The modules also referred to as task plugins or library plugins in the Ansible.

Ansible ships with several modules that are called **module library**, which can be executed directly or remote hosts through the playbook.

Users can also write their modules. These modules can control like **services, system resources, files,** or **packages,** etc. and handle executing system commands.

### 6) How does Ansible work?

Ansible works by connecting to the nodes and pushing out a small program called **Ansible modules** to them. Then Ansible executed these modules and removed them after finished. The library of the modules can reside on any machine, and there are no **daemons, servers,** or **databases** required.



The **Management Node** is the controlling node that controls the entire execution of the playbook. The **inventory** files provide the list of hosts where the Ansible modules need to be run. The **Management Node** makes an **SSH** connection and executes the small modules on the **host's** machine and install the software.

### 7) What are the Ansible Server requirements?

* If you are a windows user, then you required to have a virtual machine in which Linux should be installed.
* And it requires python 2.6 version or higher.

AD

### 8) What are the variables in Ansible?

The variable is very similar to using the variables in a programming language. It helps you to assign a value to a variable and use it anywhere in the playbook. You can put the conditions around the value of the variables and use them in the playbook accordingly.

### 9) What is Ansible Task?

Ansible Task allows us to break up bits of configuration policy into smaller files. These are the block of code which can be used to automate any process.

### 10) Explain the basic terminologies or concepts in Ansible?

Some basic terms which are commonly used in Ansible, such as:

* **Controller Machine:** The Controller machine is used to provisioning the servers, which is managed. This is the machine where Ansible is installed.
* **Inventory:** An inventory is an initialization file which has details about the different servers you are managing.
* **Playbook:** It is a code file that is written in the YAML format. A playbook contains the tasks that need to be automated or executed.
* **Task:** Every task represents a single procedure that needs to be executed, e.g., installs a library.
* **Module:** A module is the set of tasks that can be executed. Ansible has 100s of built-in modules, and also you can create custom ones.
* **Role:** The role is a pre-defined way for organizing playbooks and other files to facilitate sharing and reusing portions of provisioning.
* **Play:** The task executed from start to finish, or the execution of a playbook is called the play.
* **Facts:** Facts are global variables which are store details about the system, such as network interfaces or operating system.
* **Handlers:** Handlers are used to trigger the status of a service, such as restarting or stopping a service.

### 11) What are Ad-hoc commands?

Ad-hoc commands are simple one-line commands which are used to perform a certain task. You can think of Ad-hoc commands as an alternative to writing playbooks. An example of an Ad-hoc command, as shown as follows:

1. ansible host -m netscaler -a "nsc\_host=nsc.example.com user=ansiuser password=ansipass"

The above Ad-hoc command accesses the NetScaler module to disable the server.

### 12) How would you access a variable of the first host in a group?

Executes the following commands to access a variable of the first host in a group:

1. {{ hostvars[groups['webservers'][0]]['ansible\_eth0']['ipv4']['address'] }}

This command is accessing the hostname of the first machine in the webservers group. If you are doing this by a template, then use the jinja2 '#set', or you can use set\_fact.

### 13) How to keep secrete data in a playbook?

The following command is used to keep verbose output but hide the sensitive information from others who would like to be able to see the output.

1. -   name: secrete task
2. shell: /user/bin/do\_somthing -value={{secrete value}}
3. no\_log: True

Also, the no\_log attribute can apply to an entire play.

1. -   hosts: all
2. no\_log: True

### 14) What are Ansible Vaults, and why are they used?

Ansible Vault is a feature that allows keeping all your secrets safe. It can encrypt entire files, YAML playbooks, or even a few variables. It provides a facility where you also can integrate the sensitive data into your playbooks.

AD

Vault is implemented with file-level granularity, where the files are completely encrypted or unencrypted. It uses the same password for encrypting as well as decrypt the data, which makes Ansible vault user-friendly.

### 15) What is the way to access shell environment variables in Ansible?

In Ansible, if you want to access existing variables, then you need to use the '**env**' lookup plugin.

**For example,** you want to access the value of the Office environment on the management machine, as shown in the following command, such as:

1. ---
2. # ...
3. vars:
4. local\_home: "{{ lookup('env','Office') }}"
5. I
6. {{ ansible\_env.SOME\_VARIABLE }}

### 16) Who you can copy file recursively onto a target host?

The "copy" module has a recursive parameter. Therefore, if you want to perform more efficient for a large number of files, then the "synchronize" module is the best option for doing this task.

