

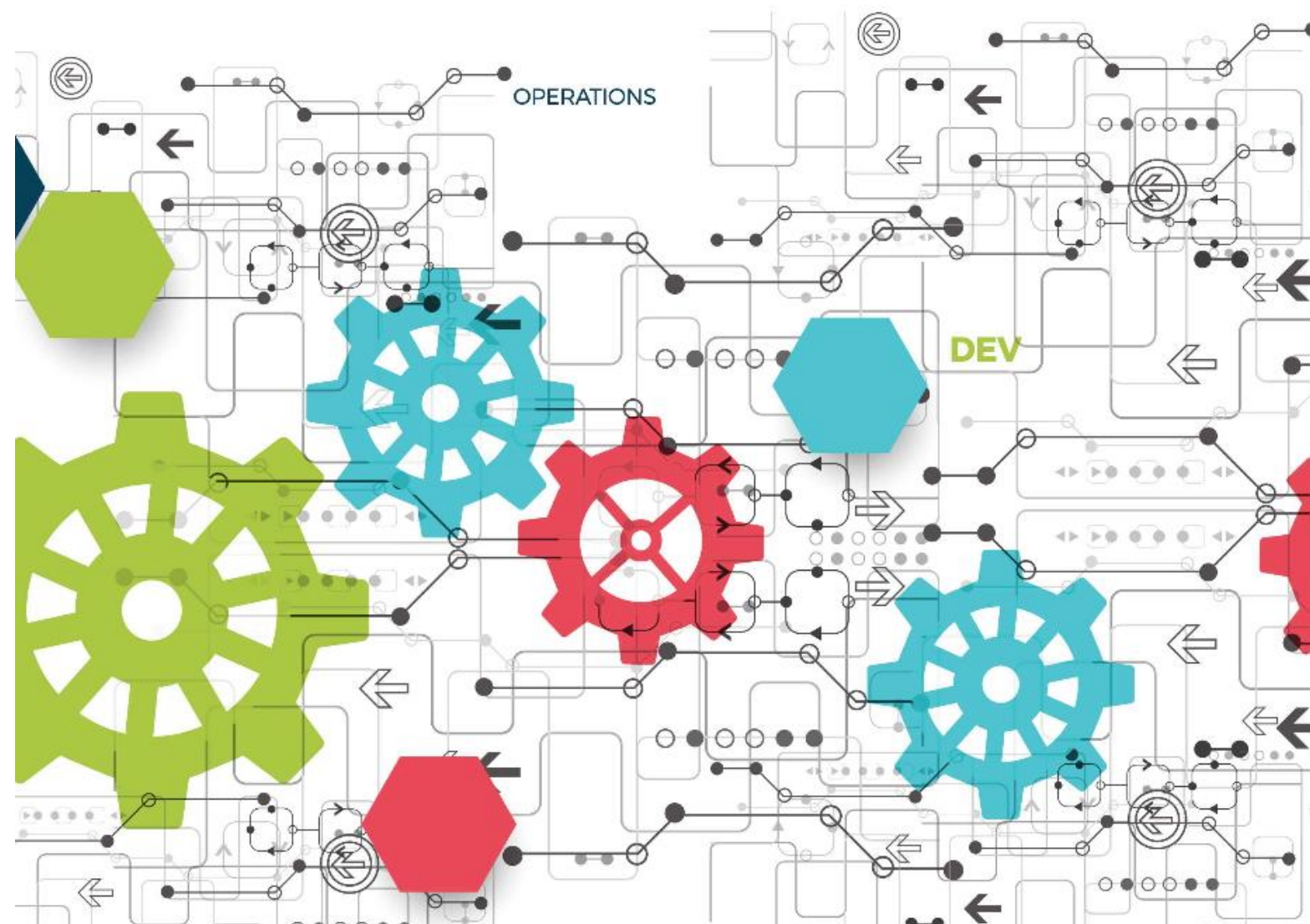


**B.Tech** Computer Science  
and Engineering in DevOps

# System Provisioning and Configuration Management

MODULE 1

## Introduction to Provisioning



# Contents

Module Objectives	1
Module Topics	2
1.1 What is Provisioning – Basic Definition	2
1.1.1 What is Provisioning – Software Definition	3
1.2 Concepts of Provisioning	3
What did You Grasp?	5
1.3 Why Provisioning Should be Exclusive ?	6
1.4 Configuration Management	7
1.4.1 Benefits of Configuration Management	8
1.4.2 Configuration Management Tools	8
1.4.2.1 Configuration Management — Ansible	9
1.4.2.2 Configuration Management — Chef	9
1.4.2.3 Configuration Management — Puppet	10
1.5 Why Provisioning is not Configuration Management?	10
1.6 Why we Need Provisioning Tools ?	11
What did You Grasp?	12
1.7 Test Machines for Provisioning	13
1.7.1 What is Kubernetes?	13
1.7.2 Test Machines — Kubernetes	14
1.7.3 Introduction to Helm	15
1.7.4 Components in Helm	15
1.7.5 Test Machines — Installation and Configuration of Helm Package Manager	16
1.7.6 Test Machines — Terraform	17
1.7.7 Test Machines – Terraform(Sample Example for Infrastructure as Code)	18
1.7.8 Approaches of Terraform	19
1.8 What is Deployment ?	19
1.8.1 What does Deployment Include?	20
1.8.2 Deployment Concepts	21
1.8.3 Deployment Terms	22
1.8.4 Relationship Between Deployment and Provisioning	23
In a nutshell, we learnt:	24

The background of the top section features a complex network of grey lines and arrows representing a DevOps cycle. Labels like 'DEVELOPMENT', 'OPS', and 'OPERATIONS' are scattered across the top. Large, interlocking gears in red, blue, and green are positioned around the central text. The text 'MODULE 1' is in a bold, black, sans-serif font.

