Infrastructure Orchestration to Optimize Testing

Rohit Naraparaju

rohit.naraparaju@intel.com

Sajeed Mayabba Kunhi

sajeed.mayabba.kunhi@intel.com

Abstract

The advent of new automated testing frameworks and tools has brought about a new revolution in the testing landscape. When used effectively, these tools and frameworks can considerably reduce manual efforts during testing. However, there still remains an area which deserves some attention and has sufficient scope for optimization. This is the setting up of the test environments.

With virtualization gaining popularity, a new dimension has opened as to how testing infrastructure can be configured. The world has moved from setting up a room full of test servers to virtualized environments. Infrastructure Orchestration is the ability to use this advancement in virtualization to spin up complex multi-node, multi-network test infrastructures at will, perform the test, and remove the infrastructure when done. The ability to do this as part of a Continuous Integration or Continuous Delivery pipeline adds vast value to testing and brings down manual effort tremendously. A team which would otherwise spend days or weeks to setup infrastructure can now quickly add value to testing the product, thereby contributing to product quality.

This paper details how Infrastructure Orchestration and Configuration Management can be used to automate test infrastructure setup to reduce the manual effort and optimizing the test cycle overall.

Biography

Sajeed Kunhi is a senior engineering manager with Intel with nearly twenty years of software engineering experience. He is highly passionate about process automation, product quality and DevOps principles. His expertise in software development, technology domain and managerial experience has helped in building optimized software development lifecycle.

Rohit Naraparaju is software builds/tools engineer with Intel with over eight years of software development experience. He is an ardent believer in DevOps and has dedicated a good amount of his career working on Infrastructure Orchestration, Continuous Integration, Configuration Management, Test Automation, and Release Management. Rohit is currently working on advancing the usage of Configuration Management in various other products within his group.

1 Introduction

Testing plays a vital role in the Software Development Life Cycle (SDLC) that helps in improving the quality and reliability of the software thus ensuring that software does what it is supposed to do. In this new competitive era, a lot of companies are looking at shortening their release cycles from years to months to days. To support this new trend of release, product teams are looking at implementing smaller feature sets which need to be tested thoroughly on a variety of environments, performing various kinds of tests, before the software makes it to the successful release. Therefore, testing plays an important role, and should be introduced in the early stages of SDLC as fixing a bug in critical stages is very expensive.

To support testing from the early stages of development, product teams need to test the new features on various testing environments as soon as they have them developed. Traditionally, the test environments are pre-created and configured as per the product team guidelines. To set up the infrastructure for each test environment could take anywhere from a week to more than month, depending on the infrastructure requirements from the product teams. This delay can cause significant impact to testing activities and delay the release of the product.

In this paper, we present the issues faced with the traditional approach of pre-creating the environments for testing by requesting the infrastructure from IT, and how we had to use the same environments for different kinds of tests based on our experience and learnings from earlier projects.

Our aim is to present how we are solving this problem by following a new approach of automating the Infrastructure Provisioning of test environments at will using various tools. We also present how we have integrated this process as part of our Continuous Integration pipelines to test the product from code check-in to running unit tests, spinning of a new testing environment, deploying the product using the Configuration Management tool, running integration tests, and sending the results of the tests as an email to the subscribers.

2 Problem Statement

Testing an application that involves multiple systems is a cumbersome task. To deliver a reliable product to the customer, we need to perform detailed testing of the independent components, systems, and other systems that integrate with the core product. The first step in performing testing is to come up with the infrastructure needs and configuration of test environments. If the product team requires a basic web server and database server for the set up, then it is easy to set up the environment. However, as the product starts integrating with other systems, the set up becomes complex and setting up an environment manually will take lot of manual effort, time, and human errors when setting up configurations.

3 Problems with the Fixed Environments and how it impacts Testing

In our earlier project, to set up a single testing environment, we required two web servers and one database server. We had to then finalize the number of environments required for the project, as well as deciding on the computing resources required in total for setting up the environments. Once we had all these details, we had to engage with IT to set up a project on their existing Team Foundation Server (TFS) lab environment and get the things finalized.

