December 2017

# Scripting for System Automation

## Assignment 3

R00169918 Joanna Wojcik

# Table of Contents

# 1. Identify key Infrastructure Automation tools

## 1.1 Configuration Provisioning

- VMware vCenter
- Terraform
- Docker
- Juju

## 1.2 Configuration Management

- Puppet
- Chef
- Ansible
- Vagrant
- XL Deploy
- VMware Tools Configuration Utility

## 1.3 Orchestration

- Ansible
- Kubernetes
- Juju
- Docker Swarm
- Terraform

## 1.4 Continuous Integration

- Jenkins
- TeamCity
- Travis CI
- Bamboo
- CruiseControl

# 2. High level description for each tool in above categories

## 2.1 Configuration Provisioning

- VMware vCenter - VMware vCenter server is a centralized management application that lets you manage virtual machines and ESXi hosts centrally. Facilitates creating, cloning, templating VMs.
- Terraform - cloud agnostic provisioning and orchestration tool for maintaining the state of your infrastructure using a concept called state files. State files are written using Terraform's own domain specific language (DSL) - Hashicorp configuration language, aka HCL.
- Docker - hypervisor based software for providing a lightweight container virtualization environment. Containers do not interact with one another making it perfect tool for spinning up test environments. Reuses host's operating system.

## 2.2 Configuration Management

- Puppet - uses a DSL to define files describing configuration of a machine. Descriptor files are referred to as "Puppet manifest". Manifests describe system resources, their target state, dependencies to be applied, etc. which are then applied against the target systems. Current information about target system is discovered by Puppet by a proprietary utility called Facter.
- Ansible - Developed to simplify complex orchestration and configuration management. Instead of defining its own DSL it utilises YAML specification files.
- Chef - facilitates configuration management using scripts called "recipes". Recipes describe how Chef manages server applications and utilities in a Chef's own DSL. Recipes can be grouped together for cohesion and easier management.

## 2.3 Orchestration

- Kubernetes - Docker container orchestration tool. Takes a YAML file describing state of the environment to end up in. Then Kubernetes will attempt to reach that state as best it can.
- Juju is a python based tool developed for orchestrating applications in cloud environments. Uses Charm files which can be written in any programming language. Charms

would contain the script and a collection of “hooks” - naming conventions to install software, start/stop a service, manage relationships, etc.

- Docker Swarm - alternative tool for managing Docker containers. It is YAML specification file based.

## 2.4 Continuous Integration

- TeamCity - agent based Java written continuous integration offering from JetBrains. Build configurations are managed from central server and individual agents are used to run actual build integration.
- Jenkins - open source, Java written automation server. Allows for extensible job configuration though either server console, Jenkins manifest files or plugin installation.
- Travis CI - Continuous Integration server for GitHub hosted projects. Build configuration is specified in YAML language in files embedded in project allowing for great build flexibility.

# 3. Similarities, Differences and user bases

## 3.1 Configuration Provisioning

### Similarities

All 3 tools support provisioning through scripting files. Since they are used for quite different purposes this is where the similarities end.

### Differences

VMware vCentre instead of defining its own DSL for management scripts can reuse PowerShell syntax. vCentre is used solely for virtual machine management.

Docker is only used to provision containers that do not have their own OS - instead the reuse the host’s one.

Terraform can be used for cloud provisioning of for example an application complete with its dependencies such as a DB offering.

## 3.2 Configuration management

### Similarities

| | Ansible | Puppet | Chef |
|---|---|---|---|
| Configuration through files | Yes | Yes | Yes |
| Proprietary DSL | No | Yes | Yes |
| Grouping of scripts into larger collections | Yes | Yes | Yes |
| Works on *nix server | Yes | Yes | Yes |
| Works on Windows servers | Yes | Yes | Yes |
| Works on MacOSX | Yes | Yes | Yes |
| Runs an agent on target machine | No | Yes | Yes |

### Differences

| Ansible | Puppet | Chef |
|---|---|---|
| Management configuration files are written in YAML | Self discovery of current state on target machine | |
| | System administrator centric | Developer/User centric |
| Cloud provisioning | | |
| Agent-less | | |
| No community or free offering | Puppet Enterprise Free – Limited to 10 nodes. | Chef Basics – Free |

## 3.3 Orchestration

### Kubernetes vs Docker Swarm

Docker and Kubernetes are both tools for container management in a broad sense. While they deliver almost the same functionality due to the skills required to setup and maintenance they will not suit everyone the same. Kubernetes, while more flexible, requires considerable more knowledge upfront to setup node clusters, configure pods, etc. Kubernetes is best suited if your target deployment is large-scale as Docker Swarm currently lacks the maturity in large scale project area is limited to Docker API and currently lacks autoscaling capabilities for services.