## MODULE 1

# Introduction to Provisioning

### Facilitator Notes:

Welcome the participants and give them an overview of the module. Tell them that they will learn about 'System Provisioning', generic definitions – both brief and elaborated and understanding Provisioning in DevOps Cycle.

You will learn about the 'System Provisioning', generic definitions – both brief and elaborated and understanding Provisioning in DevOps Cycle.

### Module Objectives

At the end of this module, you will be able to:

- Explain the term provisioning
- Discuss the significance of provisioning
- Discuss test machines
- Define provisioning for deployments



### Facilitator Notes:

Explain the module objectives to the participants.



## Module Topics

Let us take a quick look at the topics that we will cover in this module:

- What is Provisioning – Basic Definition
- What is Provisioning – Software Definition
- Concepts of Provisioning
- Why Provisioning Should be Exclusive
- Configuration Management
- Configuration Management Tools
- Why Provisioning is not Configuration Management
- Provisioning Tools
- Test Machines for Provisioning
- Deployment
- Relationship between Deployment and Provisioning



### Facilitator Notes:

Inform the participants about the topics that they will be learning in this module.

## 1.1 What is Provisioning – Basic Definition

Provisioning enables (DevOps – team) to create, modify, and/or delete infrastructure using Code or APIs. Provisioning helps us achieve three major objectives for any infrastructure, such as follows:



1. Create



2. Modify



3. Delete

Provisioning basically uses the following to do the above actions:



1. Code

(Source/Object/Compiled/Executable)



2. APIs

### Facilitator Notes:

Inform the participants about the basic definition of Provisioning.

When we talk about provisioning (Creation and Modification and Deletion), we first talk about how to do provisioning. How these basic objectives will be met. The answer is by using the **Tools**. So if we can summarize the definition of Provisioning to be even more accurate we can define as:

*“Provisioning tools enable (Software/DevOps – teams) to create, modify, and/or delete infrastructure using code and/or APIs”*

## 1.1.1 What is Provisioning – Software Definition

The Software Definition of Provisioning is

***“Provisioning is in essence about allocating the application with a license - the authorization to execute and more so can extend its execution for a defined period of time”.***

### Facilitator Notes:

Explain the participants by giving an example of steps taken to acquire license of any software.

Provisioning (Software) encloses a number of activities and they are mostly related to an application's/infrastructure's license.

You can relate to the software(s) you might have used or are still using it with respect to having a certain time limit on a certain license. Let's say for e.g., “An Anti-Virus Software Application” – sometimes gives you options of 1 year or may be 2 or may be a lifetime. It totally depends on the package you choose to procure and alongside it comes other privileges and higher security levels.

To brief down a few:

1. You get a Licensed Serial number
2. You might get an Activation key or code
3. Help to configure the same. Even a special handbook or a .pdf file or any sort of attachment for the same
4. Various forms of license agreements from both user and supplier' end

## 1.2 Concepts of Provisioning

The provisioning concepts are as follows:



**Facilitator Notes:**

Explain the participants all the above concepts one by one with an example.

1. **License:** It is kind of a legal agreement between the user and the supplier (software company) which allows the user to use the particular software under certain regulations.
2. **Serial Number:** Serial number could be vivacious in nature. Serial Number is purely upon the Software Company' think-tank to decide whether it should be numeric, or it should be alphanumeric, or it should have certain special characters. How many characters should it have. Should something change in the next iteration. This serial number represents the purchase history of the software product alongside its associated legal agreement and regulations how to use the same.
3. **Retail Serial Number:** A license has one authorization. And it grants permission to that single user to install the software in maybe 1 or 2 or 3 different systems(again this totally depends on the license agreement terms and conditions). In DevOps context, a Retail Serial Number == Serial Number, but it does need another round of verification just to see in case the Serial Number key operates as intended.
4. **Volume Serial Number:** This is purely you can say – usage of a particular software product on multiple machines. But, it has to have a multi-seat license. How many machines to be allowed is totally dependent on the license agreement terms and conditions.
5. **License Key:** This is a very sensitive piece of String defined literals which helps any user to unveil the usage of a Software Product. It unlocks the functionality of the software product as well as in some cases registers the product from trial version to enterprise version or edition.
6. **License Holder:** It could be anyone. Let's assume you buy a Security Testing Enterprise Edition of **BurpSuite** software then there will be a particular username and password which will be used. Let's say for example Ajay@university.com is the email id which is registered along with a highly secure password. So Ajay is the license holder.
7. **Serialization:** It is a process. And these are the process steps:
  - Enter a serial number
  - Validate the serial number
  - Wait for authentication
  - Launch the application post successful authentication

Once the application is fully functional after the above activities, then the application is serialized.

8. **Deserialization:** This protocol needs special permission. It is like you are instigating reversal of serialization, which mean either you are revoking all rights from the serial number entered prior or you have a new serial number and you want to make the previous serial number invalid.
9. **Activation:** Again, this is a verification process which makes sure that the serial number entered is in accordance to the terms and conditions set via the license agreement and

it gives an activation notification and also a time frame for the same and in some cases notification of renewal in case.

10. **Deactivation:** It is known as a process of removing activation of a software product. There could be many reasons. Let's say time is up for a free version and that will not operate now. Time limit is up for an enterprise edition and unless renewed it will remain in a deactivated state.
11. **EULA(End User License Agreement):** It is an agreement which has all the Do's and Don'ts and has all the terms and conditions and violation of the same could lead to whatever consequences.

