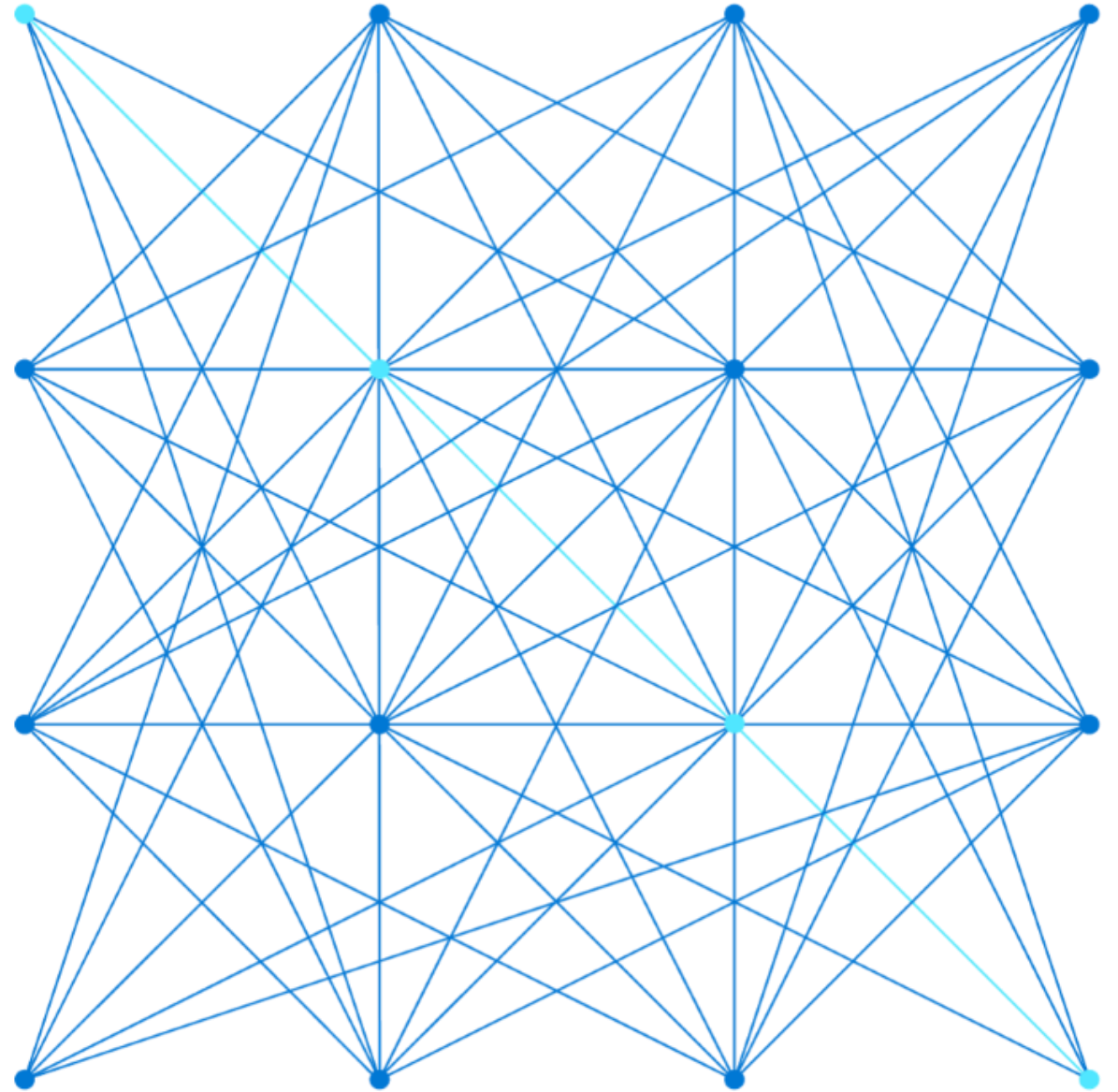


AZ-400.00

Module 14: Third party Infrastructure as Code Tools Available with Azure



Lesson 01: Module overview



Module overview



Lesson 1: Module overview



Lesson 2: Chef



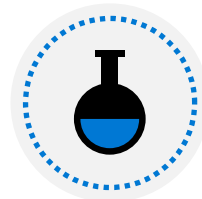
Lesson 3: Puppet



Lesson 4: Ansible



Lesson 5: Terraform



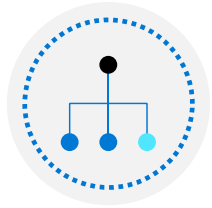
Lesson 6: Labs



Lesson 7: Module Review and Takeaways

Learning objectives

After completing this module, students will be able to:



Describe the basics of 3rd party tools such as Chef, Puppet, Ansible and Terraform in the context of how they can be used to deploy and configure resources in Azure

Lesson 02: Chef



What is Chef?