The process we followed to set up the environment was as follows:

1. System preparation for Web server:
   a. Decide on the OS for the test environment
   b. Create a Virtual Machine (VM) required with specified OS, Network LAN, RAM, CPU Cores, and Disk Space for Web server
   c. Configure Internet Information Services (IIS) for the Windows server
   d. Install the required software for the product
   e. Install all Windows update

   f. Disable the firewall settings
   g. Convert this VM as a template for Web server

2. System preparation for Database server:
   a. Decide on the OS for the test environment
   b. Decide on the MS SQL Server Version and Flavor
   c. Create a VM required with specified OS, RAM, CPU Cores, and Disk Space for Database server
   d. Install the MS SQL Server and Configure the necessary settings required for the project
   e. Install the required software for the product
   f. Install all Windows update
   g. Disable the firewall settings
   h. Convert this VM as a template for Database server
3. Clone two VMs from the Web server template
4. Clone a VM from the Database server template
5. Create an Environment Template by adding the two web server VMs of Web server and the database server VM of Database server
6. Give a unique name to the Environment Template
7. Create an environment from the Environment Template
8. Give a unique name for each environment
9. Login to each VM and change the name of the machines
10. Install Visual Studio Test Agents on each VM and configure them to talk to the TFS Test Controller
11. Take a clean snapshot of VM
12. Repeat the steps 3-8 for each planned environment
13. Deploy the product by running shell scripts from TFS Build Deployment definition as part of the CI process which execute the automated tests.

The above process worked great in the initial stages where the code base was small and few changes were required in configuring the environments. However as the product matured and integrated with multiple systems, we came across changes to versions of software, changing IIS settings, adding new components, etc. for the environments which became very difficult to update over a period of time, often spending a day or two in getting all the environments to the new set of changes.

Disadvantages with pre-created environments:

1. Requires lot of manual configuration and can be error prone
2. Making a small change to the environment configuration would require us to login in to all environments, propagate the change, and take the snapshots
3. Restricted to work with the same configuration settings on all the environments which may vary with the production environments
4. A change in the OS version or SQL Server version meant that we should have to repeat this entire process, which would delay making the test environments available for testing new features
5. If the TFS lab goes down, all the environments go down
6. Maintenance of these environment overtime became time consuming
7. Faced a lot of connectivity issues multiple times with the TFS lab environment affecting testing
8. Upgrade to a new TFS Lab environment version means recreating the entire set up of testing environments as there was no easy way to migrate the existing environments to new versions
9. Need a dedicated resource to take care of these issues, work with IT, and investigate them

10. Space limitations on the storage in the Lab environments
11. Setting up a new environment means provisioning the servers, configuring the servers, and getting the applications to run

4 Approach

Considering the issues that arise during setting up a fixed environment for testing, choosing a right approach for setting up the testing environment at will depends on following process:.

1. Determine if the team will adapt to the new technologies and tool set used to set up the environment.
2. Evaluate a cloud services platform (such as: AWS, VMware, Azure, Openstack, Private cloud) by understanding its capabilities, advantages, costs, documentation, pricing, and support
3. Choose the right Configuration Management tool, Infrastructure Provisioning tool, and Continuous Integration tool by evaluating the tool, team's adoption to new tool, documentation, training, licensing and support.
4. When we understand and realize the importance of Infrastructure Provisioning, Configuration Management, and Continuous Integration the next challenge is to select the appropriate scripting language that supports making calls to the API's exposed by the cloud services platform in order to set up the testing environment.

Based on above discussed points, we give a small overview of Infrastructure Orchestration and Configuration Management and then give a picture of how all the roles need to come together to make this solution successful.

4.1 Infrastructure Orchestration

Orchestration mean arranging, coordinating, provisioning, and managing the complex computer systems, middleware, and services. For example, if a team may not only want to spin up a server, but install the software, configure the database, provision load balancers, and get all these things orchestrated for testing or production purpose, then that is described as Infrastructure Orchestration.

Infrastructure Orchestration addresses the problem of getting infrastructure components to work together as documented in code or in a structured document. In the compute realm, it is often used to spin off a specific OS or a custom image and its resources to get a service up and running. It also addresses the problem at a layer above the configuration management system. Configuration Management systems are good at describing the details of nodes but less good at coordination amongst clusters and complex provisioning tasks. Examples of Infrastructure Orchestration tools are: AWS CloudFormation and Terraform. See details in the "Tools" section below.