### 17) How do you set the path or any other environment variable for a task?

The environment variables can be set by using the 'environment' keyword. It can be set for either a task or an entire playbook. Follow the following code to see how do set the path, such as:

1. environment:
2. PATH: "{{ansible\_env.PATH}}:/thingy/bin"
3. SOME: value

### 18) How can you see all variables for a host?

You can see all the variables using the **host vary** variable. It stores host variables with the hostname as key. For example, to look at the variables defined for localhost, you can run the following command, such as:

1. ansible -m debug -a "var=hostvars[inventory\_hostname]"

### 19) What is the difference between the Variable Name and Environment Variable in Ansible?

|  |  |
| --- | --- |
| **Variable Name** | **Environment Variable** |
| You need to add strings to create variable names. | You need existing variables to access environment variables. |
| You can easily create multiple variable names by adding strings. | To create environment variables, you must refer to the advanced Ansible playbook. |
| Use the IPV4 address for variable names. | Use {{ansible\_env.SOME\_VARIABLE}} for remote environment variables. |

### 20) What is RedHat Ansible?

Ansible and Ansible Tower by Red Hat, both are the end to end complete automation platforms which are capable of providing the following features or functionalities:

1. Provisioning
2. Deploying applications
3. Orchestrating workflows
4. Manage IT systems
5. Configuration of IT systems
6. Networks
7. Applications

All of these activities are dealing with Ansible, where it can help the business to solve real-time business problems.

Kubernetes Tutorial



What is Kubernetes?

**Kubernetes** is also known as **'k8s'.** This word comes from the Greek language, which means a **pilot** or **helmsman**.  
**Kubernetes** is an extensible, portable, and open-source platform designed by **Google** in **2014**. It is mainly used to automate the deployment, scaling, and operations of the container-based applications across the cluster of nodes. It is also designed for managing the services of containerized apps using different methods which provide the scalability, predictability, and high availability.

It is actually an enhanced version of '**Borg**' for managing the long-running processes and batch jobs. Nowadays, many cloud services offer a Kubernetes-based infrastructure on which it can be deployed as the platform-providing service. This technique or concept works with many container tools, like **docker,** and follows the client-server architecture.

Key Objects of Kubernetes

Following are the key objects which exist in the Kubernetes:

**Pod**

It is the smallest and simplest basic unit of the Kubernetes application. This object indicates the processes which are running in the cluster.

**Node**

A **node** is nothing but a single host, which is used to run the virtual or physical machines. A node in the Kubernetes cluster is also known as a minion.

**Service**

A **service** in a Kubernetes is a logical set of pods, which works together. With the help of services, users can easily manage load balancing configurations.

**ReplicaSet**

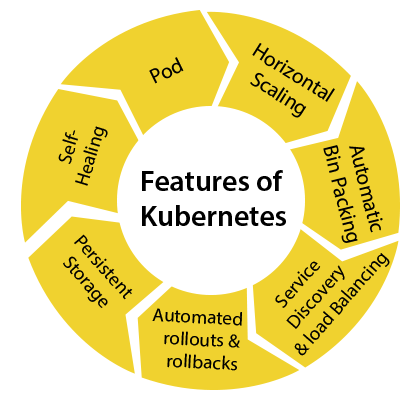
A **ReplicaSet** in the Kubernetes is used to identify the particular number of pod replicas are running at a given time. It replaces the replication controller because it is more powerful and allows a user to use the "set-based" label selector.

**Namespace**

**Kubernetes** supports various virtual clusters, which are known as namespaces. It is a way of dividing the cluster resources between two or more users.

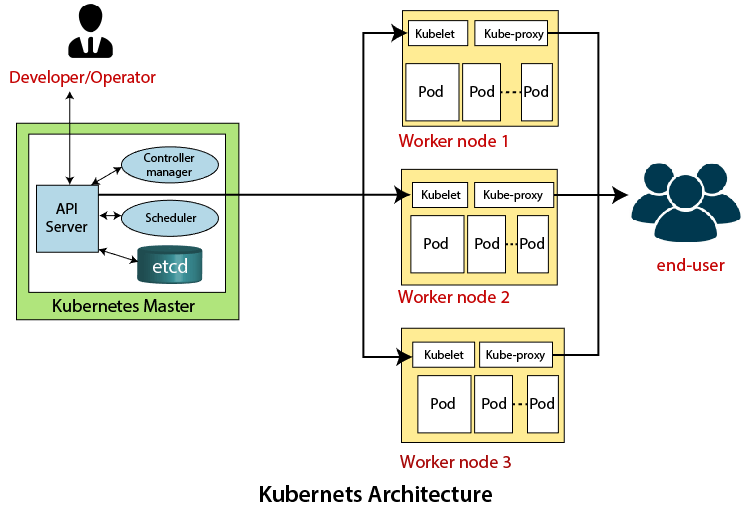
Features of Kubernetes

Following are the essential features of Kubernetes:



1. **Pod:** It is a deployment unit in Kubernetes with a single Internet protocol address.
2. **Horizontal Scaling:** It is an important feature in the Kubernetes. This feature uses a **HorizontalPodAutoscalar** to automatically increase or decrease the number of pods in a deployment, replication controller, replica set, or stateful set on the basis of observed CPU utilization.
3. **Automatic Bin Packing:** Kubernetes helps the user to declare the maximum and minimum resources of computers for their containers.
4. **Service Discovery and load balancing:** Kubernetes assigns the IP addresses and a Name of DNS for a set of containers, and also balances the load across them.
5. **Automated rollouts and rollbacks:** Using the rollouts, Kubernetes distributes the changes and updates to an application or its configuration. If any problem occurs in the system, then this technique rollbacks those changes for you immediately.
6. **Persistent Storage:** Kubernetes provides an essential feature called '**persistent storage'** for storing the data, which cannot be lost after the pod is killed or rescheduled. Kubernetes supports various storage systems for storing the data, such as **Google Compute Engine's Persistent Disks (GCE PD) or Amazon Elastic Block Storage (EBS).** It also provides the distributed file systems: **NFS or GFS**.
7. **Self-Healing:** This feature plays an important role in the concept of Kubernetes. Those containers which are failed during the execution process, Kubernetes restarts them automatically. And, those containers which do not reply to the user-defined health check, it stops them from working automatically.

Kubernetes Architecture



The architecture of Kubernetes actually follows the client-server architecture. It consists of the following two main components:

1. Master Node (Control Plane)
2. Slave/worker node

Master Node or Kubernetes Control Plane

The master node in a Kubernetes architecture is used to manage the states of a cluster. It is actually an entry point for all types of administrative tasks. In the Kubernetes cluster, more than one master node is present for checking the fault tolerance.

Following are the four different components which exist in the Master node or Kubernetes Control plane:

1. API Server
2. Scheduler
3. Controller Manager
4. ETCD

**API Server**

The Kubernetes API server receives the REST commands which are sent by the user. After receiving, it validates the REST requests, process, and then executes them. After the execution of REST commands, the resulting state of a cluster is saved in '**etcd**' as a distributed key-value store.

**Scheduler**

The scheduler in a master node schedules the tasks to the worker nodes. And, for every worker node, it is used to store the resource usage information.  
In other words, it is a process that is responsible for assigning pods to the available worker nodes.

**Controller Manager**

AD

The Controller manager is also known as a controller. It is a daemon that executes in the non-terminating control loops. The controllers in a master node perform a task and manage the state of the cluster. In the Kubernetes, the controller manager executes the various types of controllers for handling the nodes, endpoints, etc.

**ETCD**

It is an open-source, simple, distributed key-value storage which is used to store the cluster data. It is a part of a master node which is written in a GO programming language.

Now, we have learned about the functioning and components of a master node; let's see what is the function of a slave/worker node and what are its components.

Worker/Slave node

The Worker node in a Kubernetes is also known as minions. A worker node is a physical machine that executes the applications using pods. It contains all the essential services which allow a user to assign the resources to the scheduled containers.

Following are the different components which are presents in the Worker or slave node:

**Kubelet**

This component is an agent service that executes on each worker node in a cluster. It ensures that the pods and their containers are running smoothly. Every **kubelet** in each worker node communicates with the master node. It also starts, stops, and maintains the containers which are organized into pods directly by the master node.

**Kube-proxy**

It is a proxy service of Kubernetes, which is executed simply on each worker node in the cluster. The main aim of this component is request forwarding. Each node interacts with the Kubernetes services through **Kube-proxy**.

**Pods**

A **pod** is a combination of one or more containers which logically execute together on nodes. One worker node can easily execute multiple pods.