Chef Infra is an *infrastructure* automation tool that is used for deploying, configuring, managing, and ensuring compliance of applications and infrastructure

Chef Infra Components:



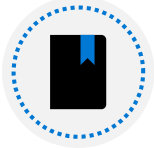
Chef Server – The management point, which has two options for the Chef Server: A hosted solution, and an on-premises solution



Chef Client (node) – A Chef agent that resides on the servers you are managing



Chef Workstation – The Admin workstation where you create policies and execute management commands. You run the **knife** command from the Chef Workstation to manage your infrastructure



Cookbooks – Policy which defines the scenario we are applying, contains recipes, written in Ruby



Recipes – Basic configuration unit, written in Ruby

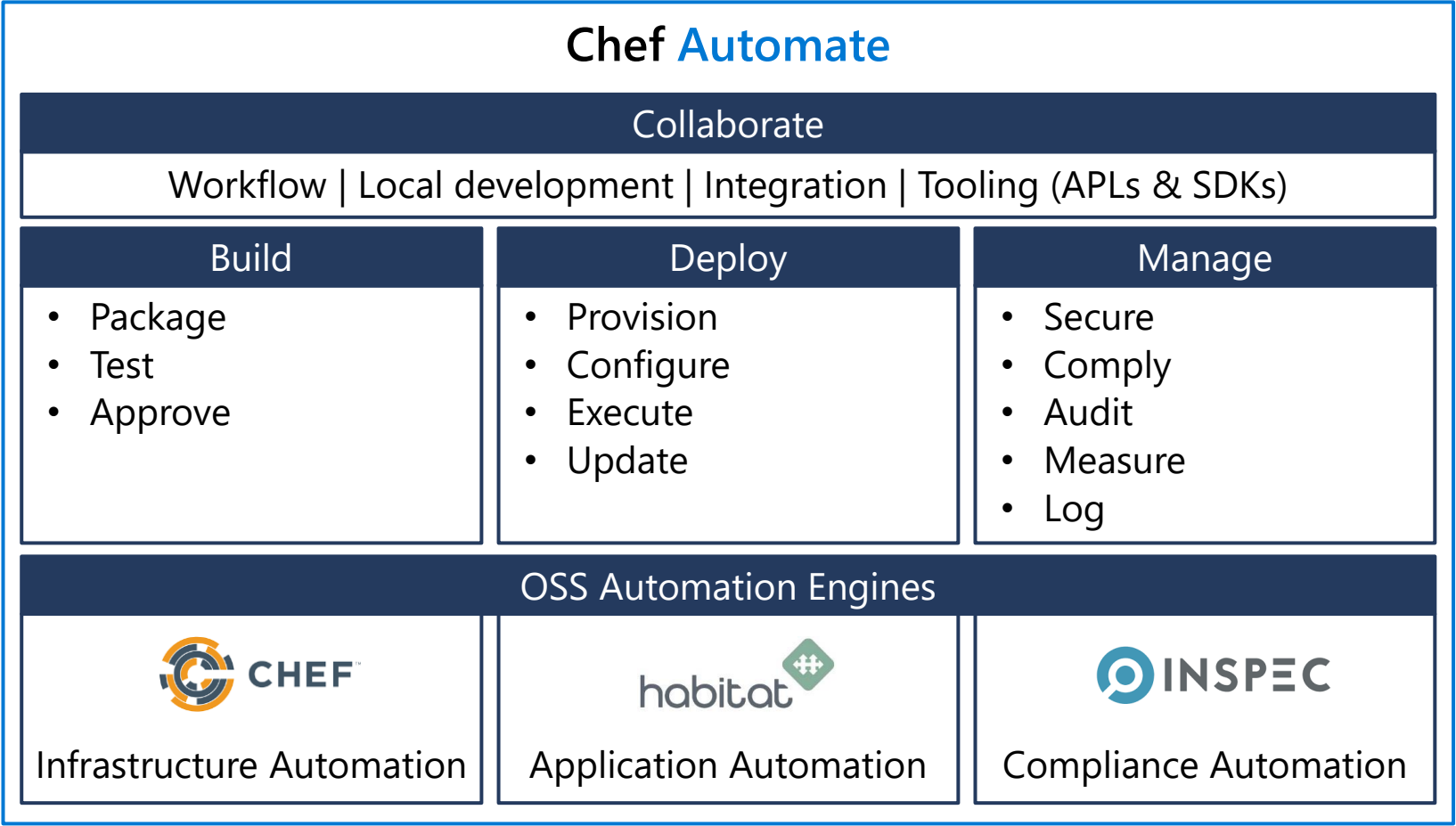
Chef Automate

Chef Automate is a Chef product that allows you to **package** and **test** your applications, and **provision** and **update** your **infrastructure**. It **ensures node visibility and compliance**

Available directly on Azure through the Marketplace

Integrates with Chef [Habitat](#) and Chef InSpec

Chef [InSpec](#) is pre-installed in Azure Cloud Shell



Chef Cookbooks

A *cookbook* is a set of tasks that you use to configure an application or feature

Contains recipes and defines a scenario and everything required to support it

Create a cookbook:

Before creating a cookbook, you first configure your Chef Workstation by setting up the Chef Development Kit on your local workstation

You can download and install the Chef Development Kit from <https://downloads.chef.io/chefdk>

To create a cookbook, you can use the ***chef generate cookbook*** command

After you create a cookbook, you can then create a Role. A *Role* defines a baseline set of cookbooks and attributes that you can apply to multiple servers

Chef Knife Command

Knife is a command that's made available as part of the **Chef Development Kit** installation

Provides a wide range of management and configuration commands for cookbooks, clients, environments, roles, users and many more