From the Developers/Testers perspective, they will use the resources intended for their purpose and release them after they have completed their task. Therefore Infrastructure Orchestration must support assigning and removing the environments from the shared pool. The main goals of Infrastructure Orchestration are to manage the resources, avoid conflicts, and enable user productivity.

Advantages:

1. Cost Savings
2. Manageability
3. With the Infrastructure Orchestration in place, IT can collaborate with product team and come up with new infrastructure topologies which can be later be converted to standardized templates for other teams to consume/integrate with.
4. Product team can spin up multiple environments on an as-needed basis
5. Reduces manual effort of setting up the testing environment
6. Testing environments will be consistent
7. The setup process can be integrated into the Continuous Integration and Delivery processes

4.2 Configuration Management

Configuration Management can be defined as a process that deals with maintaining the hardware and software for a product. It records information about the computer system and updates that information as needed. It is a declarative system for managing the configuration of an instance. These details can be installation of packages, starting/stopping services, and editing config files on the instance. This simplifies engineering efforts in determining which programs are installed and which systems need to be upgraded.

A Configuration Management system allows you to describe the infrastructure pieces in a high-level, Domain Specific Language (DSL) and often provides facilities that abstract the underlying actions being taken. This allows you to track changes to the systems over time through source control, easily move between OS/platforms, quickly modify the state of the infrastructure, and test the changes. In a nutshell, Configuration Management systems do the detail work of making one or many instances perform their roles without the operator needing to specify the exact commands needed to configure the system under test.

Deployment of applications, and maintaining infrastructures are very critical and delicate tasks which were traditionally managed by hand. If the above tasks could be handled by a tool then it would save lot of time and also remove the element of human error. Therefore, this led to the rise of Configuration Management tools which handle the automation of configuration states for the application. Like any other tool, this requires effort to learn, and depends on the skills of the person putting them together to work.

The two most popular Configuration Management tools are Chef and Puppet. Both these tools serve the same purpose and are designed to accomplish the Configuration Management for an application.

Advantages:

1. Faster restoration of the application
2. Increased efficiency of configuring the system
3. Cost reduction
4. Strong security

4.3 Automated Infrastructure Orchestration Workflow

Once we have a clear understanding of the terms Infrastructure Orchestration and Configuration Management, the next step is to establish a workflow around setting up a fully automated infrastructure orchestrated solution.

The diagram below represents the workflow and the jobs performed by the roles.