AD

Installation of Kubernetes on Linux

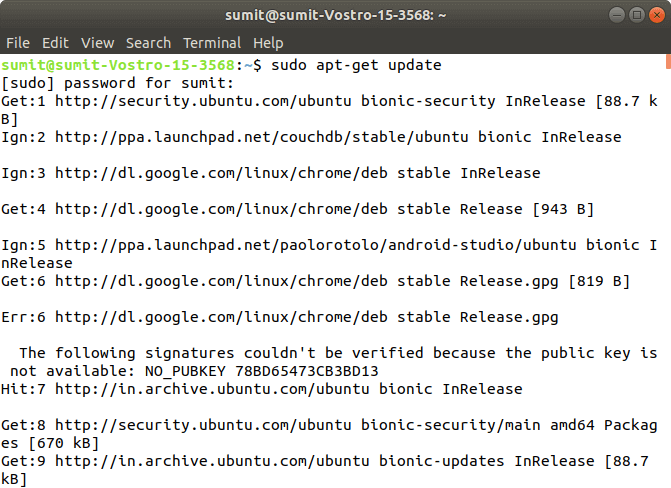
The installation of Kubernetes on Linux is a straight forward process. Follow the below steps to install the Kubernetes. In the installation of Kubernetes, each step is mandatory.

**Step 1:** In this step, we have to update the necessary dependencies of a system using two commands.

The first command is used to get all the updates. Execute the following command in the terminal; it will ask to enter the system's password.

1. sudo apt-get update

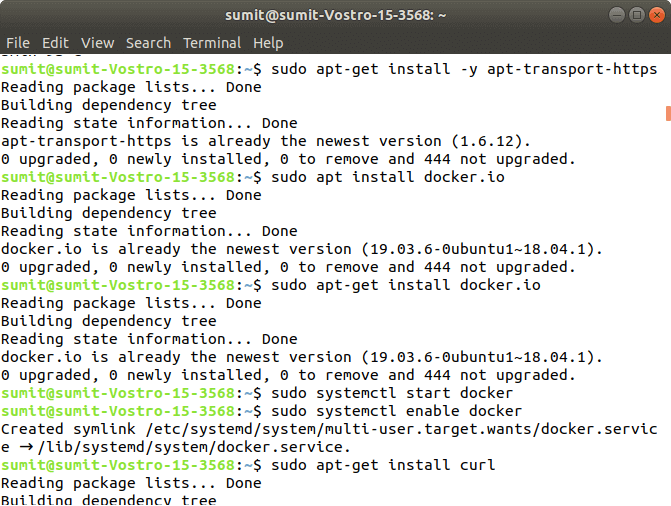
**Output:**



When the first command is successfully executed, type the following second command, which is used to make the repositories.

1. sudo apt-get install -y apt-transport-https

**Output:**

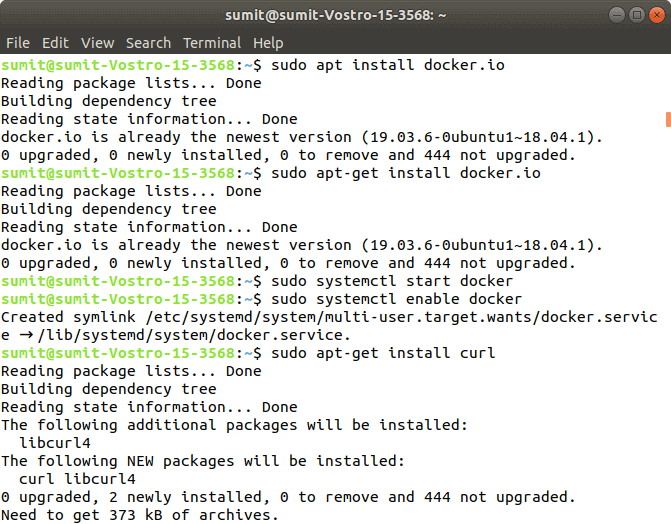


**Step 2:** After the above steps are successfully executed, we have to install the dependencies of docker in this step.

Type the following command to install the docker. In the installation process, we have to choose Y for confirmation of the installation.