## What did You Grasp?



1. State True or False  
Can we use Provisioning tools or Configuration tools to do provisioning?Git tagging  
A) True  
B) False

Facilitator Notes:


Answer: B. False



2. State True or False  
Ansible is a better option than Terraform to do Provisioning?  
A) True  
B) False

Facilitator Notes:

Answer: B. False



2. State True or False

Un-provisioned software can run equally good with a pirated licensed key across all Systems?


A) True  
B) False

**Facilitator Notes:**


**Answer:** B. False

### 1.3 Why Provisioning Should be Exclusive ?


Provisioning should be exclusive due to the following reasons:



1. Interaction with other teams



2. Interaction with business



3. Empowerment of business

**Facilitator Notes:**

Inform the participants how provisioning makes a business contract even more concrete.

Provisioning as a practice should be made exclusive and tools for Configuration Management should not be used for provisioning, otherwise it just makes the situation more complex and more painstaking.

Provisioning allows a whole lot of interaction with various teams and businesses and shows the business that this is what your business is powered by and this is how it runs and this is how it will run for certain duration of time with certain effectiveness.

All of this is only possible because of the Code or APIs used by Provisioning tools. Let's discuss how.

The focus on code and APIs is very important. When we talk about DevOps or the DevOps culture, one of the things we focus is about not throwing things over the wall, not siloing things. With code and APIs, this enables the breakdown of silos because we have a common language throughout which we can interact with other teams, interact with other businesses



and show the business that this is what your business is empowered by and this is how it runs. Imagine a business running without provisions, without setups and without licenses. Hence, provisioning should be exclusive.

## 1.4 Configuration Management

What is Configuration management (CM)?

***Configuration management (CM) is a systems engineering process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life.***

***Configuration Management is a progressively increasing, highly important platform for a Techno-Functional Organization***

### Facilitator Notes:

Discuss about the configuration management.

There could be few exceptions in real life to the statement “Bigger the better”. But managing the bigger becomes bit more complex and needs more groundbreaking and sophisticated adjustments. Same applies to DevOps Principle of a Simple Medium Sized firm transferring or may be called transforming into Cloud and expanding its horizons in due course of time

A micro particle experiment becomes more ground breaking when it switches to nano particle experimentation but somehow in the software industry as Legacy IT systems slowly get to become Modern we need to take certain ground rules into consideration. Some of them being automation, build, and maintainability of these IT Systems.

So what does Configuration Management have to do here ?

Answer: It is an automated method to maintain all these IT Systems in a consistent state of modernity.

An IT System can comprise of Servers, Storage, Networking and Software. What configuration management aims is at these building blocks of an IT Management System. It helps to maintain these systems in an operable, determined state of tune. Another proverbial aspect of configuration management is the description of the desired state which is wants to keep the IT System in. The most important aspect of a configuration management system is the process of Automation

## 1.4.1 Benefits of Configuration Management

Following are the benefits of configuration management:



1. IT System' consistency



2. Software consistency



3. Increased efficiency



4. Scalability

### Facilitator Notes:

Explain the benefits of configuration management.

- The most important benefit of configuration management is consistency of IT systems and the software involved. Configuration management ensures that a system is correctly configured. It is not like the scare which involves with manual configurations.
- Configuration management in combination with automation improves efficiency.
- Scalability is another benefit as configuration management helps to decide to scale up to what level as per usage and requirements.

## 1.4.2 Configuration Management Tools

Following are the popular tools of configuration management:



Ansible



Chef



Puppet

### Facilitator Notes:

Define the popular tools of configuration management.

Let's discuss the popular tools of configuration management:

- **Ansible:** Ansible is a powerful automation language. It is an open source. Ansible is a deployment and orchestration tool.
- **Chef:** Chef is also an open source tool. It uses Ruby language. It is used in infrastructure automation provisioning and it reduces repetitive work. It is also known as programmable infrastructure.

- **Puppet:** Puppet is the most mature and sophisticated configuration management tool today. It is the most widely used and popular tool too. It deploys, configures and manages servers.

### 1.4.2.1 Configuration Management — Ansible

How Ansible Works?



#### Facilitator Notes:

Define the popular tools of configuration management.

**Automate:** Ansible deploys the apps and web application. It manages the IT Systems. And it is a tool which helps in reduction of code complexity. Complexity is sometimes confused with degrading of Coding Script' standard. To make it sub-standard is not the case with Ansible. What Ansible basically does is — it makes sure the code is in compliance with the Architectural Design.

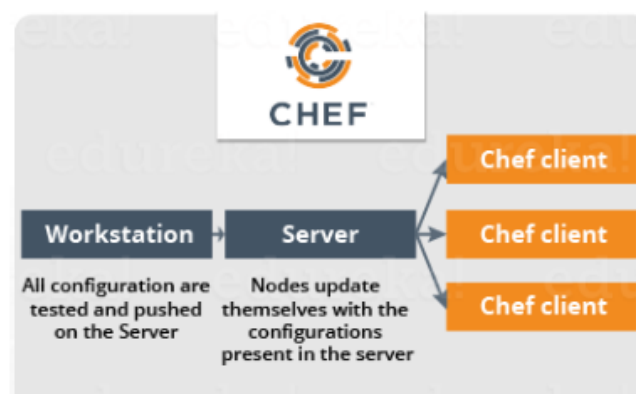
**Accelerate:** Then Ansible solves the problems. And once the problems are solved, it is shared with everyone.

**Collaborate:** Ansible breaks down silos and creates a culture of Automation.

**Integrate:** Ansible integrates the Automated Module with the software technologies already in use.

### 1.4.2.2 Configuration Management — Chef

How Chef Works?



#### Facilitator Notes:

Define the popular tools of configuration management.