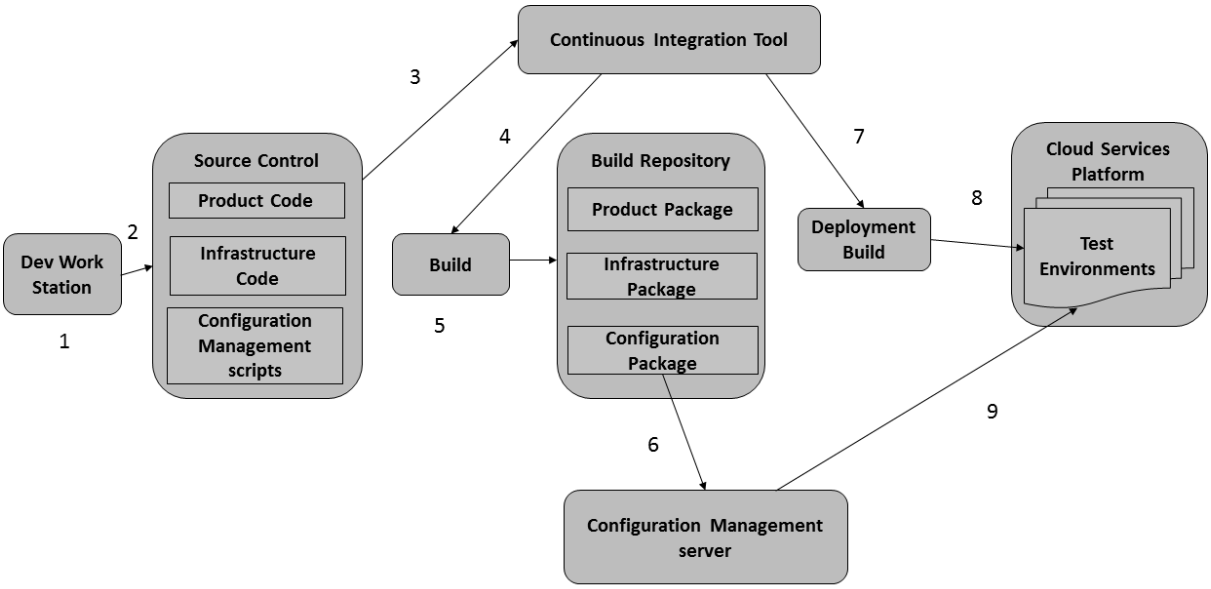

Fig 4.1a: Continuous Deployment Process

1. Write a structured Infrastructure Provisioning and Configuration Management scripts required for setting up the test environment
2. Check-in these scripts to the source control
3. Source control sends a notification to the Continuous Integration tool
4. A Continuous Integration tool kicks off the compile build
5. Compiled build copies the packages to the Build Repository
6. Get the latest Configuration Management scripts from Build Repository and upload to the Configuration Management server
7. Continuous Integration tool kicks off a deployment build
8. Deployment build is responsible for spinning up a test environment dynamically in the cloud service platform provider space
9. After the test infrastructure is up, the deployment build runs the build step where the Configuration Management scripts bring the test environment to the desired state

5 Tools

Based on the points described in the approach section, we have classified the tools required for setting up the infrastructure automatically into four categories:

1. Cloud Services Platforms
2. Infrastructure Provisioning Tools
3. Configuration Management Tools
4. Continuous Integration Tools

Below, we give a brief overview of the most popular tools in the market for each category.

5.1 Cloud Services Platforms

5.1.1 Amazon Elastic Compute Cloud (EC2)

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides scalable computing capacity in the Amazon Web Services (AWS) cloud. It is designed to make cloud computing easier, develop and deploy applications faster thus eliminating the need to invest in hardware upfront.

EC2 web service interface allows you to launch virtual servers, configure capacity, security and network, and manage storage that you need as well as providing complete control of the computing resources. It enables you to scale the capacity both up and down depending on changes in computing requirements thereby allowing you to pay just for the capacity that is used. It also provides developers a lot of tools to handle build failure applications and isolate the common failure scenarios.

Benefits of Amazon Elastic Compute Cloud:

1. Elastic web-scale computing
2. Completely controlled
3. Flexible cloud hosting services
4. Designed for use with other Amazon web services
5. Reliable
6. Secure
7. Inexpensive
8. Easy to start

5.1.2 Microsoft Azure

Microsoft Azure is a public cloud computing platform and infrastructure created by Microsoft for developing, deploying, and managing the applications on a global network of data centers managed by Microsoft. It is widely considered as both Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) which offers cloud services such as compute, analytics, storage, and network thus supplementing the use of on-premise servers.

Facts about Azure:

Flexible – Scale compute resources up and down as needed

Open – Supports almost any language, OS, tool, or framework

Reliable – 99.95% availability and 24 X 7 tech support

Global – Data housed in geo-synchronous data centers

Economical – Only pay for what you use

Microsoft Azure categorizes its services into 11 types:

1. Compute
2. Web and Mobile
3. Data Storage
4. Analytics
5. Networking
6. Media and Content Delivery Network
7. Hybrid Integration
8. Identity and Access Management
9. Internet of Things
10. Development

11. Management and Security

5.1.3 VMware VCloud Air Virtual Private Cloud

VMware VCloud Air Virtual Private Cloud is a multitenant, logically isolated cloud Infrastructure as a Service (IaaS) compute solution that lives on shared infrastructure. It is architected on VMware VSphere design elements that delivers high availability and fully private networking capabilities. It is also a secure, flexible, capable, and expandable IaaS environment that helps to launch new applications onto the cloud.

Benefits:

1. Compatible with an On-Premises VSphere
2. Flexible VM sizing
3. Application Portability
4. Advanced Networking
5. High Availability
6. Cost effective starting point

Features offered by Virtual Private Cloud are:

1. Trusted Hypervisor
2. High Performance
3. Custom VM sizing
4. Tiered storage options
5. Enterprise-grade network virtualization
6. VM lease and quota management
7. Direct console access

5.2 Infrastructure Provisioning Tools

5.2.1 Terraform

Terraform is a tool to build, change, and version the infrastructure such as VMs, network switches, etc safely and efficiently. It can manage existing service providers as well as the custom built in-house providers

Terraform uses a declarative language to describe the resource configurations. The main advantage of Terraform is that it is cloud agnostic and enables multiple providers and services to be combined and composed.