Use the Knife command to perform tasks such as:

Manage cookbooks and recipes using the *Knife Cookbook* command

Create a role to define a baseline set of cookbooks and attributes that you can **apply to multiple servers** using the *Knife role* command

Bootstrap a node or client and assign a role using the *Knife Bootstrap* command

Manage clients using the *Knife client* command

(#mw: Have look here: [Knife Azurerm \(chef.io\)](https://chef.io), don't use *Chef Knife* commands as those are ASM based)

Lesson 03: Puppet



What is Puppet?



Puppet is a deployment and configuration management toolset

Puppet operates using a client-server model

Puppet components:



Puppet Master. Responsible for compiling code to create agent catalogs



Puppet Agent. The machine (or machines) managed by the Puppet Master



Console Services. Web-based user interface for managing systems



Facts. Metadata related to state

Deploying Puppet in Azure

There are two options for deploying Puppet in Azure:

Azure Marketplace:

Puppet Enterprise is available to install directly into Azure using the [Azure Marketplace](#)

The Puppet Enterprise image allows you to manage up to 10 Azure VMs for free, and is available to use immediately

Azure virtual machines:

Install a Linux VM in Azure and deploy the Puppet Enterprise package manually

Manifest files

#mw - [Example](#)

Puppet defines state in manifest files known as *Puppet Program* files.

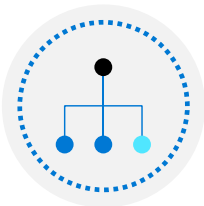
Puppet program files have the following elements:



Class – A bucket that you put resources into. That class then becomes an entity that you can use to compose other workflows



Resources – A single elements of your configuration that you can specify parameters for



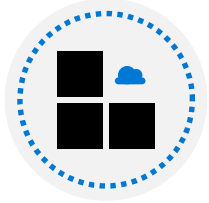
Module – The collection of all the classes, resources, and other elements of the Puppet program file in a single entity

#mw – you can find many modules [here](#) and an example of using a module [here](#)

Lesson 04: Ansible



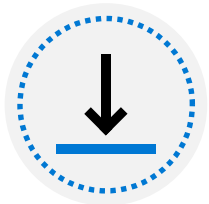
What is Ansible?