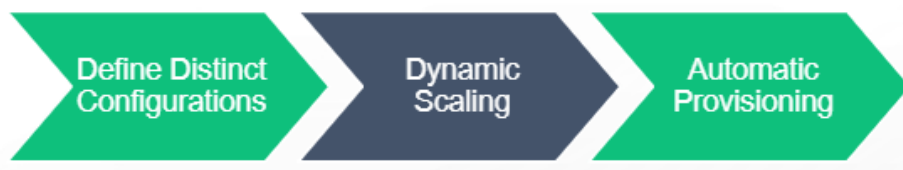
There are Nodes in Chef. They are dynamically updated with the configurations(Pull Configuration) in the Server. Because of Pull Configuration, we need not execute even a single command on the Chef server. Nodes will automatically configure and update themselves with the configurations present in the Server.

All configurations are tested and pushed on to the server [This is the role of WorkStation]

And in the Server, nodes configure and update themselves. And then these are forwarded to the various Chef Clients.

### 1.4.2.3 Configuration Management — Puppet

#### How Puppet works?



#### Facilitator Notes:

Define the popular tools of configuration management.

There would be many hosts. Puppet defines distinctly the configuration setup for each and every host. And it continuously checks and confirms the required configuration is almost to a state of exactment on the host. A reverse to the previous mode should be permissible.

The scaling levels can go up and down depending on various factors. And Puppet allows the same. Scale up and go is smooth via Puppet in all the machines.

Puppet provides control over all configured machine setups. A centralized server (master server) helps get the changes associated to all using Automation. It reduces manual intervention and therefore results in less human errors.

### 1.5 Why Provisioning is not Configuration Management?

Let's discuss why provisioning is not configuration management.

1. Increases complexity
2. Functionality may or may not work
3. Integration might fail at certain point
4. Needs extremely talented and skillful resource to carry out the impossible



**Facilitator Notes:**

Explain the participants why provisioning is not configuration management.

Let's take an example:

Suppose there is an Active Infrastructure which we need to deploy with 1 or 2 domains from scratch. And this has to be automated.

Imagine using **Ansible** to provision the servers, to provision the Active Infrastructure and to provision the Domains. This setup will work fine for some point of time. It will look great. But it will deviate from its intended function with introduction of more provisional setups and integration with other existing IT Machines. For example, challenge comes when:

1. You need to associate security groups with those instances (creating security groups with **Ansible** is easy, but associating them with stuff is very complicated)
2. Imagine you need to add more instances — like create a password reset server
3. Imagine adding more tags

So with every add on to make the business more concrete, indirectly we are making the system highly complex even to see the culmination of reality.

It will take great expertise, overtime work and scratching and crawling over everything and then maybe end up with something which may or may not work.

So use specific tools for provisioning and let configuration management tools only configure.

**Simply put Provisioning != Configuration Management**

## 1.6 Why we Need Provisioning Tools ?

Let's discuss why we need provisioning tools:

- Infrastructure Deployment Procedures as code — no more .docx or .pdf runbooks
- Saves time from writing own provisioning code
- Enables infrastructure deployment testing just like code — no fragile infrastructure

**Facilitator Notes:**

Explain the participants why we need Provisioning tools.

To explain why we need Provisioning tools more than ever, imagine a time when there always existed a guide book or a runbook with someone Authored by scene. Check for the latest version, check if that person still works in the same array of things and then see if that can

be implemented with minimal changes in the current scenario. Was not such an easy task.

And now:

There is no more run books — previously there were runbooks in .docx and .pdf and we could just hope they were all updated. But today it is all in code which is stored in git, bitbucket, in svn or any cloud or in our your SCM.

Code is straightforward and ready to read and understand. You can show the code to anybody in the organization and the likelihood of them understanding them is greater. This in return saves a lot of time. So we get to save time and we need not write thousands of lines of code to provision infrastructure with any existing tools that weren't exactly meant to do that. Since, it is all automated and accounted for.

So we can treat infrastructure like software and benefit is — the software is testable and ensure software does what it is intended to do the following:

- We can enable unit test
- We can enable integration tests
- We can enable regression tests
- Can make your infrastructure bullet proof and remove all the parameters which could make it fragile

This is why Provisioning tools are so important as they help us use simple straight forward code which can be changed, updated with minimal fuss and maximum result

## What did You Grasp?



1. Which of the following statements are correct for provisioning?  
A) Provisioning improve system security  
B) Provisioning makes room for better QA operations  
C) Provisioning makes our system more robust and concrete  
D) All of the above

### Facilitator Notes:

**Answer:** D. All of the above

## 1.7 Test Machines for Provisioning

Following are the test machines for provisioning:

- Kubernetes
- Helm Package manager on Kubernetes
- Terraform for Infrastructure as a code



### Facilitator Notes:

Describe the participants about test machines.

There are lot of open source systems (machines) for automation deployment, scaling and managing containerized applications. You might have some knowledge about Kubernetes and some of you might have got to work with it at a practical level. So why are we learning this. The reason being:

- Kubernetes is an open source system
- Kubernetes is highly scalable
- Kubernetes is flexible and adjusts itself to complexity of requirements

### 1.7.1 What is Kubernetes?

Following are the key details about Kubernetes:

- It is an open source container orchestration engine
- Helps in upscaling of containerized applications
- Helps in the management of the containerized applications pre and post upscaling
- Automates the deployment procedures