A configuration file tells Terraform the components needed to set up the application. Terraform generates an execution plan that describes the desired state to reach and later executes it to build the desired infrastructure described in the plan. If the configuration changes, Terraform has the capability to detect the changes and create incremental execution plans to be applied to the set up the infrastructure.

Terraform operates at a lower level than the Configuration Management tools. While Configuration Management tools work on the existing resources, Terraform actually builds those nodes by managing the components such as storage, compute instances, networking, DNS entries, etc.

Key features of Terraform are:

1. Infrastructure as code
2. Resource graphs
3. Execution Plans
4. Change Automation

5.2.2 AWS CloudFormation

AWS CloudFormation is a service to model and set up the Amazon Web Services resources. It gives the developers a means to create and manage the resources, provision, and update them in less time and helps them to focus more on developing/testing the applications.

Developers can use the sample templates offered by CloudFormation or create their own templates to describe all the AWS resources like instances, associated dependencies, and runtime parameters required to run the application. Developers don't need to worry about the order in which the provisioning takes place as CloudFormation takes care of this for them. After the AWS resources are deployed, they can modify and update them in a controlled way.

Features of CloudFormation are:

1. Supports a wide range of AWS resources
2. Easy to use
3. Declarative and Flexible
4. Infrastructure as code
5. Customized via parameters
6. Visualize and edit with drag-drop interface
7. Integration ready

5.2.3 VMware vRealize Automation

VMware vRealize Automation is designed to improve the deployment, application management, personalized infrastructure, and compute services thus improving the operational efficiency. It is a software product for unified cloud management that provides the administrators the ability to provision and configure the storage, network, compute resources across private and public cloud platforms. Its key benefit is that we can deploy an automated, on-demand cloud infrastructure thereby giving flexibility. It is also very well integrated with the VMware VCloud Air Virtual Private Cloud platform. It helps in automating the infrastructure set up by leveraging the variety on CLI/APIs exposed by the cloud platform for requesting and configuring the VMs.

5.3 Configuration Management Tools

5.3.1 Chef

Chef is a ruby based Configuration Management tool which can be easily installed and configured to specific needs with little bit of programming knowledge. It is available as a free open source tool as well as a paid enterprise subscription. Chef utilizes a master-agent model. Apart from the master server, it requires a workstation setup that controls the master server. A Chef client would be running on all the nodes which have to be configured.

Chef has the unique concept of cookbooks. A cookbook is the fundamental unit of configuration in Chef. Cookbook is a collection of recipes and associated attributes. Recipes are collections of resources that are written in ruby and Attributes provide specific details of the node. The Chef client is responsible for pulling the cookbooks from master server and running the specific configurations on respective nodes to bring them to a desired state. Chef has a lot of features such as text-based search and support for multiple test environments. Its command line interface, testing mode, and large database make it ideal for companies that need to store records for a large number of computers. The capability to install or even create different modules makes this one of the most customizable Configuration Management options.

5.3.2 Puppet

Puppet is a ruby based Configuration Management tool like Chef. It is also available as free open source and paid enterprise versions. The configuration code is written using Puppet's Domain Specific Language (DSL) and wrapped in modules. Puppet also runs an agent on all the nodes to be configured and it pulls the compiled module from the Puppet server and install required software

packages specified in the module. Role-based access, orchestration, automation, and event inspector are some important features on Puppet enterprise. It is also designed to work on a variety of platforms.

5.3.3 Ansible

Ansible is relatively a new tool in the market when compared to Chef and Puppet. It is an agentless tool that use Secure Shell (SSH) to provide a simple Configuration Management tool yet with strong security capabilities. Configuration modules in Ansible are call playbooks which are written in Yet Another Markup Language (YAML). It is relatively easy to learn and write. Besides Configuration Management, it offers workflow monitoring and automating application deployment for updates like an orchestration tool. Ansible focuses on five principles:

1. Small learning curve
2. Ease of use
3. Automation everything
4. Efficiency
5. Strong security

5.4 Continuous Integration Tools

5.4.1 Jenkins

Jenkins is an open source, cross-platform, Continuous Integration and Continuous Delivery application that increases the productivity. Continuous Integration is the practice of running the automated tests on a non-developer machine every time new code is pushed into a source repository.