Ansible is an open-source platform that automates cloud provisioning, configuration management, and application deployments



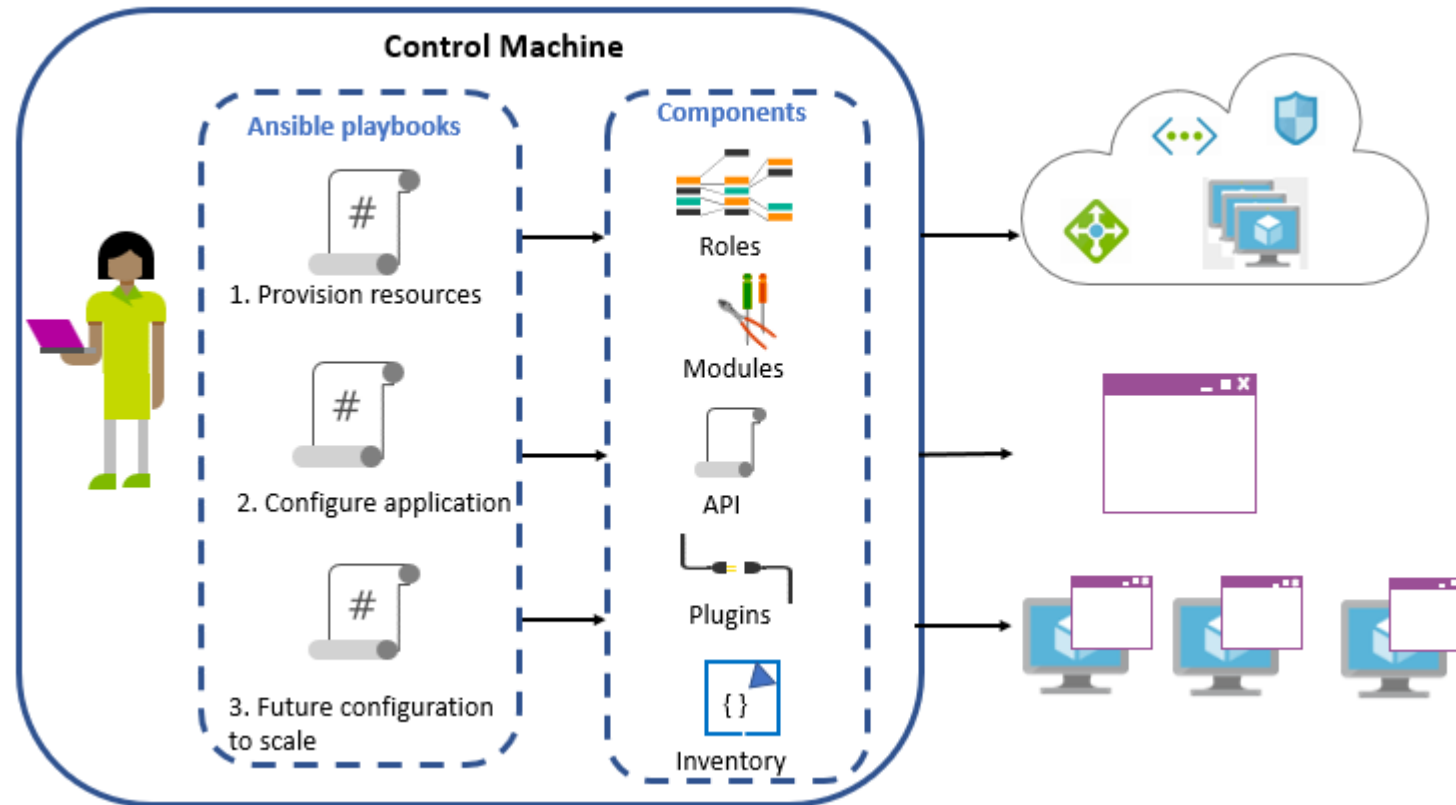
Allows you to automate deployment and configuration of resources in your environment such as virtual networks, storage, subnets, and resources groups



Unlike Puppet or Chef, Ansible is agentless, so you do not have to install software on the managed machines

Ansible workflow

The following workflow and component diagram outlines how playbooks can be run in different circumstances, one after another



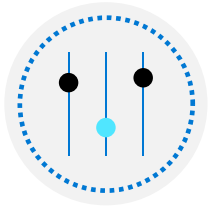
1. Provision resources

2. Configure the application

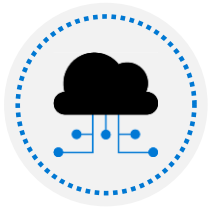
3. Manage future configurations to scale

Ansible components

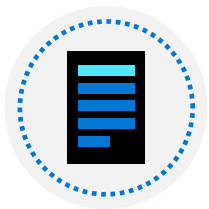
Some of the core components of Ansible include:



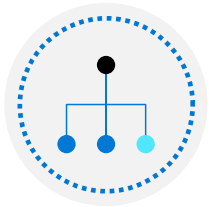
Control Machine. This is the machine from which the configurations are run