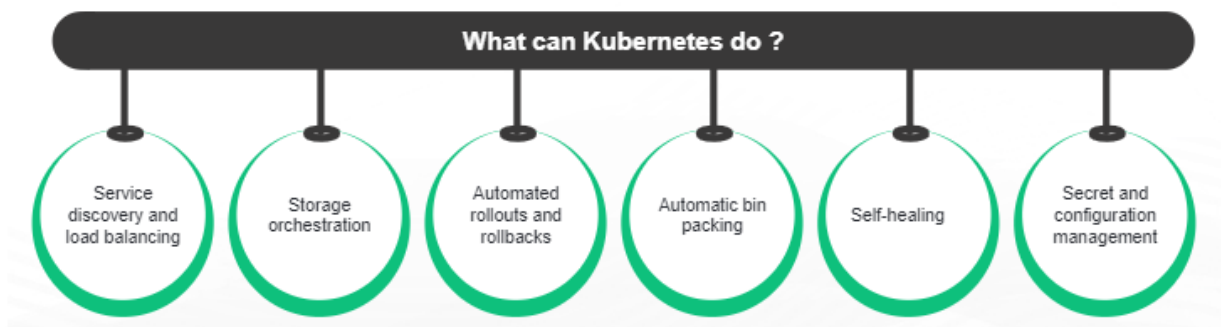
### Facilitator Notes:

Inform the participants about the topics that they will be learning in this module.

Following are the key details about Kubernetes:

- It is an open source container orchestration engine
- Helps in upscaling of containerized applications
- Helps in the management of the containerized applications pre and post upscaling
- Automates the deployment procedures

## 1.7.2 Test Machines — Kubernetes



### Facilitator Notes:

Inform the participants about the importance of Kubernetes.

Kubernetes uses its own IP addresses to expose a container. It also uses the DNS name sometimes. Sometimes a container has to deal with a lot of traffic. Especially this happens, when we upscale. Upscaling from 1 to few containers is fine, but when upscaling becomes beyond the complexity level, e.g., 50 or 100 containers, then the traffic handling becomes a challenge. Kubernetes handles load balancing and distributes the network traffic to respective channels which interact (here channels mean other containers in the system). This makes the deployment stable.

Kubernetes gives us the option to choose any storage system let it be local or public for mounting. And this is an automation process.

Kubernetes primary advantage is the interaction between containers when the quantity and complexity is high. And it controls the state of the containers. Also, it can remove existing containers and deploy their mechanism in a newly appointed container and allocate the resources too. And all this works in an automated way.

Another really great advantage of Kubernetes is optimum use of resources. Let's say you have a certain allocation of CPU memory usage and RAM usage. Kubernetes utilizes the resource available by fitting the containers to the nodes to not overload the system.

Kubernetes restarts the containers which have failed. Reboots them to verify if it is in healthy state to operate and ready to serve the clients or not. Sometimes it removes the containers and replaces them with healthier ones.

Kubernetes is secure with password, SSH key protocols and Oauth tokens.



## 1.7.3 Introduction to Helm

Following are the key details about Helm:

- Helm is a package manager for Kubernetes
- Gives developers and operators to package, configure, deployment of applications and services to Kubernetes clusters in an easy way



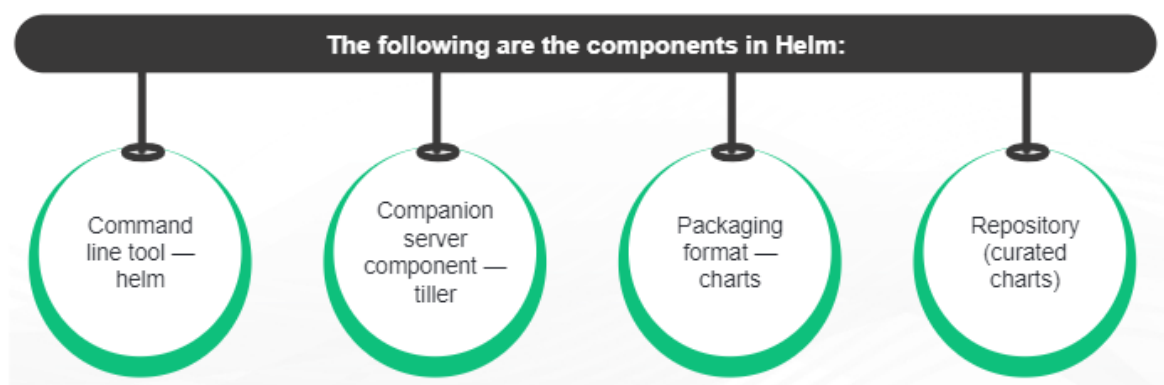
### Facilitator Notes:

Inform the participants about Helm.

Helm can do the following:

- Install software
- Install software dependencies in automation approach
- Software upgradation
- Deployment of softwares
- Connect with repositories and fetch software packages

## 1.7.4 Components in Helm



### Facilitator Notes:

Discuss the components in Helm.

**Helm** basically is a command line tool. It is an application manager running on top of Kubernetes, and it's a huge paradigm shift from traditional server side ways of handling and managing applications, but essentially it is a tool. Like Python has pip or Debian has apt, so helm almost has the same basic features as many of the other package managers.

**Tiller** is a very important component. It runs on Kubernetes cluster. It receives all its commands from helm command line tool and it helps in configuration and deployment of software releases on the cluster.

**Charts** are nothing but packages of helm. Charts mostly have yaml configuration files for example Chart.yaml, requirements.yaml, values.yaml

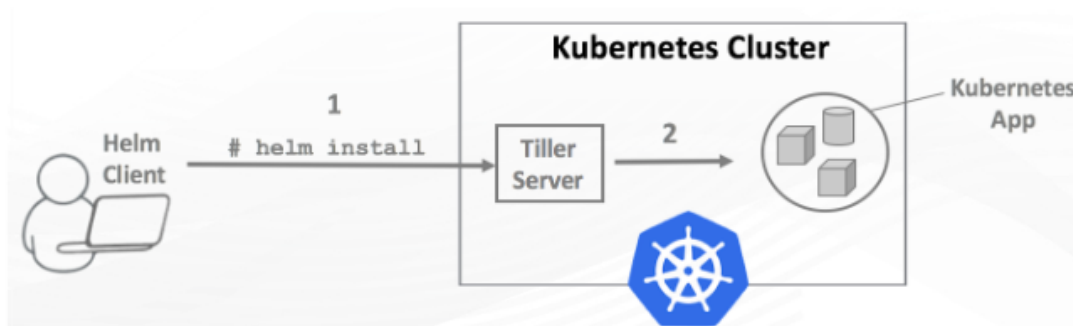
Also there are other directories inside a Chart such as Charts/ , templates/ , etc.