It has tremendous advantage of always knowing if all tests work and getting fast feedback. The fast feedback is important so you always know right after you broke the build (introduced changes that made either the compile/build cycle or the tests fail) what you did that failed and how to revert it.

If you only run your tests occasionally the problem is that a lot of code changes may have happened since the last time and it is rather hard to figure out which change introduced the problem. When it is run automatically on every push then it is always pretty obvious what and who introduced the problem. Some important features of Jenkins include easy to install, and configure, extensibility, and support for a large number of plugins.

5.4.2 TeamCity

TeamCity is a Java-based Continuous Integration server package from JetBrains. It is easy to install and configure. Though it is a java based, it supports a lot of .NET development shops. TeamCity server is the main component and comes with a browser hosted interface. The interface is designed to be the primary source to administer TeamCity users, agents, projects, and build configurations.

TeamCity provides project status and reporting dashboards which can be subscribed by the interested users and project stakeholders. It also provides build progress, drill down detail, and history information on the projects and configurations.

TeamCity installation also comes with a system tray utility to get the information of build status and progress. Build notifications such as passed/failed are received in this tray instead of email.

Some important features of TeamCity include easy setup, use, configurability, and being well-documented. The tool also has the ability to integrate with wide range of tools and technologies.

6 Solution

Our new project is based on the Big Data platform technologies which usually require a large infrastructure cluster to be up and running for supporting the deployment of an application. The main

aim of this new project is to store different types of threat events coming from McAfee agent into a scalable distributed database and move away from storing into Microsoft SQL Database Server.

In the new project we had to use the Cloudera Distribution including Apache Hadoop (CDH) platform that provides the Apache Hadoop based software, support, and services. Apache Hadoop is an open-source software for reliable, scalable, distributing computing. It is a framework that supports distributed processing of large data sets across clusters of computers. It is mainly designed to scale up from a single machine to thousands of machines, offer local computation, storage, handle failures, and deliver highly-available service on top of a cluster of computers.

As Cloudera has numerous products in the offering, we handpicked a few of the products that were useful for designing our solution. In order to make the solution more reliable and robust, we needed a fairly big infrastructure set up for deployment and testing.

For setting up a single testing environment, we required about 12 VMs with a higher configuration specification per VM (such as higher memory, CPU cores, and larger disk spaces). If we had taken a traditional approach of setting up this environment by going through IT channel, it could have taken anywhere from a day to weeks just to get approved for infrastructure. It could have taken another few days to configure and deploy the application on the environment manually. Once we had the environment configured manually, validating the environment and fixing the issues could take anywhere from few hours to days due to the number of VM's involved.

Since setting up a test environment such as this was very time consuming, and required a lot of manual intervention, and expense, we decided to adopt the approach of Infrastructure Provisioning combined with a Configuration Management tool to set up the environment of testing.

We selected the following technologies and tools to develop the Infrastructure Provisioning solution that helps in spinning up a testing environment for our project based on the technical expertise within the team.

1. In-house VMware VCloud Virtual Private Cloud as Cloud Services Platform
2. Chef for Configuration Management
3. Jenkins for Continuous Integration
4. VMware vRealize Automation (vRA) for Infrastructure Provisioning
5. Python scripting to bundle the vRA API calls, Chef knife bootstrap commands, and running automated tests

To set up and configure this infrastructure automatically, we followed these steps:

1. Create a job in Jenkins for provisioning the infrastructure
2. Write Chef cookbooks and check-in to source control
3. Write a python script for provisioning the VMs using vRA API's and bootstrapping the Chef commands
4. Check-in the python script to the source control
5. Configure the Jenkins job to call this python script that provisions, configures the infrastructure, and run automated tests
6. Configure a post build deployment step in Jenkins to publish the test results and send out an email notification to the interested subscribers.