1. sudo apt install docker.io

**Output:**



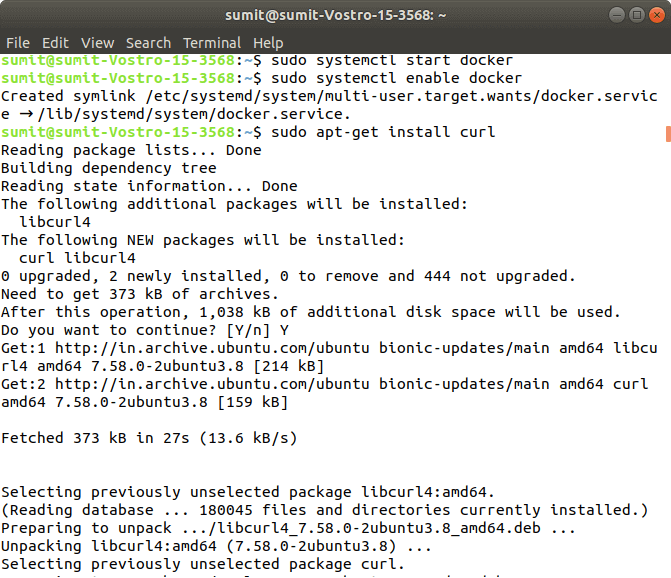
After installing the docker, we have to type the different two commands for starting and enabling the docker. Type the following first command, which starts the docker:

1. sudo systemctl start docker

Now, type the following second command, which enables the docker:

1. sudo systemctl enable docker

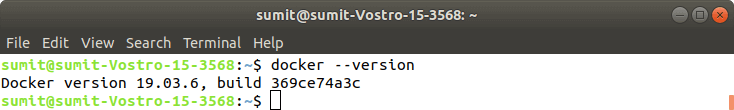
**Output:**



Now, we can check the version of docker by typing the following command:

1. Docker -version

**Output:**

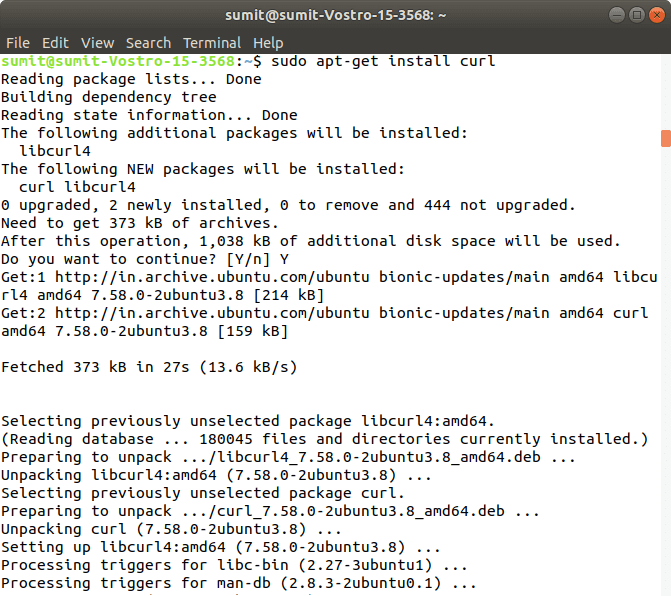


**Step 3:** After the successful execution of all the commands of the second step, we have to install the curl command. The curl is used to send the data using URL syntax.

Now, install the curl by using the following command. In the installation, we have to type Y.

1. sudo apt-get install curl

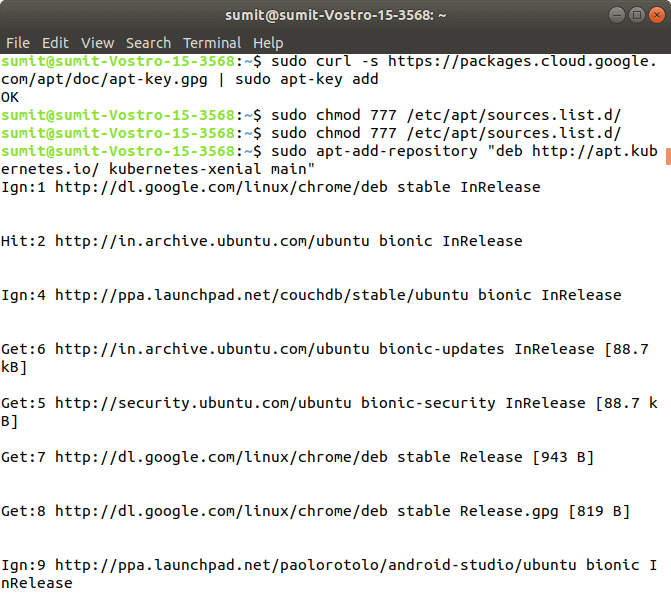
**Output:**



Now, we have to download the add package key for Kubernetes by the following command:

1. sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add

**Output:**

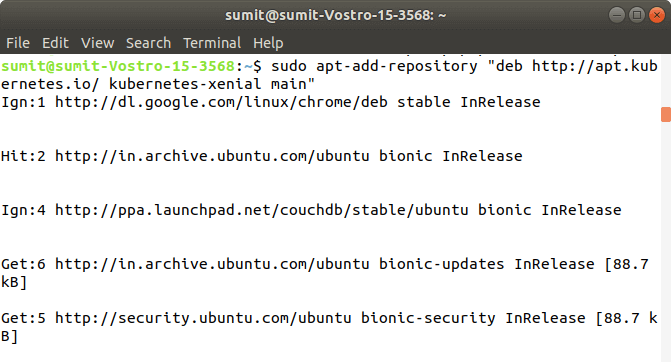


If you get an error from the above command, then it means your curl command is not successfully installed, so first install the curl command, and again run the above command.

Now, we have to add the Kubernetes repositories by the following command:

1. sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"

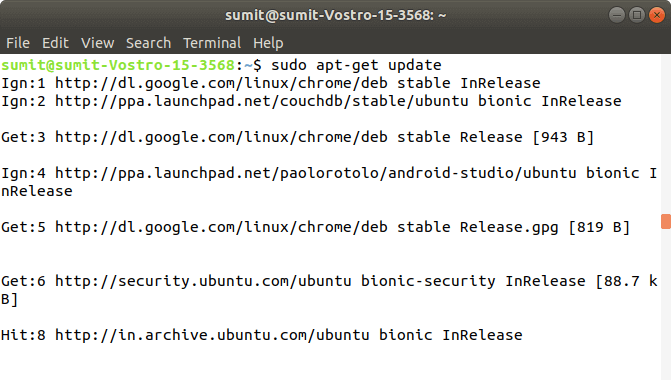
**Output:**



After the successful execution of the above command, we have to check any updates by executing the following command:

1. sudo apt-get update

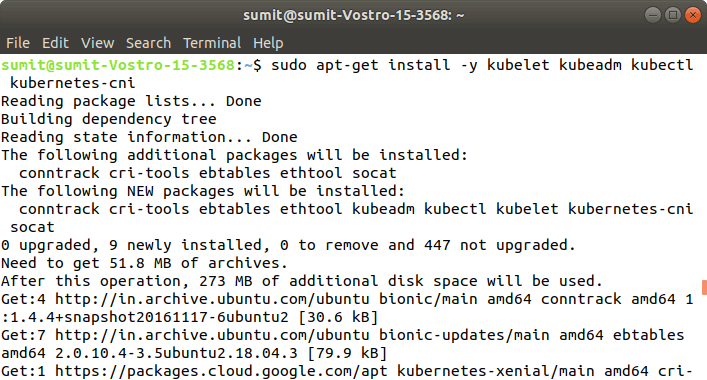
**Output:**



**Step 4:** After the execution of the above commands in the above steps, we have to install the components of Kubernetes by executing the following command:

1. sudo apt-get install -y kubelet kubeadm kubectl kubernetes-cni

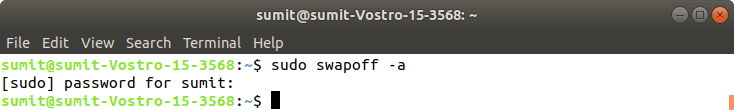
**Output:**



**Step 5:** After the above installation is done, we have to initialize the kubeadm by executing the following command. The following command disables the swapping on other devices:

1. sudo swapoff -a

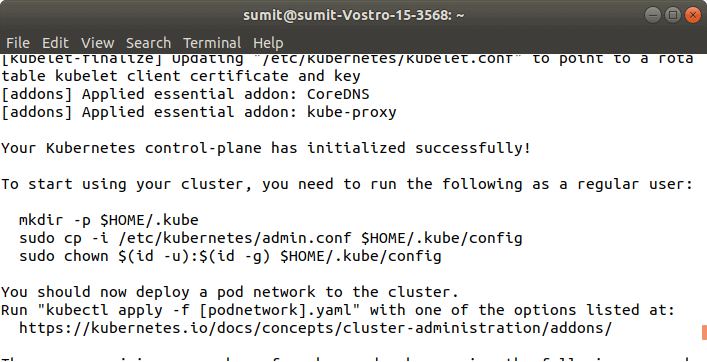
**Output:**



Now, we have to initialize the kubeadm by executing the following command:

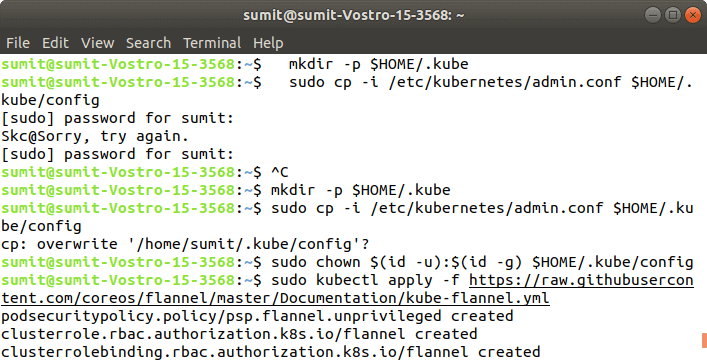
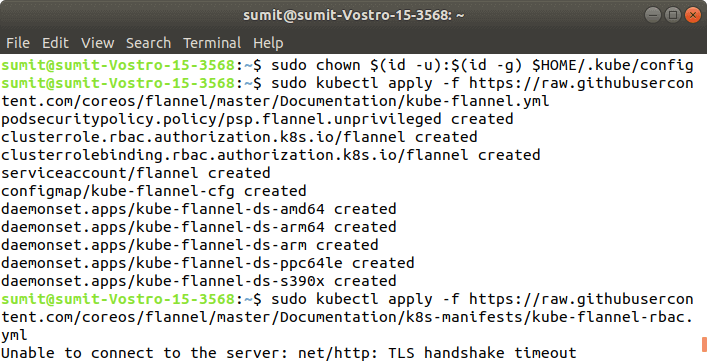
1. sudo kubeadm init

**Output:**



**Step 6:** After the above command is successfully executed, we have to run the following commands, which are given in the initialization of kubeadm. These commands are shown in the above screenshot. The following commands are used to start a cluster:

1. mkdir -p $HOME/.kube
2. sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
3. sudo chown $(id -u):$(id -g) $HOME/.kube/config

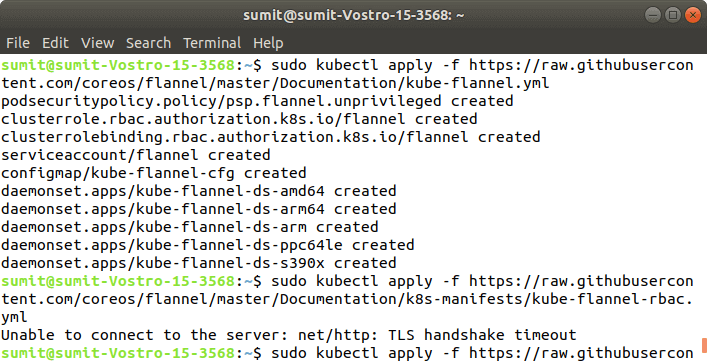
  


**Output:**

**Step 7:** In this step, we have to deploy the paths using the following command:

1. sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

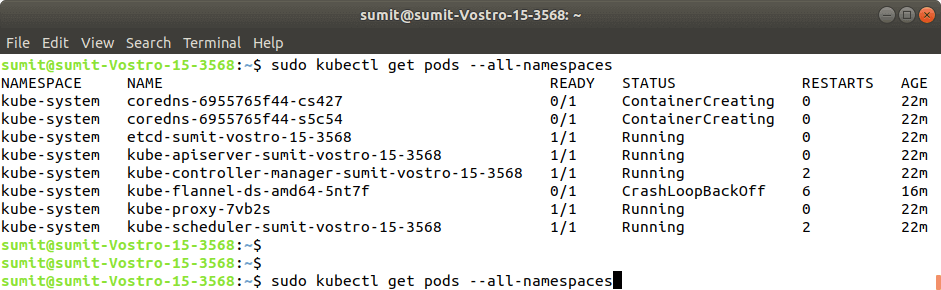
**Output:**



**Step 8:** After the execution of the above command, we have to run the following command to verify the installation:

1. sudo kubectl get pods --all-namespaces

**Output:**



If the output is displayed as shown in the above screenshot. It means that the Kubernetes is successfully installed on our system.