There could be some other files as well. A license file such as LICENSE and a readme file such as README.md

**Repositories** It is a simple HTTP site. It needs the index.yaml file and also .tar.gz packaged charts. And helm command helps in packaging charts by use of various subcommands. And these subcommands also help in creation of the necessary index.yaml file. There are certain default repositories that helm comes with. One such very popular repository is stable. Stable points to (<http://kubernetes-charts.storage.googleapis.com>). This has a huge cluster of a lot of Google APIs which can be extrapolated during execution.

### 1.7.5 Test Machines — Installation and Configuration of Helm Package Manager

- Pre-requisites
- Helm Installation
- Tiller Installation



#### Facilitator Notes:

Inform the participants about the installation and configuration of helm package manager.

Pre-requisites:

- A running Kubernetes cluster
- The Kubectl command-line tool installed in your machine

Helm can be installed through binary as well as Script.

#### Step1: Helm Installation

Helm has an installer script that will allow automatically to grab the latest version of the Helm Client and install it locally.

You can fetch that script, and then execute it locally by running below command

```
#wget https://git.io/get_helm.sh
#chmod 700 get_helm.sh
# ./get_helm.sh
```

## Step2: Tiller Installation

Tiller is a companion to the helm command that runs on your cluster receiving commands from helm and communicating directly with the Kubernetes API to do the actual work of creating and deleting resources.

To give Tiller the permissions it needs to run on the cluster, we are going to make a

### Step1: Create the tiller service account

```
#kubectl -n kube-system create serviceaccount tiller
```

### Step2: Bind the tiller service account to the cluster-admin role

```
#kubectl create clusterrolebinding tiller --clusterrole cluster-admin -serviceaccount=kube-system:tiller
```

**Step3:** Now you can run helm init, which installs Tiller on our cluster, along with some local housekeeping tasks such as downloading the stable repo details:

```
#helm init --service-account tiller
```

To verify that Tiller is running, list the pods in the kube-system namespace:

```
#kubectl get pods --namespace kube-system
```

## 1.7.6 Test Machines — Terraform

### Basic Introduction to Terraform

- Tool used for building, changing and versioning infrastructure
- Terraform is a Product of Hashicorp
- Manages service providers and custom in house solutions



### Facilitator Notes:

Inform the participants about Terraform.

Terraform is a product of HashiCorp. It allows infrastructure for variety of providers — Amazon Web Services, Microsoft Azure, Google Cloud, Digital Ocean and OpenStack.

Terraform can manage anything with an API. It uses a simple declarative programming language called Hachiko Configuration Language also known as HCL and to deploy and manage the infrastructures it uses few CLI Commands.

## 1.7.7 Test Machines – Terraform(Sample Example for Infrastructure as Code)

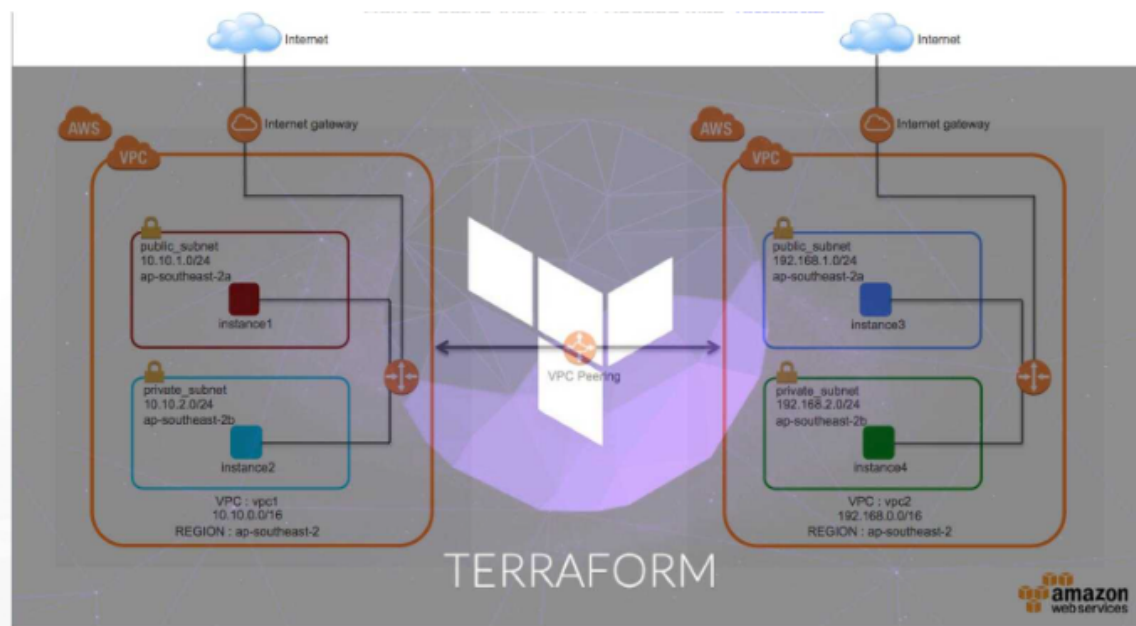


Image Reference: <http://vcloudynet.blogspot.com/2017/03/how-to-setup-vpc-peering-with-terraform.html>

### Facilitator Notes:

Inform the participants about the sample example for infrastructure as code.