Python script layout

We chose python because of the features it offers, its simplicity, and ease of maintenance. Also, most of the members in the team were comfortable with this scripting language. This script handles the core functionality of setting up the testing environment. Below is the layout of the script

1. Read the VM configurations specified in the config file
2. Call the vRA API's exposed by the Private Cloud for requesting the VMs

3. Check the provisioning status for the VMs
4. Store the result of the provisioned VM in a JSON file
5. Call the Chef knife bootstrap command to configure the respective application by passing the Chef's recipe list for the VM

This python script can be invoked by passing it as a build step in Jenkins as part of continuous deployment or from a developer work station with the required parameters for setting up the infrastructure required for the testing environment.

Advantages

With the above tools, technologies, and scripts in place and configured to the needs of the product team, the team was able to

1. Spin up environments on an as-needed basis by developers/testers
2. Integrate Infrastructure Provisioning, Configuration Management, and Automated Tests into the Continuous Integration process
3. Spin up environments with different configurations in less time
4. Enable immediate developer testing, rather than waiting for a specific stable build
5. Run multiple builds performing various tests on various environments
6. Save time that would have been spent on manual intervention
7. Create environment configurations consistent across all environments

7 Conclusion

This paper presents our experience with how fixed environments can cause delay and impact testing activities, and how we wanted to address this problem in our new project by moving to Infrastructure Provisioning combined with Configuration Management tools to set up the environments for different kinds of tests on an as-needed basis. Testing is one of the key components of Software Development Life Cycle (SDLC) and it needs to be started from the beginning of a project, and not left to the end.

In order to support testing activities, many factors need to be considered, and the one that tops the list is infrastructure availability. Once we have this issue resolved, then the product teams can spin up multiple testing environments as needed, to perform various tests (i.e. BVT, Regression, Integration, System tests, etc), optimize the testing phase, and qualify the product for production grade. When we get the entire process correct, a product team may have the capacity to release product to production more often, and in shorter periods of time, leading to increased customer satisfaction.

References

Upguard, “7 Configuration Management (CM) Tools You Need to Know About”, https://www.upguard.com/articles/the-7-configuration-management-tools-you-need-to-know

DEVOPSCUBE, “10 Devops Tools for Infrastructure Automation”, http://devopscube.com/devops-tools-for-infrastructure-automation/

Adam Bertram, June 24, 2015, “Choosing The Right Configuration Management Tool”, http://www.tomsitpro.com/articles/configuration-management-tools,2-920.html

Quora, “Infrastructure Orchestration and Configuration Management”, https://www.quora.com/How-do-I-learn-orchestration-and-configuration-management

“Microsoft Azure”, http://searchcloudcomputing.techtarget.com/definition/Windows-Azure

Spencer Towle, August 31, 2015, “Microsoft Azure Explained: What It Is and Why It Matter”, http://ccbtechnology.com/what-microsoft-azure-is-and-why-it-matters/

“What is Python? Executive Summary”, https://www.python.org/doc/essays/blurb/

“VMware Virtual Private Cloud”, http://vcloud.vmware.com/service-offering/virtual-private-cloud

“VMware vCloud Air Virtual Private Cloud”, http://www.vmware.com/cloud-services/infrastructure/vcloud-air-virtual-private-cloud/

Joao Miranda, August 4, 2014, “Terraform Infrastructure”, https://www.infoq.com/news/2014/08/terraform

“Terraform”, https://www.terraform.io/intro/

“Cloud Formation”, https://aws.amazon.com/cloudformation/

“Cloud Formation”, http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

“VMware vRealize Automation”, http://searchservervirtualization.techtarget.com/definition/VMware-vCloud-Automation-Center-vCAC

“VMware vRealize Automation”, http://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/product/vcloud/vmware-vrealize-automation-datasheet.pdf

Stephen Ritche, November 20, 2012, “TeamCity vs Jenkins: Which is the Better Continuous Integration (CI) Server for .NET Software Development?”, https://www.excella.com/insights/teamcity-vs-jenkins-better-continuous-integration-server

“Amazon Elastic Compute 2”, http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

“Amazon Elastic Compute 2”, https://aws.amazon.com/ec2/

“FountainHead, March 16, 2009”, http://fountnhead.blogspot.com/2009/03/what-is-infrastructure-orchestration.html

https://en.wikipedia.org/

http://www.cloudera.com/

http://hadoop.apache.org/