Managed Nodes. These are the devices and/or machines and environments that are being managed



Playbooks. Playbooks are ordered lists of tasks, written in YAML, that have been saved so you can run them in the same order repeatedly



Modules. Ansible works by connecting to your nodes and then pushing out to the node's small programs (or *units of code*), called *modules*. *Modules* are the units of code that define the configuration. They are modular, and can be re-used across playbooks

Installing Ansible

To enable a machine to act as the control machine from which to run playbooks, you need to install both Python and Ansible

Python:

Must install either Python 2 (version 2.7), or Python 3 (versions 3.5 and higher)

Ansible:

Only need to install Ansible on one machine, which could be workstation or a laptop—you can manage an entire fleet of remote machines from that central point. Can be Linux, macOS or Windows

No database is installed as part of the Ansible setup

No daemons are required to start or keep running

Ansible on Azure

There are several ways you can use Ansible in Azure

Azure marketplace:

Must install either Python 2 (version 2.7), or Python 3 (versions 3.5 and higher)

Azure virtual machines:

Only need to install Ansible on one machine, which could be workstation or a laptop—you can manage an entire fleet of remote machines from that central point. Can be Linux, macOS or Windows(#mw?)

No database or daemons are installed as part of the Ansible setup

Azure Cloud Shell:

Ansible is installed by default in Azure Cloud Shell

(#mw? Ansible runs on Linux, so to run it on Windows you'd need WSL or something similar)

Playbook structure

Playbooks manage configurations of and deployments to remote machines

Declarative (#mw: [from your notes](#))

Structured with YAML

Include detailed information regarding the number of machines to configure at a time

Demonstration: Run Ansible in Azure Cloud Shell

Create a resource group in Azure using Ansible in Azure Cloud Shell with bash. Azure Cloud Shell, has Ansible pre-installed, so you do not have to install or configure anything to be able to run Ansible

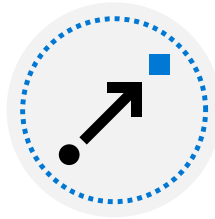


You can complete this walkthrough task by completing the steps in the course content, or you can simply read through them, depending on your available time



(skip) Demonstration: Run Ansible in Visual Studio Code

Install the Ansible extension in Visual Studio Code; create an Ansible YAML file to create a virtual machine in Azure; run the YAML file from within Visual Studio Code; verify creation of the virtual machine in Azure



You can complete this walkthrough task by following the steps outlined below, or you can simply read through them, depending upon your available time



(#mw – you won't be able to do this because the extension ***vscode-ansible*** no longer exists)

Lesson 05: Terraform



What is Terraform?



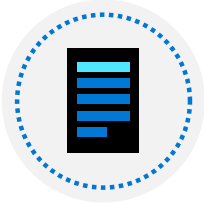
HashiCorp Terraform is an open-source tool that allows you to provision, manage, and version cloud infrastructure

Terraform's command-line interface (CLI) provides a simple mechanism to deploy and version the configuration files to Azure

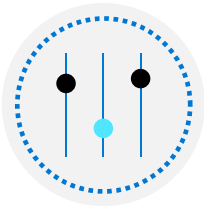
Supports multi-cloud scenarios which enables developers to use the same tools and configuration files to manage infrastructure on multiple cloud providers

Terraform components

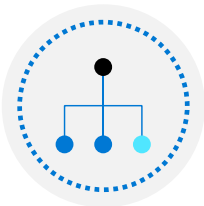
Some of the core components of Terraform include:



Configuration files – Text-based configuration files allow you to define infrastructure and application configuration, and end in the [.tf or .tf.json](#) extension



Terraform CLI – A command-line interface from which you run configurations. You can run command such as **Terraform apply** and **Terraform plan**, along with many others



Modules – Self-contained packages of Terraform configurations that are managed as a group ([#mw: terraform registry](#)) ([#mw: modules for Azure](#))

Terraform on Azure

Options for deploying Terraform in Azure include:

Azure Marketplace:

Offers a **fully-configured Linux image containing Terraform**

Azure virtual machines:

Deploy a Linux or Windows VM in Azure VM's IaaS service, **install Terraform** and the relevant components, and then use that image

Azure Cloud Shell:

Terraform is installed by default in Azure Cloud Shell

Installing Terraform

Options for deploying Terraform in Azure include:

You must install Terraform on the machine from which you are running the Terraform commands.