Let's say we are trying to setup a Cloud infrastructure on AWS.

We have 2 VPC and 4 Virtual machines in different subnets and we wanted to find necessary security policy between subnets, between virtual machines. We need to do VPC peering. We need to setup the internet gateway and routing so that the VM can go out to the internet and so on. Normally, we will log into Amazon Web Management console and we will click around the WebUI and create the necessary resources and build the infrastructure. All this is a very manual process.

Instead of doing it in a manual way, we can pick one of the IC tools Terraform and it uses HCL, it also supports JSON syntax. Then we write code to define and provision and manage the cloud infrastructure we want. By doing this we automate the whole provisioning and the deployment process very quickly.

Terraform is agnostic to underline platform supported by the resource providers. Example: ec2 instances, virtual machines, virtual private cloud VPC, load balancer, security groups, docket containers, etc. The provider is responsible for underlying API interaction with and exposing their resources and Hashicorp works with a lot of light providers. It includes Version control systems such as GitHub, BitBucket, GitLab. Monetary systems like Digital and Liberado.

For more providers list please head to Terraform Website (<https://www.terraform.io/intro/index.html>).



## 1.7.8 Approaches of Terraform



### Facilitator Notes:

Describe the approaches of Terraform.

**WRITE (infrastructure as Code):** This means writing the code using the language HCL. It is simple and easy to read. This is saved in a file extension .tf

**PLAN(Preview Changes Before Applying):** This is a very important step as it will review the changes which will affect the infrastructure planning. So it reduces mistakes and errors before execution.

**CREATE(Reproducible Infrastructure):** We can use the same code and configuration in multiple places for example, staging environment, production environment, Dev environments and QA Environments. Teams can also have a share of the configurations. Also the Organization can have the share. So, this is called Reproducible Infrastructure.

Provisioning resources in more than one cloud and consuming these resources simultaneously, what terraform can do is it can combine multiple providers consistently in one safe single workflow.

## 1.8 What is Deployment ?

*Putting application files where we need them to do a certain kind of work which is either technical or functional or techno-functional.*

### Facilitator Notes:

Inform the participants about deployment.

We are talking about software and in the software universe, Deployment is like all the hard work done to reach to that stage finally getting accomplished to see how the software application behaves under certain conditions. Deployment can be synonymous with installation, but it is way beyond only installation. It is also not only putting the application files where it is needed to be put but also it is about how that system works in tandem with certain dependent and independent systems which have certain parameters of their own to gauge upon and in sync with software being deployed.

### 1.8.1 What does Deployment Include?

Deployment includes the following:

1. Installation
2. Up-dation
3. Uninstallation
4. Re-installation
1. Installation Track Record

#### Facilitator Notes:

Form the participants about deployment.

**Installation:** You either copy or download certain file or files either from a physical device or from world wide web or from any other source of electronic medium to your local system (it could be a Test System / UAT System / Production System) and you run that software application and in some cases need to do certain configuration (for example configuring System Variables) to make sure the launch of the application is working fine. And also keep a note of the messages across the duration before, during and post installation of the software.

**Up-dation:** Sometimes you need to modify a certain file or a group of files, may be add in case there is a certain patch which has been upgraded or in some cases remove as the upgraded application needs more time to be fixed, so reverting back to the last upgrade. And in some cases completely replace the existing application file or group of files from the system. It could be a small upgrade, a downgrade, Auto update patch with bare minimum changes, a security patch, or completely overhauling the application. Updating a software application has quite a long aperture.

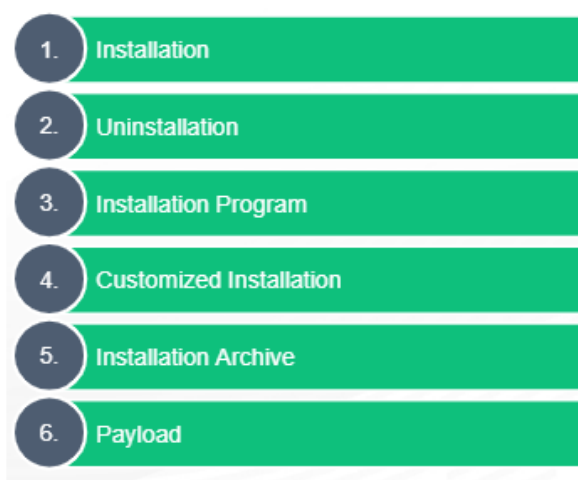
**Uninstallation:** There has always been a debate between deletion and uninstallation. But deletion is a lay man's workaround and uninstallation is a software professional' workaround. Uninstallation removes application files which sometimes are too rigid to not get deleted even by Admin right holders of a System. Thus, uninstallation is pre-dominantly removing application files completely and auto configuring the System so that the application is no more ready for launch or give any offering to launch.

**Re-installation:** It is as simple as first you uninstall it and then install it again. Now why do such an activity is an important question? Sometimes installation might need certain pre-requisites to operate better, sometimes there might have been a network lag, and it led to some misalignment in terms of operational procedures of the installation. In such cases re-installation really comes handy.

**Installation Track Record:** This is mostly about keeping a tab on the number of installation and uninstallation of a particular software in a particular system under whatever circumstances. They can have a version data, installation time data, uninstallation time date, re-installation time data as well.

## 1.8.2 Deployment Concepts

Let's look at the following deployment concepts:



### Facilitator Notes:

Inform the participants about the deployment concepts.