### Kubernetes/Docker Swarm vs Juju

Juju is the only of the 3 that is model based. It also the only one that can be used for orchestration of services and application in cloud environments as other 2 tools are container based. Unlike Juju Kubernetes and Docker Swarm are limited to managing docker containers while Juju can be used to orchestrate any service.

### Summary

| | Kubernetes | Juju | Docker Swarm |
|---|---|---|---|
| Docker Container management | Yes | No | Yes |
| Configuration and orchestration managed via scripts | Yes | Yes | Yes |
| Can be managed via UI | Yes | Yes | Yes |
| Flexible setup | Yes | Yes | Yes |
| Can be used for non-container services | No | Yes | No |
| Open Source | Yes | Yes | Yes |

## 3.4 Continuous Integration

### Similarities

Jenkins and Travis CI both utilize file based configuration to describe the build process. As such both provide support for Infrastructure as Code for organizations.
All 3 tools can be used to build and deploy artifacts to cloud based solutions.
Both Jenkins and TeamCity are agent-based tools meaning that the machine doing used to perform the build has to have an agent installed. This can be both advantage and disadvantage. Disadvantage as all agents must go through an upgrade and maintenance. The advantage of having multiple machines with agents means that all build machines comply with a single setup. You're also spreading the load - as individual builds can only be executed by a single agent you're parallelizing the process.

While Travis CI is docker container based you can also setup Jenkins to execute builds on docker containers instead of complete VMs.

### Differences

While build definition in both Jenkins and Travis CI is file based in Jenkins manifests are written in Groovy and Travis CI uses YAML.
Travis CI is only available for projects hosted on GitHub platform. Even at that it wouldn't build .NET applications - for that one would use AppVeyor offering.
As of currently TeamCity cannot be made to utilize containers for performing builds. IT has itself been dockerized however and made available on DockerHub.

# 4. For each category pick 1 tool and give detailed description

## 4.1 Configuration provisioning - Docker

Docker is an open platform for developing, shipping, and running applications. It enables its users to separate the applications from the infrastructure while at the same time the style of management is the same. You can manage your infrastructure with Docker the same way your applications are.
It is achieved through use of containers - loosely separated environment constructs. Loosely since containers due to their size alone do not contain any operating system, therefore they must reuse the OS of the host. Still, the separation between the containers is clear and while some resources may be shared such as a single volume mounted to multiple containers it is very discouraged.
Due to the container separation there can be many different ones running at any given time on a given host. The host itself doesn't have to be physical one as containers are run directly within the host machine's kernel meaning that even virtual machines can be Docker container hosts.
In order to run Docker on your machine you need at minimum 2 things:

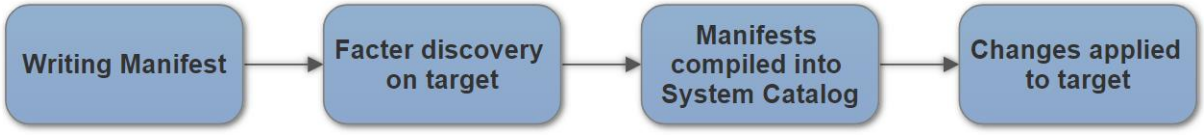
- Access to hypervisor
- Docker client installed

As soon as you're done installing the client you can start downloading containers. For many of the useful services and application there already is a container created. Take TeamCity for example - you don't have to take time to install the server in order to configure your builds anymore - download container from the library instead. You need an MQTT server for latest Internet of Things prototype - there already is a container available for you. And the best thing is that those 2 would before be separated by being installed on at least separate VMs - and that's a lot of storage for OS alone - can now run on a single host side by side and not interfere with each other.

Sample docker commands:
To download a container use docker pull <container-name> e.g. docker pull eclipse-mosquitto

## 4.2 Configuration management - Puppet

Puppet is an open source systems, model-driven management tool for centralizing and automating configuration management. It supports configuration of both *nix systems, such as Linux, Unix, Debian and Windows Server offerings as well as consumer ones like Windows 8.1 or Windows 10.
Being model driven simplifies and limits the programming knowledge required to effectively write Puppet manifests, albeit this benefit may be somewhat reduced by having to learn Puppet's manifest DSL.
Managing configuration mandates that at any given time there is visibility into what the current configuration on the target system is. Otherwise changes cannot be guaranteed to be compatible nor idempotent. Puppet handles target configuration discovery by ts own tool called Facter. At the same time it would be very difficult to maintain cohesion of your configuration if it was done via an unsorted collection of individual scripts. How Puppet deals with this complexity is by compiling DSL written manifests into a System Catalog, from which changes are applied to target systems. The workflow of applying configuration changes is as follows:




Specifying manifest file is relatively easy as examples range from defining a number of operations to be executed consecutively:

```
# execute 'apt-get update'
exec { 'apt-update':                    # exec resource named 'apt-update'
  command => '/usr/bin/apt-get update'  # command this resource will run
}

# install apache2 package
package { 'apache2':
  require => Exec['apt-update'],        # require 'apt-update' before installing
  ensure => installed,
}

# ensure apache2 service is running
service { 'apache2':
  ensure => running,
}

# install mysql-server package
package { 'mysql-server':
  require => Exec['apt-update'],        # require 'apt-update' before installing
  ensure => installed,
}

# ensure mysql service is running
service { 'mysql':
  ensure => running,
}
```

## 4.3 Orchestration - Kubernetes

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. It is described and perceived as the best choice for containerized solutions at scale with support for scaling up to 5000 node-clusters. Core functionalities include:

- Overcoming of Docker API
- Autoscaling on the fly
- Limiting hardware usage to required resources only

At the core of Kubernetes there are a number of key concepts one must come to understand in order to effectively setup and manage it:

- Node
- Cluster
- Pod
- Services
- Ingress

**Node** is a worker machine in Kubernetes which may be a VM or physical server depending on the cluster setup. Each node must have all the services in order to run the pods.

**Ingress** - the only gateway to provide access from Kubernetes to the networks outside the immediate network the cluster is located on.

**Services** - load balancers. They facilitate reliable communication between 2 or more applications residing within pod(s).

**Cluster** -

**Pod** - an abstraction above your application. While it is supported for a pod to contain of more than 1 container it may be discouraged to preserve container boundaries.

Sample Kubernetes configuration file describing an ngnix deployment process with maximum of 3 instances:

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

## 4.4 Continuous Integration - Jenkins

Jenkins is probably the most widely known Continuous Integration server in the industry right now. Better still, it supports Continuous Deployment through it's Declarative Pipeline syntax. Despite being written in Java it supports building and integration of projects on many platforms, including .NET. It is flexible, highly configurable and it integrates with many of the source control software offerings such as Atlassian's Bitbucket out of the box.

Jenkins is agent based meaning that a tiny program used to control the build status and dependencies will be installed on target build machine. While typically build machines - slaves - were virtual machines, currently one can also setup Jenkins to use Docker containers instead. Jenkins supports both explicit and implicit build configuration. By explicit I mean configuring build parameters through Jenkins UI for a given application. Implicit build management to my mind occurs when each and every application contains a Jenkins manifest through which Jenkins server auto-configured the build job.

Both options have merit but it is my belief that explicitly configuring a job through Jenkins UI is too restrictive and in all likelihood not all developers will have sufficient access privileges necessary to configure some of the more advanced features. Plus, if all your setup is done that way you'll never get to DevOps and you're completely rescinding the ability to utilize benefits of Infrastructure as Code.

Even though Jenkins manifests are written in Groovy one cannot utilize, at least currently, all Groovy compatible syntax. Despite that Jenkins with what it supports at the moment is extremely versatile and widely utilized making it a strong choice for anyone looking to get into Continuous Integration.

Sample Jenkinsfile defining a deployment pipeline with custom notification:

```
import org.mycompany.slack.SlackClient
node('platform') {
    if (env.BRANCH_NAME == "master") {
        checkout(scm)
        try {
            String namespace = "jenkins-production"
            String jenkinsUrl = "jenkins"
            String jenkinsPort = 8700
            echo("JenkinsURL: ${jenkinsUrl}\nJenkinsPort: ${jenkinsPort}")
            sh("ssh-keyscan -t rsa -p ${jenkinsPort} ${jenkinsUrl} >>/home/jenkins/.ssh/known_hosts")
            sh("git remote add wfprod ssh://deploy@${jenkinsUrl}:${jenkinsPort}/workflowLibs.git")
            sh("git push wfprod +HEAD:master --force")
            parallel backend: {
                build job: "be-buildjob-script/develop"
            }, frontend: {
                build job: "ui-buildjob-script/develop"
            }
            SlackClient slack = new SlackClient()
            slack.notifyChannel("#channel-name", "Pipeline pushed to ${namespace} from Bitbucket.")
        } catch (Exception err) {
            throw err
        } finally {
            sh("git remote remove wfprod")
        }
    } else {
        echo "Branch change detected, nothing to do."
    }
}
```

There is no restriction on having to define every build operation in a single script as well - notice how Slack notification is delegated to an external client - which is just another Groovy script defining how to handle communication with Slack.

# References

Juju: https://jujucharms.com/how-it-works

Terraform: https://www.terraform.io/intro/index.html

Puppet: https://puppet.com/

Chef: https://www.chef.io/chef/

Ansible: https://www.ansible.com/

Kubernetes: https://kubernetes.io/

Kubernetes sample file: https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/

Jenkins: https://jenkins-ci.org/

Docker: https://www.docker.com/

Docker Swarm: https://docs.docker.com/engine/swarm/

Infrastrcture tools: https://devopscube.com/devops-tools-for-infrastructure-automation/