Can be installed on Windows, Linux or macOS environments

You can download appropriate Terraform install package from

<https://www.terraform.io/downloads.html>

Authenticate Terraform with Azure using:

The Azure CLI

A Managed Service Identity (MSI)

A service principal and a client certificate or secret

Terraform config file structure

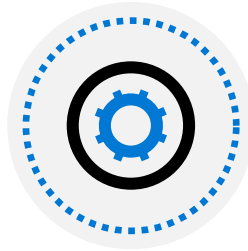
```
# Configure the Microsoft Azure Provider
provider "azurerm" {
  subscription_id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx"
  client_id       = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx"
  client_secret   = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx"
  tenant_id       = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx"
}

# Create a resource group if it doesn't exist
resource "azurerm_resource_group" "myterraformgroup" {
  name      = "myResourceGroup"
  location  = "eastus"

  tags {
    environment = "Terraform Demo"
  }
}
```

Demonstration: Run Terraform in Azure Cloud Shell

Create a create a resource group in Azure using Terraform in Azure Cloud Shell



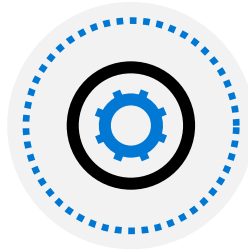
You can complete this demonstration task by following the steps outlined in the course, or you can simply read through them, depending on your available time



(skip) Demonstration: Run Terraform in Visual Studio Code

(#mw – in your own time if you're interested.)

Create a VM in Visual Studio Code
using Terraform

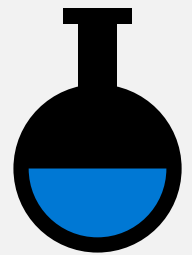


You can complete this demonstration
by completing the steps outlined in the
course, or you can simply read through
them, depending on your
available time



Lesson 06: Labs

We're skipping this.
Nothing specific to Azure DevOps here.
Can do outside class time



Ansible with Azure



Lab overview:

In this lab we will deploy, configure, and manage Azure resources by using Ansible.

Objectives:

- Install and configure Ansible on Azure VM
- Download Ansible configuration and sample playbook files
- Create and configure Azure Active Directory managed identity
- Configure Azure AD credentials and SSH for use with Ansible
- Deploy an Azure VM by using an Ansible playbook
- Configure an Azure VM by using an Ansible playbook

Duration:



We're skipping this.
Can do outside class time

Automating infrastructure deployments in the cloud with Terraform and Azure Pipelines



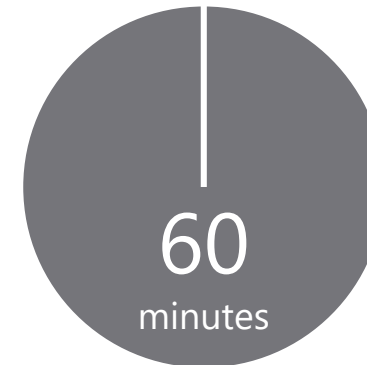
Lab Overview:

In this lab, you will learn how to incorporate Terraform into Azure Pipeline for implementing Infrastructure as Code.

Objectives:

- Use Terraform to implement Infrastructure as Code
- Automate infrastructure deployments in Azure with Terraform and Azure Pipelines

Duration:

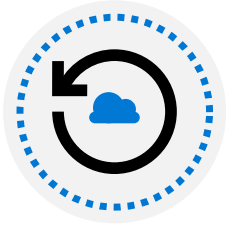


We're skipping this.
Can do outside class time

Lesson 07: Module review and takeaways



What did you learn?



Deploy and configure infrastructure using 3rd party tools and services with Azure, such as Chef, Puppet, Ansible, and Terraform

Module review questions

1 Which of the following are main architectural components of Chef?

2 Which of the following are open-source products that are integrated into the Chef Automate image available from Azure Marketplace?

3 Which of the following are core components of the Puppet automation platform?

4 Complete the following sentence. The main elements of a Puppet Program (PP) Manifest file are class, resource and _____.

5 Which of the following platforms use agents to communicate with target machines?

Module review questions, continued

6 True or false: The control machine in Ansible must have Python installed?

7 Which of the following statements about the cloud-init package are correct?

8 True or false: Terraform ONLY supports configuration files with the file extension .tf

9 Which of the following core Terraform components can modify Terraform behavior, without having to edit the Terraform configuration?