**Installation:** You either copy or download certain file or files either from a physical device or from world wide web or from any other source of electronic medium to your local system (it could be a Test System / UAT System / Production System) and you run that software application and in some cases need to do certain configuration (for example configuring System Variables) to make sure the launch of the application is working fine. And also keep a note of the messages across the duration before, during and post installation of the software.

**Uninstallation:** There has always been a debate between deletion and uninstallation. But, deletion is a lay man's workaround and uninstallation is a software professional's workaround. Uninstallation removes application files which sometimes are too rigid to not get deleted even by Admin right holders of a System. So uninstallation is pre-dominantly removing application files completely and auto configuring the System so that the application is no more ready for launch or give any offering to launch.

**Installation Program:** It is a program which installs a software application on a Local/Remote Machine and performs certain provisioning activities like licensing and configuration. And sometimes it is Operating System Dependent. This is because Environment Variables for MacOS and Windows are different, even directory paths are different.

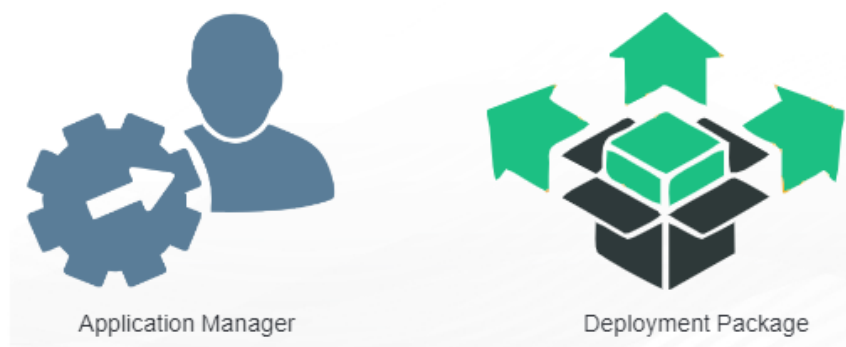
**Customized Installation:** Sometimes an installation comes with pre-defined choices and it totally depends on the UI to acknowledge these choices or ignore them.

**Installation Archive:** Take an example of a .apk folder or a zipped folder which has some source code and other supported files from GitHub repository. We download the zipped file and then extract the downloaded zipped file into a specific folder or directory path. This folder contains all the necessary files for Deployment. This folder is known as Installation Archive

**Payload:** This is actually a set of instructions. Certain set of pre-intra-post requisites about an installation. Where to install, how to install, what minor or major changes need to be done at system level or folder level. Also different operating systems might have different installation protocols. Sometimes a certain extra additive might be there in the payload which can be used only post installation. It could be a security feature or workaround feature. A payload can be in form of a file or multiple files. A folder or multiple folders. It totally depends on how your software has been designed.

### 1.8.3 Deployment Terms

We will look into these two terms:



#### Facilitator Notes:

Inform the participants about the deployment terms.

#### Application Manager:

Before we understand this, let us first go through how it works.

- You downloaded a software from the Internet. Let's say it has been downloaded in a zipped format.
- So you select a Folder in your Directory and extract the zip file.
- Once extracted it might make its own unique folder and inside the folder there will be a lot of files. Some text files might be there. Some code related files. Some license and other related files. And there might be an executable file. So you click on that executable file.
- Deployment begins. At some point of time it might pop up whether you want to do some settings of your own on which all supporting files you need to install or you want default installation. It could be in the form of checkboxes or radio buttons or whatever may be.
- Then it might ask you to enter a Serial License Key to further proceed or ignore installation and then you just go ahead with it.



Have you ever given a thought who regulates all these workaround — yes this is the job of Application Manager. This Manager manages the installation process.

### Deployment Package:

This technically falls under the Application Manager, but as a whole it lists down all the components which can be chosen or else mark it as default. So exactly at that point Deployment Package comes into picture. It can be said as a subset of Application Manager.

## 1.8.4 Relationship Between Deployment and Provisioning

Let's look at the relationship between deployment and provisioning.

1. Frequently happens together
2. Installation program includes both deployment and provisioning
3. Deployment includes standard provisioning steps
4. Some Deployment activities do provisioning during self activation

### Facilitator Notes:

Inform the participants about the relationship between deployment and provisioning.

Even though just by pure definition, deployment and provisioning are separate activities, but in an operational sense they tend to happen together. A lot of tools, infrastructures, process and other software systems use both Deployment and Provisioning as standard practices without differentiating much. As they have such internal logic which has aspects of both to be implemented simultaneously.

- What happens when we begin installation? Some executable files run in the background and once the operation is finished, the application gets launched. This process of putting all the files into the system is known as deployment. Of course, it is not possible unless it accepts and processes a Serial Number which is provided by provisioning. It also configures installed applications, and some of the configuration might include repressing the EULA display and other forms of registration of the software and sometimes auto update checks which falls under provisioning.
- Similarly when we deploy any software application, for run-time usage certain standard preferences are deployed. This is nothing but a provisioning step. And also let's take an example – you launch an application and if that particular application does not find any preferred setting what will it do? Either it will wait for an input or else it will take a default setting which is self-designed. This is also provisioning.

Hence, this is how provisioning and deployment are related.

## In a nutshell, we learnt:

In this module, you have learned:

1. What is Provisioning – Basic Definition
2. What is Provisioning – Software Definition
3. Concepts of Provisioning
4. Why Provisioning Should be Exclusive
5. Configuration Management
6. Configuration Management Tools
7. Why Provisioning is not Configuration Management
8. Provisioning Tools
9. Test Machines for Provisioning
10. Deployment
11. Relationship between Deployment and Provisioning



### Facilitator Notes:

Share the module summary with the audience.

Ask the participants if they have any questions. They can ask their queries by raising their hands.